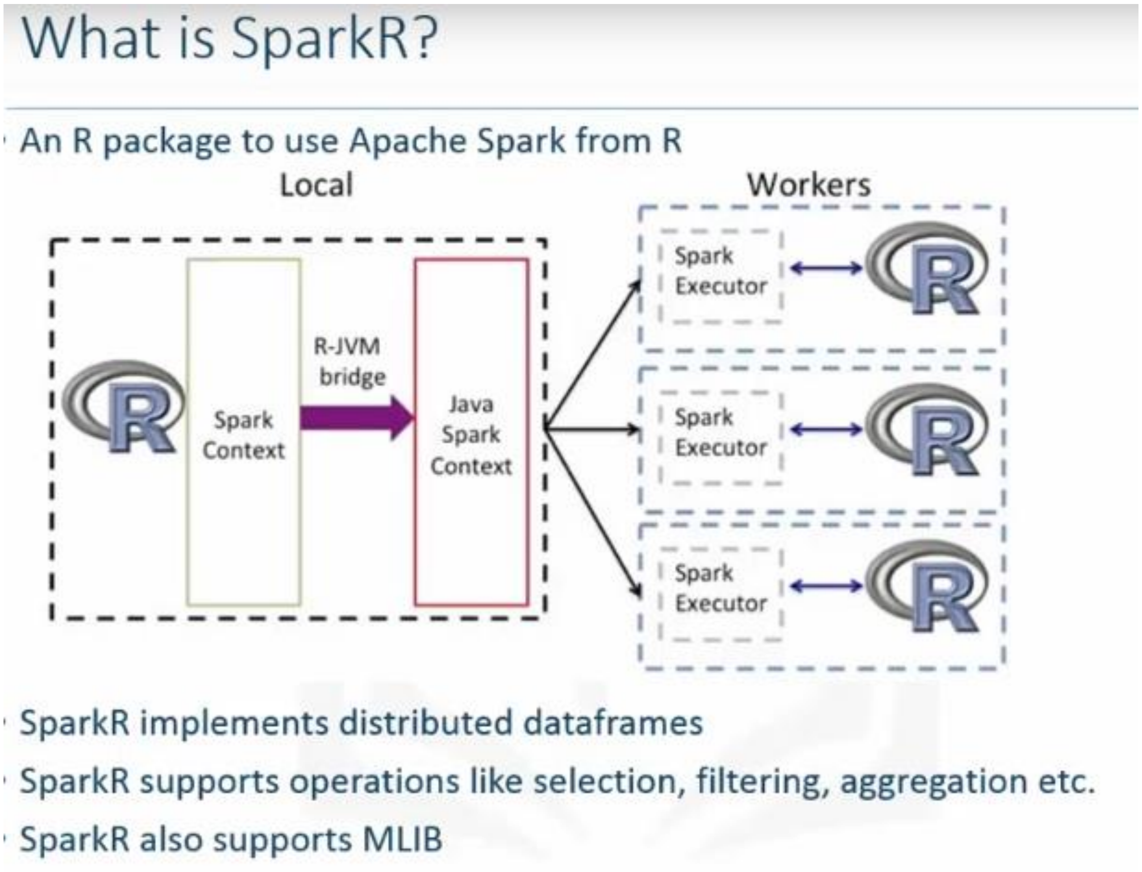
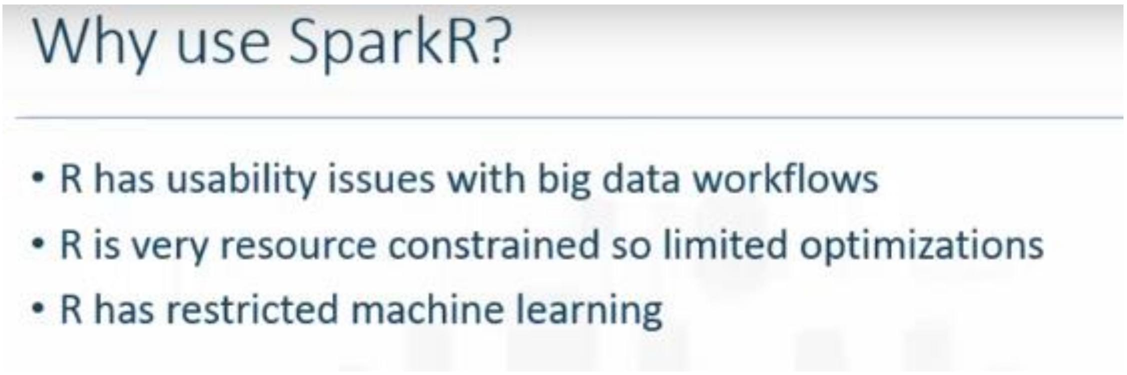
MODUE 1 INTRODUCTION TO SPARKR



What is SparkR?

- An R package to use Apache Spark from R

- SparkR implements distributed dataframes
- SparkR supports operations like selection, filtering, aggregation etc.
- SparkR also supports MLIB



Why use SparkR?

- R has usability issues with big data workflows
- R is very resource constrained so limited optimizations
- R has restricted machine learning

Courseware , current location
Course Info
Discussion
Wiki
Resources
Progress

About this course
Module 1: Introduction to SparkR

Learning Objectives

SparkR lesson 1 part 1 (6:29) current section

SparkR lesson 1 part 2 (7:41)

SparkR lesson 1 part 3 (4:42)

Introduction to Data Scientist Workbench (DSWB)

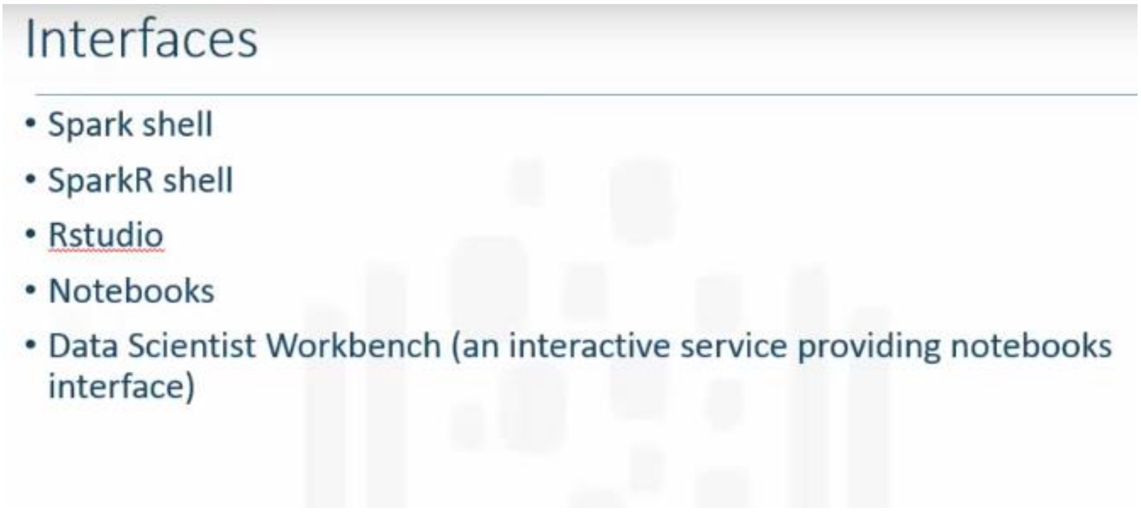
Graded Review Questions

Review Questions This content is graded
Module 2: Data manipulation in SparkR
Module3: Machine learning in SparkR
Final Exam
Completion Certificate
Course Survey and Feedback

SparkR lesson 1 part 1 (6:29)
Skip to a navigable version of this video's transcript.
6:18 / 6:28
Maximum Volume.
Skip to end of transcript.

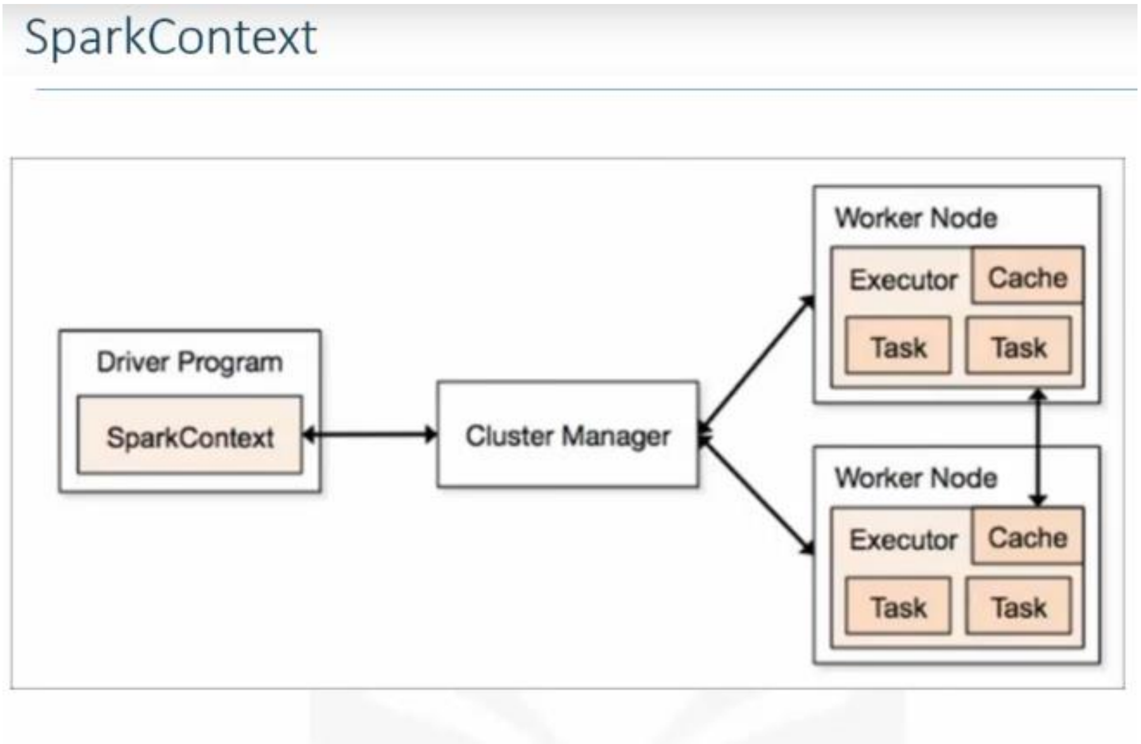
Welcome to the course, Introduction to analyzing Big Data in R using Apache Spark. This course
consists of three lessons, this being lesson 1, which is introduction to SparkR. The objectives
of this lesson are to learn what SparkR is, to learn why we would use SparkR, to list
the features of SparkR, and to understand the various interfaces into the SparkR environment.
So let's start by understanding exactly what SparkR is. Apache Spark is a cluster computing
framework for large scale data processing. Spark does not use the map reducers and execution
engine, however it is closely integrated with the Hadoop ecosystem and can run YARN using
Hadoop file formats and HDFS storage. Spark is best known for its ability to cache large
data sets in memory between jobs. It's default API is simpler than MapReduce. The favorite
interface is via Scala, but there is also support using Python. SparkR is a R package
that provides a light-weight front-end to users of Apache Spark from R. R is a popular
statistical programming language with a number of extensions that support data processing
and machine learning tasks. However, interactive data analysis in R is usually limited, as
the run time is single threaded and can only process data sets that fit in a single machines
memory. SparkR, and R package initially developed in the AMP lab, provides a front-end to Apache
Spark. And using Sparks distributed computation engine allows us to run large-scale data analysis
from the R shell. In Spark 1.61, SparkR provides a distributed dataframe implementation. And
this is the central component of SparkR. Dataframes are a fundamental data structure used for data
processing in R, and the concept of data frames has been extended to other languages with
libraries like Pandas. The distributed dataframe implementation supports R operations like
selection, filtering, aggregation, and so on, but on large data sets. Projects like
dplyr have further simplified expressing complex data manipulation tasks on data frames. SparkR

data frames present an API similar to dplyr and local R data frames that can scale to
large datasets using support for distributed computation in Spark. A data frame is a
distributed
collection of data, organized into named columns. It is conceptually equivalent to a table in
a relational database, or a data frame in R. Both with optimizations under the hood.
Data frames can be constructed from a wide array of sources such as structured data files,
tables in Hive, external databases, or existing local R data frames. SparkR also supports
distributed machine learning using MLlib. As we can see in the diagram on the left hand
side, we have the R environment, with the Spark Context, the JVM, and the Java Spark
Context. And then on the right hand side, we have the task distributed across the cluster
so that we are not limited to single threading, or resourced on a single unit, a single
machine.
We can use as many machines that are in the cluster. So, that raises a question, why would
you use SparkR? R, as we notice in the previous slide is restricted to single thread and the
amount of memory on a single machine. So using R on Big Data is very difficult. SparkR is
multi--threaded across many machines, as many machines as there are in the cluster, as well
as having access to the memory and all machines in that cluster. A typical example would be
where there's a large amount of initial data which needs cleaning, filtering, and aggregating
before the statistical analysis can be performed. This means reducing the volume of raw
data
to a relevant subset, which can fit onto a single machine, where R can apply the statistical
learning model. This is very difficult, if not impossible in R. Once we have the set
of data we're going to work on, we need to train the model. Model training involves applying
a number of parameters to learn which parameter value works best. Most users want to do
this
parameter tuning in parallel, and aggregate the parameters to find the best model for
the dataset. Performance in R is constrained by having to do this work on a single machine.
SparkR can do this across multiple nodes in a cluster, improving performance by several
orders of magnitude. When the raw data is very large, and the user wants to apply a
number of features to the whole dataset, and so a distributed machine learning algorithm
is needed. R cannot do this, but SparkR comes with a machine learning library and Spark
operates in a distributed environment, so large scaled machine learning is now possible
with R.

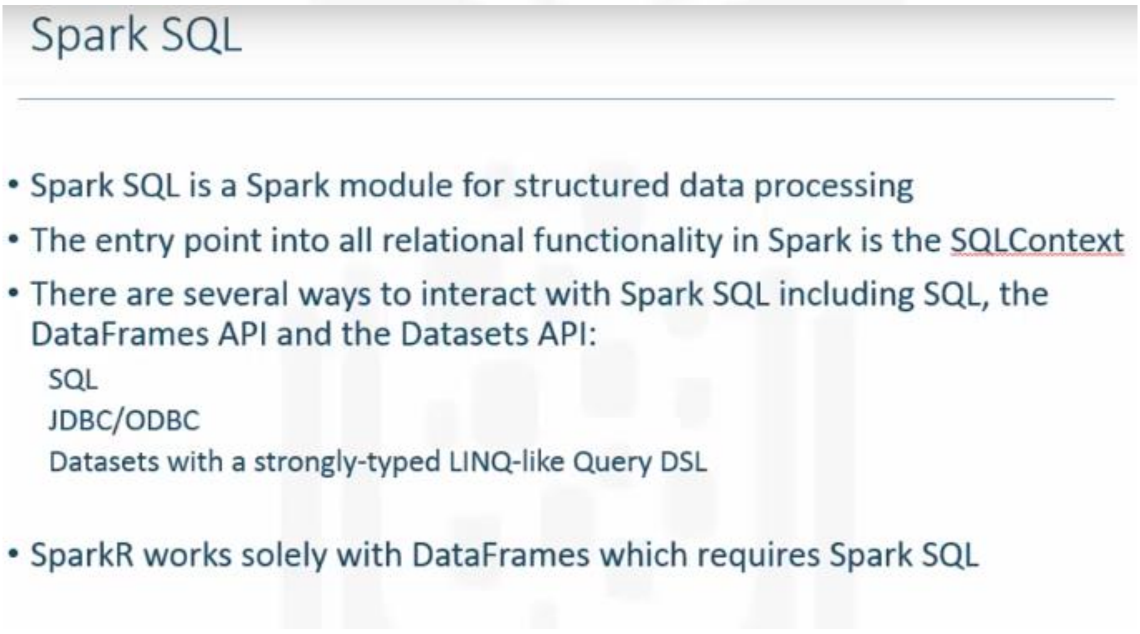


Interfaces

- Spark shell
- SparkR shell
- Rstudio
- Notebooks
- Data Scientist Workbench (an interactive service providing notebooks interface)



SparkContext



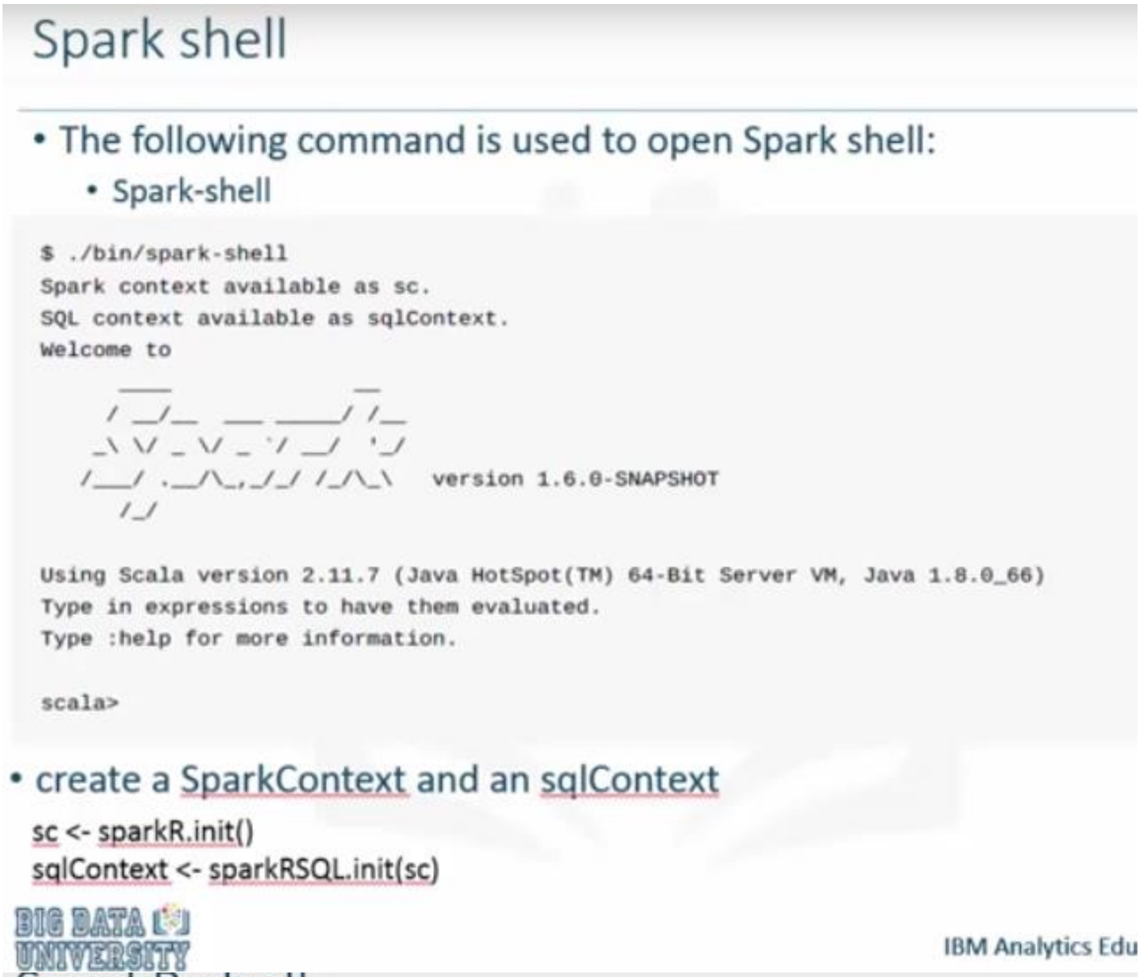
Spark SQL

- Spark SQL is a Spark module for structured data processing
- The entry point into all relational functionality in Spark is the SQLContext
- There are several ways to interact with Spark SQL including SQL, the DataFrames API and the Datasets API:
  SQL
  JDBC/ODBC
  Datasets with a strongly-typed LINQ-like Query DSL
- SparkR works solely with DataFrames which requires Spark SQL

So, what features does SparkR have? We've already talked about the scalability with many machines and many cores, operations being executed on SparkR dataframes get automatically distributed across all the cores in the cluster. As a result of all of this, SparkR dataframes can be used on terabytes of data, and run on clusters of thousands of machines. Another benefit is the dataframes optimizations. SparkR dataframes inherit all the optimizations made to the computation engine in terms of code, generation, and memory management. Then there are all the data sources APIs, by tying into Sparks' SQL data sources API, SparkR can read in from a variety of sources including Hive tables, JSON files, packet files, etc. Then we have the RDDs as distributed lists. SparkR exposes the RDD API's of Spark as a distributed list in R. For example we can read an input file in HDFS and process every line using lapply on an RDD. In addition to lapply, SparkR also allows closures to be appplied

on every partition using lapply with partition. But the supported RDD functions include operations
like reduce, reduce by key, group by key, and collect. Then there's the serializing
the closures. SparkR automatically serializes the necessary variables to execute the function
on the cluster. For example, if we use some global variables in a function passed to
lapply, SparkR will automatically capture these variables and copy them to the cluster. In
addition, we can use existing R packages. SparkR also allows easy use of existing R
packages inside closures. The include package command can be used to indicate packages that
should be loaded before every closures executed on the cluster. Now we are going to look at
how we interface with SparkR. There are several ways to interface with SparkR. You can interface
through the Spark shell, you can interface through the SparkR shell, the difference between
the two being that the SparkR shell has the SQLContext and SparkContext already created
for you. You can use Rstudio, which is an integrated development environment. You can
use notebooks and a notebook gives you an interactive web-based editor, which you can
do all sorts of interesting things with. And there's the Data Scientists Workbench, which
provides a unified platform of diverse tools, capable of using open source tools including
Jupyter and Zepplin, as well as Rstudio, and we're going to look further into these. The
entry point into SparkR from the R environment is SparkContext. There is a conceptual R program
to a Spark cluster. First step in any Spark application is to create a Spark context.
Spark context allows your Spark application to access the cluster through open-source
managers such as YARN or Sparks own cluster manager. It represents the client connection
to a Spark execution environment, and has to be created before using any of Sparks features
or services in your applications, such as RDDs, accumulators, broadcast variables, etc.
To work with dataframes, we also need an SQLContext. And this is created from the SparkContext,
and provides the interface into Spark SQL. If you're working from the SparkR shell rather
than the Spark shell, the SQL context and SparkContext will already have been created
for you. Where as in the Spark shell, you'll have to create those yourself. With a SQLContext,
applications can create dataframes from a local R dataframe, from a Hive table, or from
other data sources. Spark SQL is a Spark module for structured data processing. It is a distributed
SQL engine designed to leverage the power of Sparks computation model, which is based
on RDDs. Unlike the basic Spark RDD API, the interface is provided by Spark SQL, provide
Spark with more information about the structure of both the data and the computation being performed.
Internally, Spark uses this extra information to perform extra optimizations. SparkR works
with dataframes, which means we will need an SQLContext, which can be created from the
SparkContext. The entry point into all the relational functionality in Spark is through
the SQLContext. There are several ways to interact with Spark SQL including SQL itself,
the DataFrames API and the Datasets API. Data analysts will likely use SQL as their query
language. The SQL function on the SQLContext enables applications to run SQL queries programmatically,
and return the results as a dataframe. Spark SQL also includes a data source API that can
read data from other databases using JDBC. This functionality should be preferred over
using JDBC RDD API. This is because the results are returned as a dataframe and can be easily
processed in Spark SQL or joined with other data sources. The JDBC data source is also
easier to use from Java or Python, as it does not require the user to provide the class
tag. Datasets are similar to RDDs, however instead of using Java serialization or Kryo,

they use a specialized encoder, to serialize the objects for processing or transmitting of the network. When computing a result, the same execution engine is used, independent of which API or language you are using to express the computation. This unification means that developers can easily switch back and forth between the various, based upon which one provides the most natural way to express a given transformation.



Spark shell

- The following command is used to open Spark shell:
  - Spark-shell

```
$ ./bin/spark-shell
Spark context available as sc.
SQL context available as sqlContext.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0-SNAPSHOT
      /_/

Using Scala version 2.11.7 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_66)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

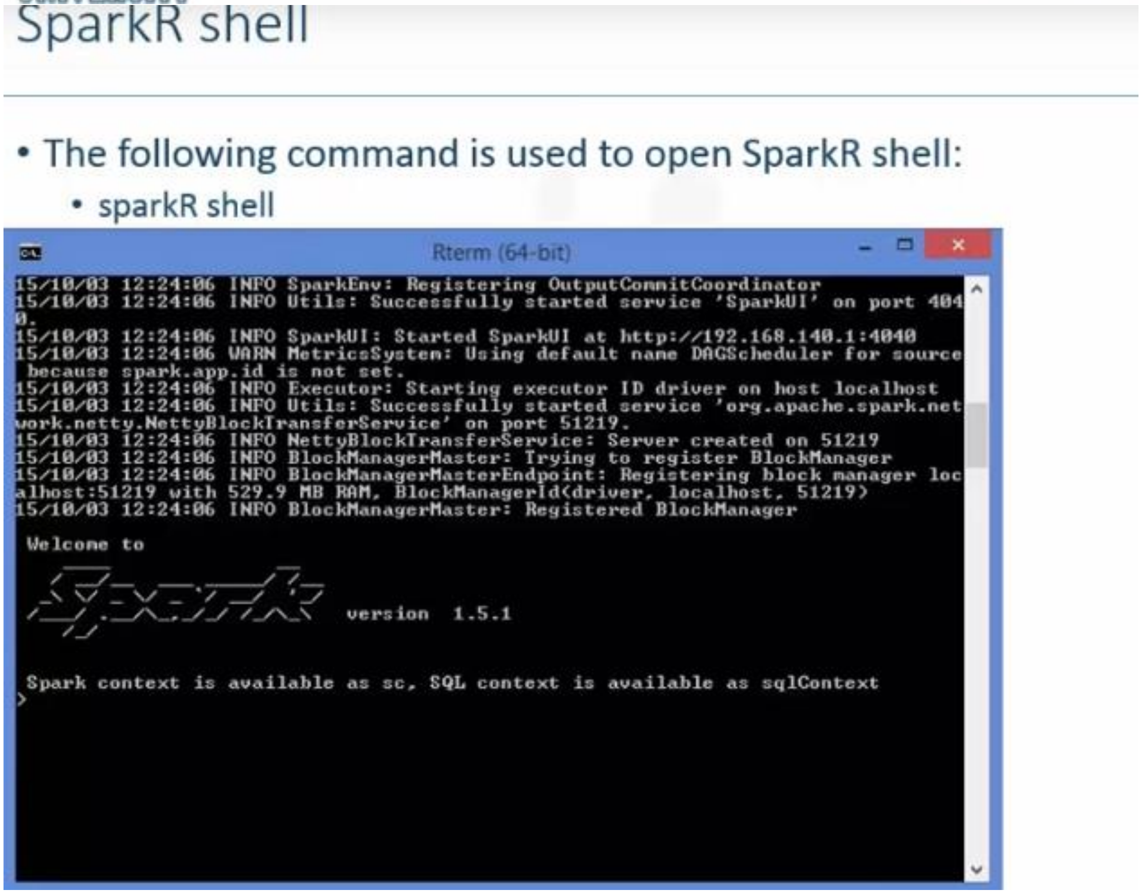
- create a SparkContext and an sqlContext

  sc <- sparkR.init()

  sqlContext <- sparkRSQL.init(sc)

BIG DATA UNIVERSITY

IBM Analytics Edu

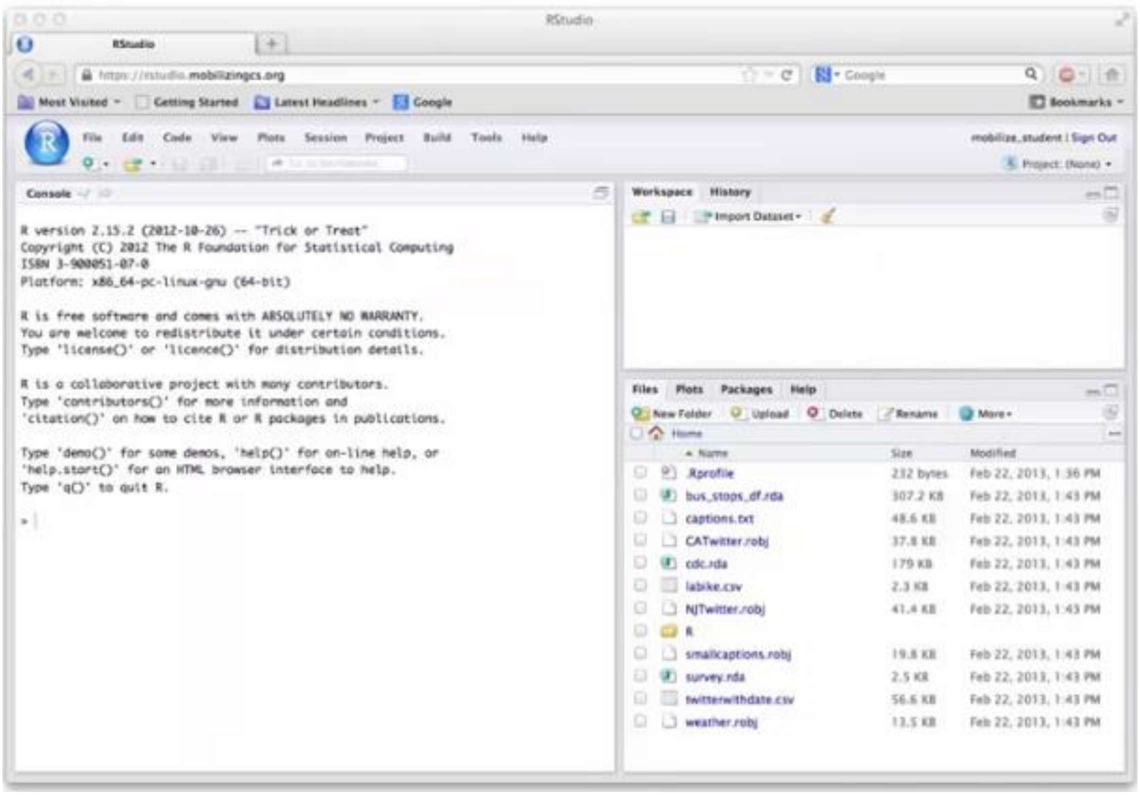


SparkR shell

- The following command is used to open SparkR shell:
  - sparkR shell

```
15/10/03 12:24:06 INFO SparkEnv: Registering OutputCommitCoordinator
15/10/03 12:24:06 INFO Utils: Successfully started service 'SparkUI' on port 404
0.
15/10/03 12:24:06 INFO SparkUI: Started SparkUI at http://192.168.140.1:4040
15/10/03 12:24:06 WARN MetricsSystem: Using default name DAGScheduler for source
 because spark.app.id is not set.
15/10/03 12:24:06 INFO Executor: Starting executor ID driver on host localhost
15/10/03 12:24:06 INFO Utils: Successfully started service 'org.apache.spark.net
work.netty.NettyBlockTransferService' on port 51219.
15/10/03 12:24:06 INFO NettyBlockTransferService: Server created on 51219
15/10/03 12:24:06 INFO BlockManagerMaster: Trying to register BlockManager
15/10/03 12:24:06 INFO BlockManagerMasterEndpoint: Registering block manager loc
alhost:51219 with 529.9 MB RAM, BlockManagerId(driver, localhost, 51219)
15/10/03 12:24:06 INFO BlockManagerMaster: Registered BlockManager

 Welcome to
    ____              __
   / __/__  ___ _____/ /__
  _\ \/ _ \/ _ `/ __/  '_/
 /___/ .__/\_,_/_/ /_/\_\   version  1.5.1
    /_/


 Spark context is available as sc, SQL context is available as sqlContext
>
```

RStudio

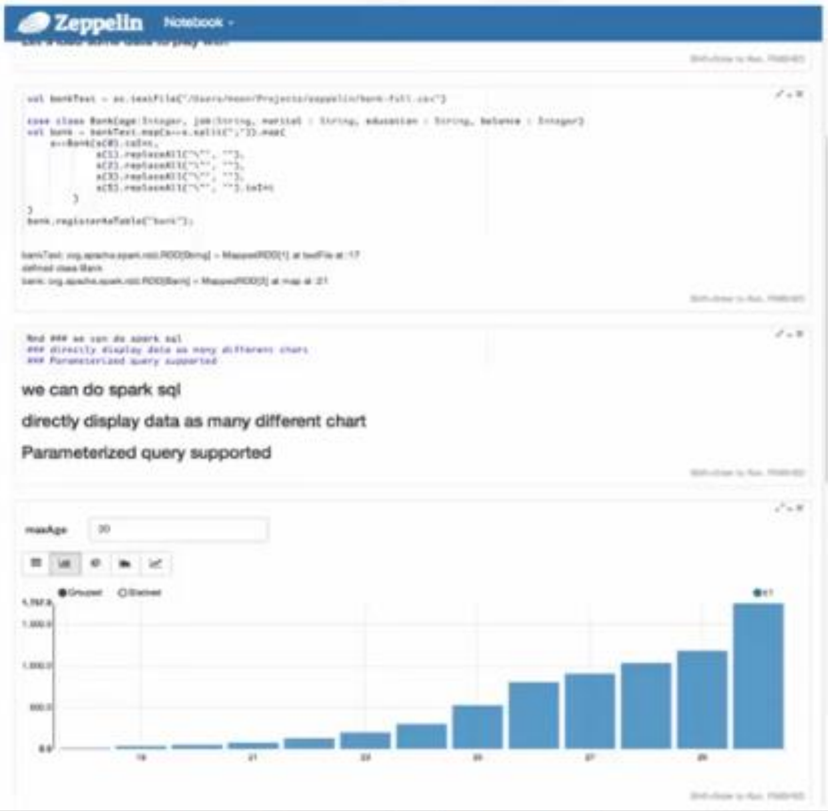

RStudio
https://rstudio.mobilizingcs.org
File Edit Code View Plots Session Project Build Tools Help
mobilize_student | Sign Out
Project: (None)
Console
Workspace History
Import Dataset
Files Plots Packages Help
New Folder Upload Delete Rename More
Home

```
R version 2.15.2 (2012-10-26) -- "Trick or Treat"
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-pc-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.

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.

>
```

Name	Size	Modified
.Rprofile	232 bytes	Feb 22, 2013, 1:36 PM
bus_stops_df.rda	307.2 KB	Feb 22, 2013, 1:43 PM
captions.txt	48.6 KB	Feb 22, 2013, 1:43 PM
CATwitter.robj	37.8 KB	Feb 22, 2013, 1:43 PM
cdc.rda	179 KB	Feb 22, 2013, 1:43 PM
labike.csv	2.3 KB	Feb 22, 2013, 1:43 PM
NJTwitter.robj	41.4 KB	Feb 22, 2013, 1:43 PM
R		
smallcaptions.robj	19.8 KB	Feb 22, 2013, 1:43 PM
survey.rda	2.5 KB	Feb 22, 2013, 1:43 PM
twitterwithdate.csv	56.6 KB	Feb 22, 2013, 1:43 PM
weather.robj	13.5 KB	Feb 22, 2013, 1:43 PM

Notebooks

- Zeppelin notebook



```
val bankText = sc.textFile("/Users/moon/Projects/zeppelin/bank-full.csv")

case class Bank(age:Integer, job:String, marital : String, education : String, balance : Integer)
val bank = bankText.map(s=>s.split(";")).map(
    s=>Bank(s(0).toInt,
            s(1).replaceAll("\"", ""),
            s(2).replaceAll("\"", ""),
            s(3).replaceAll("\"", ""),
            s(5).replaceAll("\"", "").toInt
        )
)
bank.registerAsTable("bank");
```

```
bankText: org.apache.spark.rdd.RDD[String] = MappedRDD[1] at textFile at :17
defined class Bank
bank: org.apache.spark.rdd.RDD[Bank] = MappedRDD[3] at map at :21
```

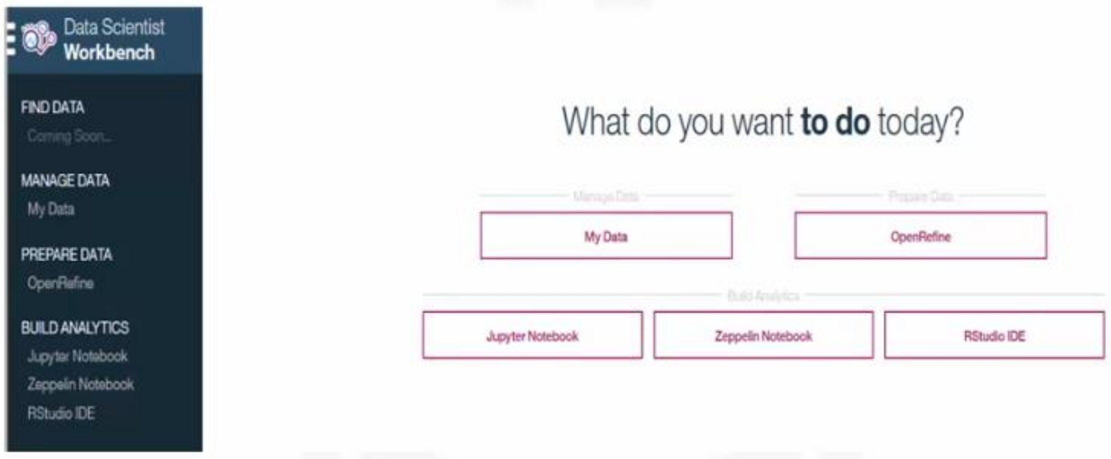
```
%md ### we can do spark sql
### directly display data as many different chart
### Parameterized query supported
```

we can do spark sql

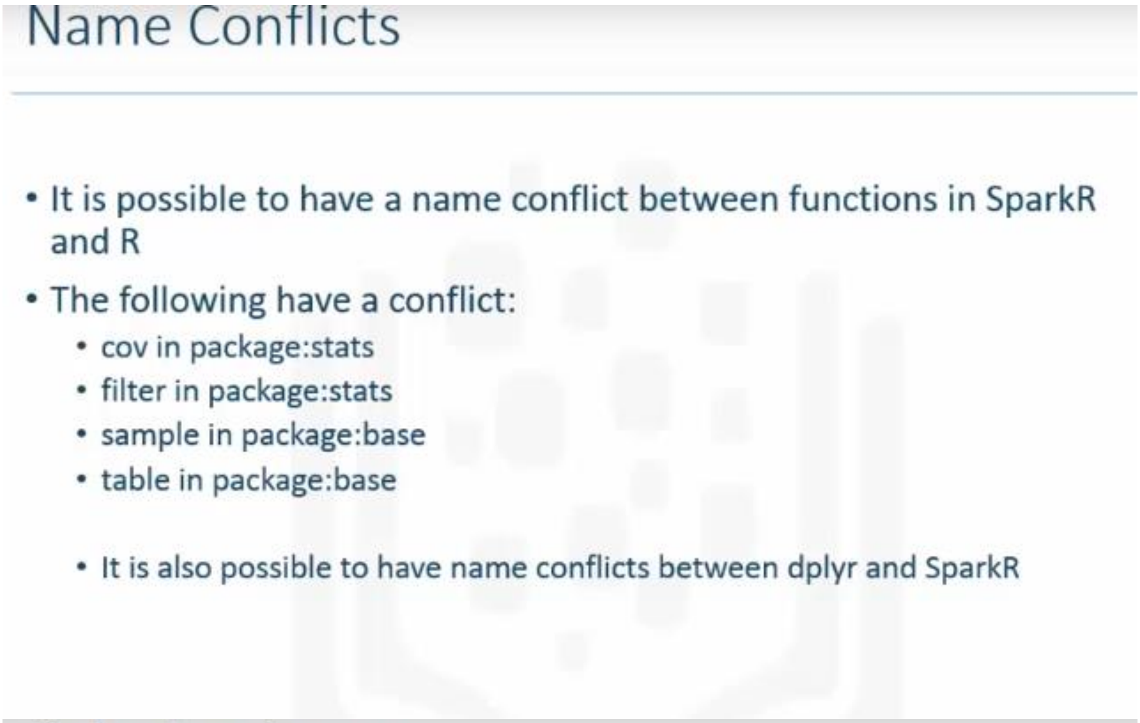
directly display data as many different chart

Parameterized query supported

Data Scientist Workbench (DSWB)



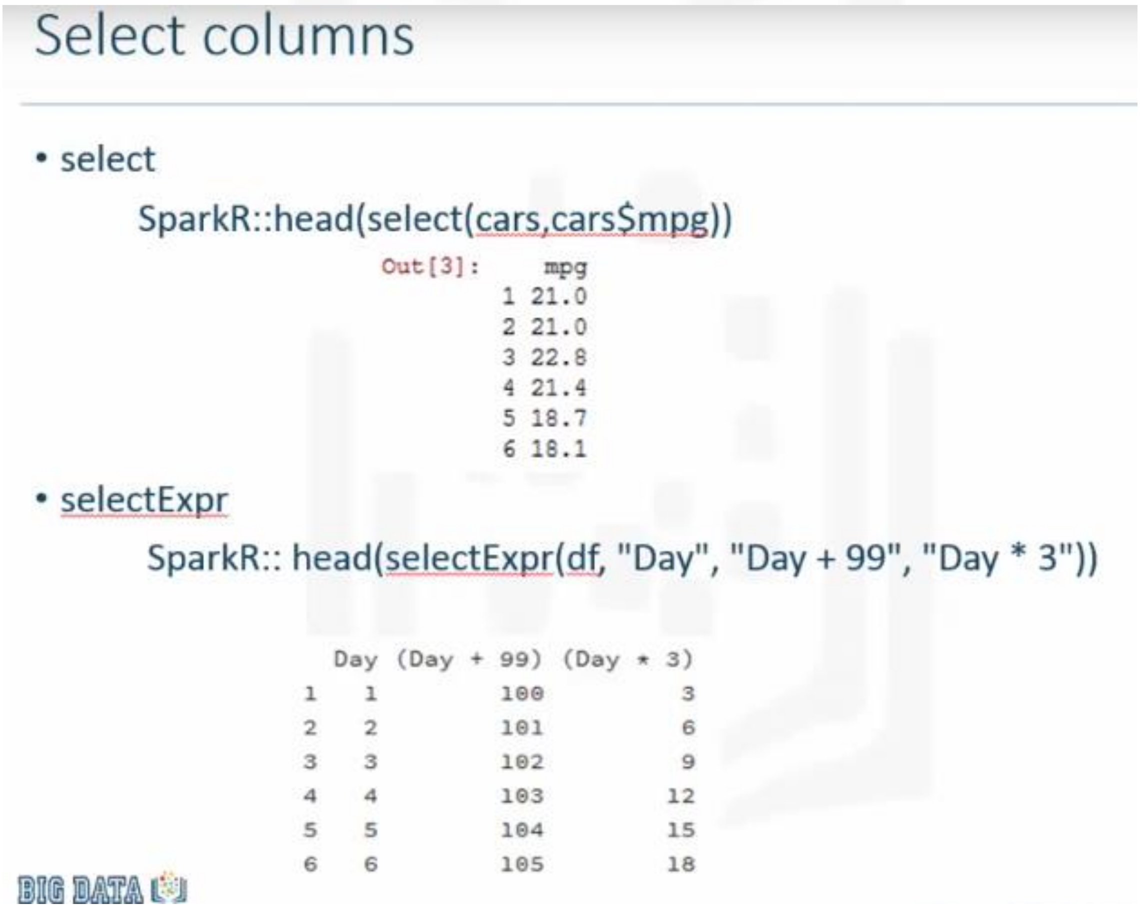
MODULE 2 DATA MIANIPULATION IN SPARKR



Name Conflicts

- It is possible to have a name conflict between functions in SparkR and R
- The following have a conflict:
  - cov in package:stats
  - filter in package:stats
  - sample in package:base
  - table in package:base

  - It is also possible to have name conflicts between dplyr and SparkR



Select columns

- select

```
SparkR::head(select(cars,cars$mpg))
```

```
Out[3]:    mpg
         1 21.0
         2 21.0
         3 22.8
         4 21.4
         5 18.7
         6 18.1
```

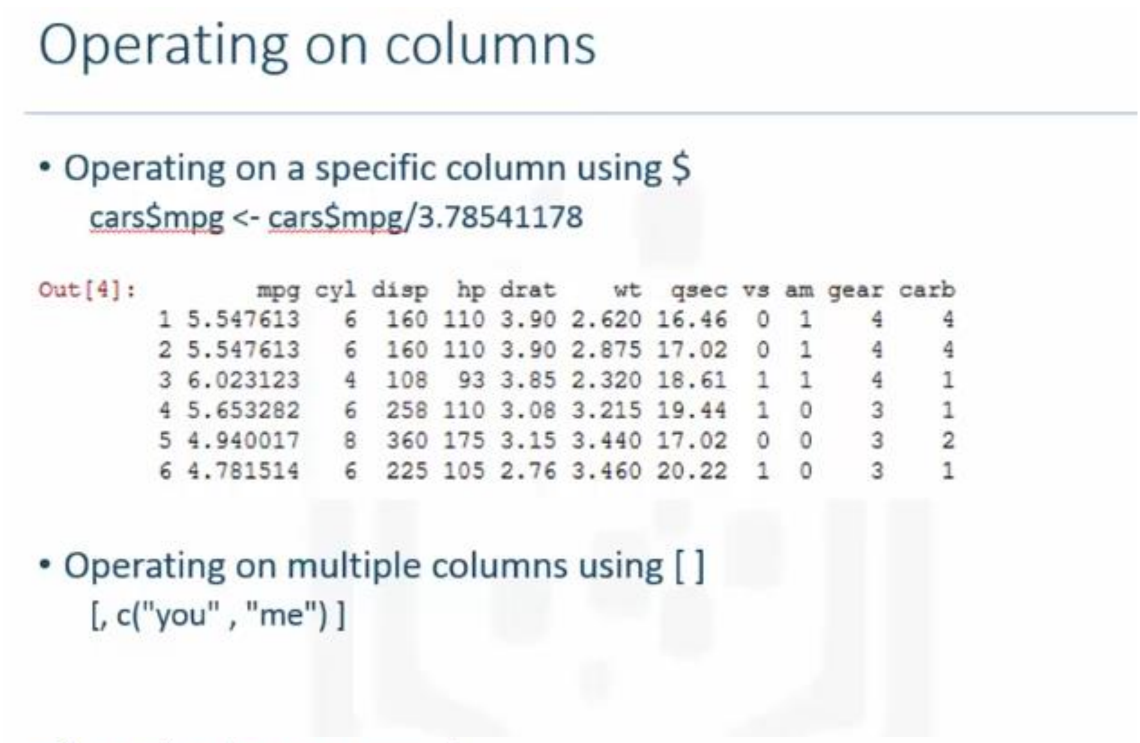
- selectExpr

```
SparkR:: head(selectExpr(df, "Day", "Day + 99", "Day * 3"))
```

```
  Day (Day + 99) (Day * 3)
1   1        100         3
2   2        101         6
3   3        102         9
4   4        103        12
5   5        104        15
6   6        105        18
```

Welcome back to the course, Introduction to analyzing Big Data in R using Apache Spark.
This second lesson, we will use data manipulation in SparkR to demonstrate R programming in
Spark. The objectives of this lesson are to understand how to use dataframes, learn how to select data, to filter data, and to aggregate data. To learn how to operate on columns and then to understand how to write SQL queries. Here we are going to be looking at dataframes. SparkR began as an R package that imported Sparks RDD functionalities into

R. This was followed by adding Spark SQL and schema RDDs. However, schema RDDs soon became
deprecated. An alternative needed to be found. This is where dataframes came in. Dataframes
use the distributed parallel capabilities offered by RDDs. But imported a schema on
top of the data giving it metadata, column names, data types, etc. This gives structure
to the data, making life easier for users. In addition, it makes our integration much
easier, since R already uses dataframes as its standard. The SparkR back-end passes data
structures and method calls into the Spark JVM. SparkR dataframe features column access,
just like in R. Also, there is dplyr dataframe manipulation such as filter, group by, summarize,
mutate, and access to external R packages that extend the R syntax. There are several
ways to create a dataframe in SparkR. The simplest way to create a dataframe is to convert
a local R dataframe into a Spark R dataframe. Specifically, we can use the create dataframe
command and pass in the local R dataframe to create a SparkR dataframe. Because R dataframes
already have a schema, SparkR uses this for its own dataframe. A dataframe can be created
from raw data by creating a list of lists. Creating a schema structure as required, and
finally using the create dataframe command to create a dataframe in the Spark cluster
by combining the two. Importantly, SparkR supports operating on a variety of data sources
through the dataframe interface. The general method of creating dataframes from data sources
is read.df. This method takes in the SQLContext, the path for the file to load, and type of
data source. SparkR supports reading JSON natively, and through Spark packages, you
can find data source connectors for popular file formats like csv. Something to be aware
of when using SparkR is the possibility of having name name conflicts between functions
in SparkR and R. The following functions are masked by SparkR packages. cov in the package:
stats, filter in the package: stats, sample in the package: base, table in the package:
base. It's also possible to have name conflicts between dplyr and SparkR, since part of the
SparkR is modeled on the dplyr package, certain functions share the same name in SparkR and
dplyr. Depending on the load order of the two packages, some function from the package
load first are masked by those in the package loaded after. In such cases, prefix search
calls with the package name, for instance SparkR::function name, or dplyr::function
name. If you want to find out what the search path is, you can find it in R with the search
command. Here we see how to select columns in SparkR. We have two flavors of select.
One, select, that gets a list of column names, and another one, selectExpr, that we pass
a string containing a SQL expression. The select function selects the specified columns
and returns it as a new dataframe. This is similar to the select statement in SQL. Here
in the examples, you can see what was referenced in the name conflicts previously, where we
have the package name proceeding the function that could be masked in this case head, so
we have SparkR::head. Then we have the select command, followed by the dataframe name, and
the column dataframe $, column name (mpg in this case). selectExpr is a variant of select
that selects columns in dataframe while projecting SQL expressions, and we can see that here,
again with the package prefix and the selectExpr dataframe name, and then the columns and the
expression that we want to use for the select. Since select returns a SparkR dataframe, we
have to use functions like head or take or collect to see the new dataframe.



Operating on columns

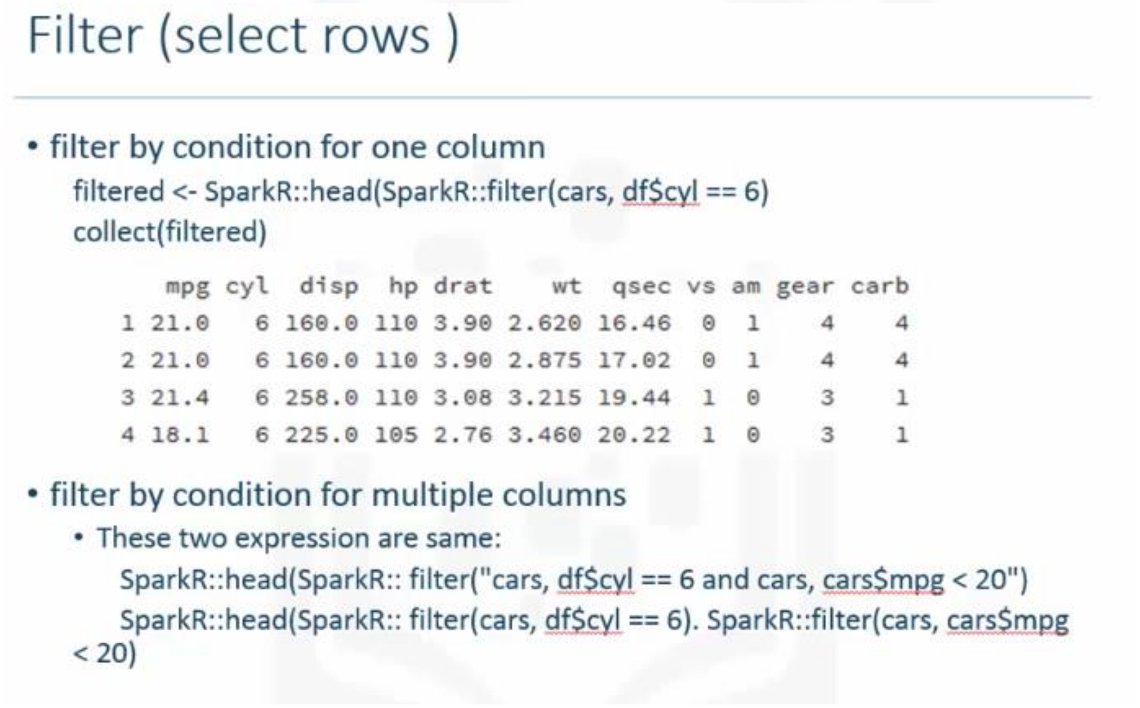
- Operating on a specific column using $

  cars$mpg <- cars$mpg/3.78541178

```
Out[4]:        mpg cyl disp  hp drat    wt  qsec vs am gear carb
        1 5.547613   6  160 110 3.90 2.620 16.46  0  1    4    4
        2 5.547613   6  160 110 3.90 2.875 17.02  0  1    4    4
        3 6.023123   4  108  93 3.85 2.320 18.61  1  1    4    1
        4 5.653282   6  258 110 3.08 3.215 19.44  1  0    3    1
        5 4.940017   8  360 175 3.15 3.440 17.02  0  0    3    2
        6 4.781514   6  225 105 2.76 3.460 20.22  1  0    3    1
```

- Operating on multiple columns using [ ]

  [, c("you" , "me")]



Filter (select rows)

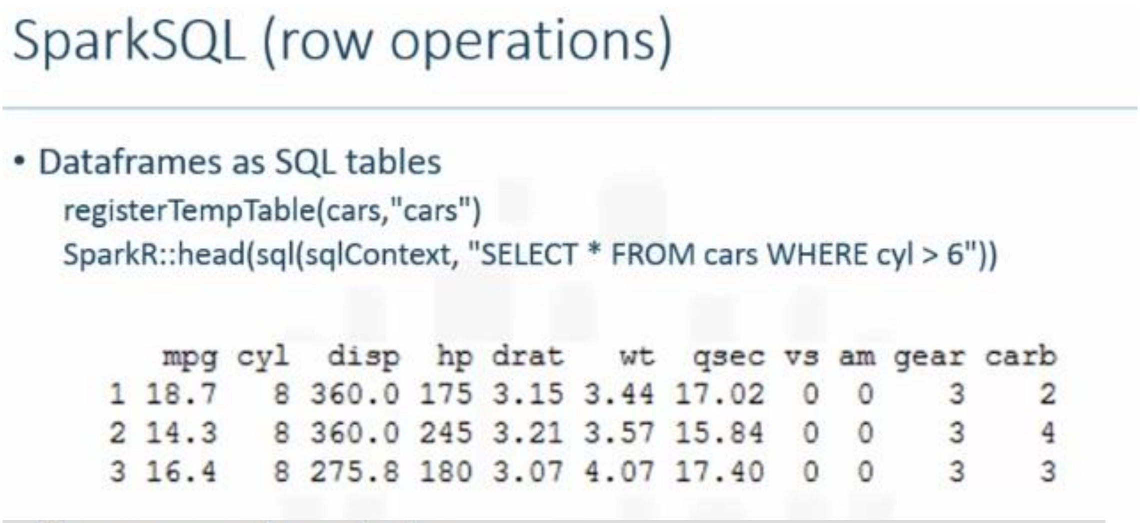
- filter by condition for one column

  filtered <- SparkR::head(SparkR::filter(cars, df$cyl == 6)
  collect(filtered)

```
   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
1 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
2 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
3 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
4 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
```

- filter by condition for multiple columns
  - These two expression are same:

    SparkR::head(SparkR:: filter("cars, df$cyl == 6 and cars, cars$mpg < 20")
    SparkR::head(SparkR:: filter(cars, df$cyl == 6). SparkR::filter(cars, cars$mpg
  < 20)

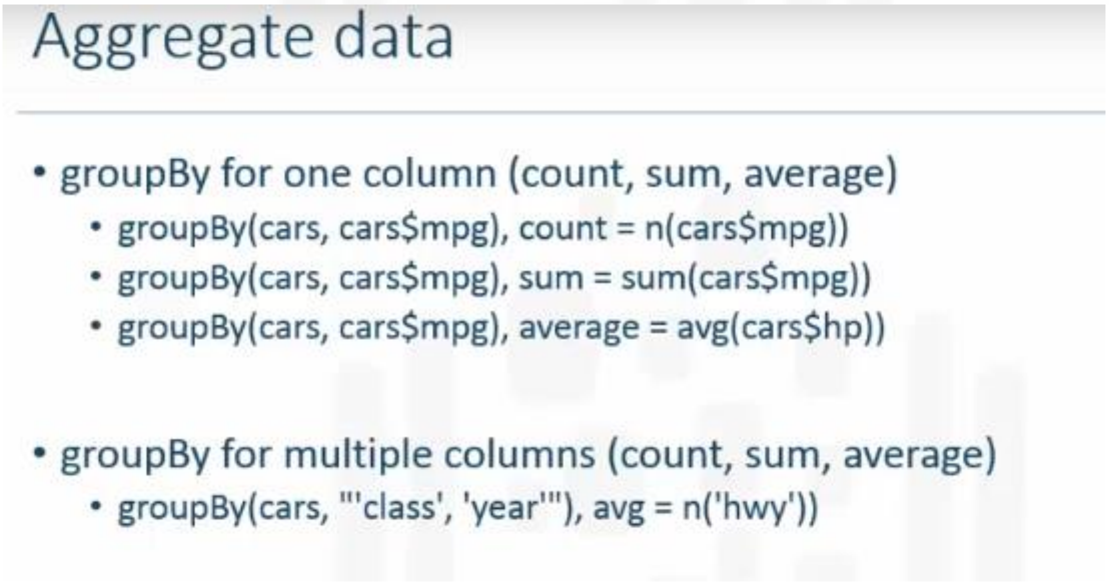


SparkSQL (row operations)

- Dataframes as SQL tables

```
registerTempTable(cars,"cars")
SparkR::head(sql(sqlContext, "SELECT * FROM cars WHERE cyl > 6"))
```

```
   mpg cyl  disp  hp drat   wt  qsec vs am gear carb
1 18.7   8 360.0 175 3.15 3.44 17.02  0  0    3    2
2 14.3   8 360.0 245 3.21 3.57 15.84  0  0    3    4
3 16.4   8 275.8 180 3.07 4.07 17.40  0  0    3    3
```



Aggregate data

- groupBy for one column (count, sum, average)
  - groupBy(cars, cars$mpg), count = n(cars$mpg))
  - groupBy(cars, cars$mpg), sum = sum(cars$mpg))
  - groupBy(cars, cars$mpg), average = avg(cars$hp))

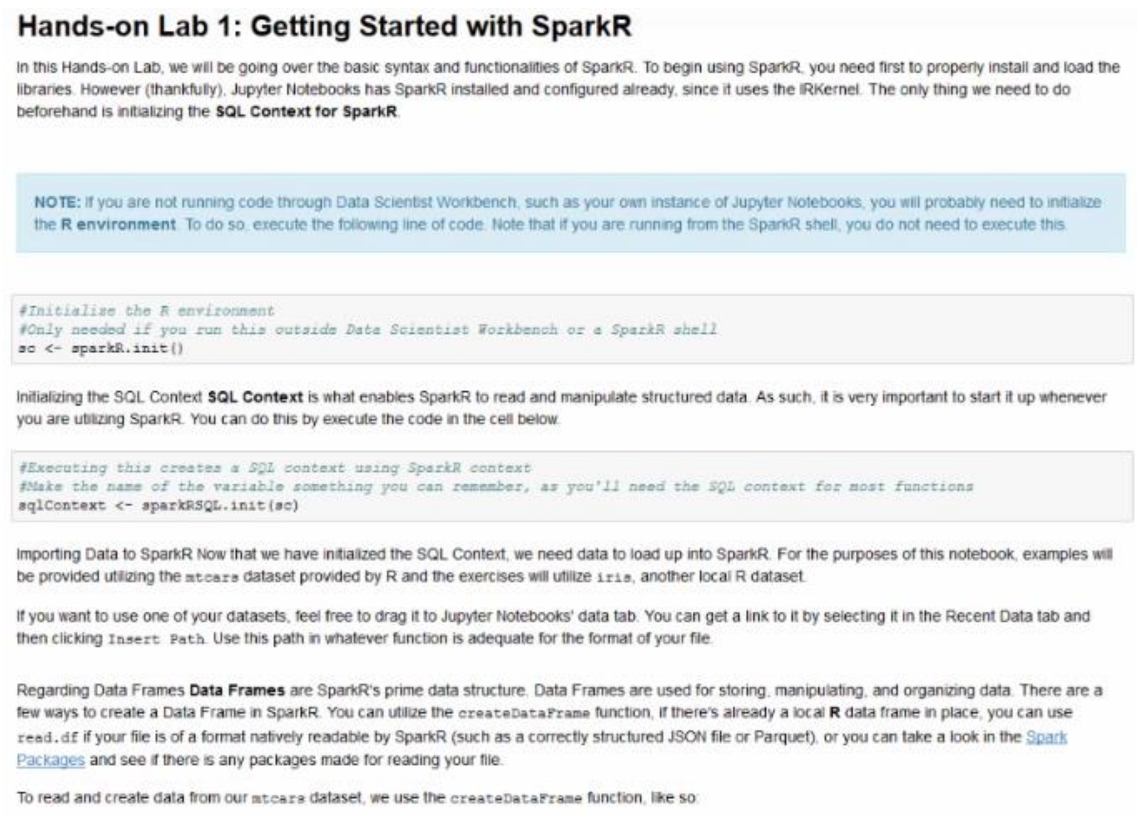
- groupBy for multiple columns (count, sum, average)
  - groupBy(cars, "'class', 'year'"), avg = n('hwy'))

Here we are going to look at how we operate on columns. SparkR also provides a number of functions that can directly be applied to columns in data processing and during aggregation.

Anyone used to R will immediately recognize these operations in SparkR. So here we have, in an example operating on a specific column using the $. We have the dataframe $ column name, or cars$mpg and divide by, so the operation is dividing the cars mpg column by 3.785 and so on. Here, rather than operating on columns, we are operating on rows using the filter command. With the filter command, we filter the rows of a dataframe according to a given condition, which we pass as an argument. We can define conditions as SQL conditions using column names, or by using the $ notation. Filter returns the rows of a dataframe that matches the given conditions. For example, here, filter by condition for a column. So we have the SparkR prefix for head, then we have SparkR prefix for filter within the head, so we are going to review the dataframe. And then we have the dataframe, cars and the column with the $ notation cyl equal to 6. We can also perform the previous selections and filtering by using standard SQL queries against a SparkSQL dataframe. In order to do that, we need to register the table. As we can see here, registerTempTable cars as "cars". Then we can use SparkR SQL function, using SQLContext as in the example here. SELECT* FROM cars WHERE cyl > 6. This

standard SQL should be familiar to a majority of people. When we're trying to aggregate
data, a basic operation we can do that data aggregation with on the dataframes is the
groupBy. The groupBy function groups your SparkR dataframes on the specified column
or columns so that you can run aggregations on them just like an SQL groupBy statement.
groupBy is a transformation operation. This operation will return a new dataframe which
is basically made up of a key which is the group, or grouping, and a list of items in
that group. We can add as many summary aggregation columns as there are functions we
want to
use for operations. The groupBy function outputs group data object, which can then be
passed
into aggregation functions. SparkR defines the following aggregation functions that we
can apply to dataframe object calls. The avg, the min, the max, the sum, count, count
distinct,
and some distinct. We use them by passing columns with the $ notation and then we get
returned columns, so they need to be part of a select call for a dataframe. As you can
see here in the examples, we have the groupBy for a single column, with a count or the sum
or the avg. And then we have the groupBy for multiple columns, where we have the car and
the class and the year, and then the avg for the highway. So, within this lesson, we've
come to understand how to use dataframes, we've learned how to select data, how to filter
data, and how to aggregate data. We've learned how to operate on columns and rows and
we've
come to understand how to write SQL queries.

>> Lab1:



Hands-on Lab 1: Getting Started with SparkR

In this Hands-on Lab, we will be going over the basic syntax and functionalities of SparkR. To begin using SparkR, you need first to properly install and load the libraries. However (thankfully), Jupyter Notebooks has SparkR installed and configured already, since it uses the IRKernel. The only thing we need to do beforehand is initializing the **SQL Context for SparkR**.

NOTE: If you are not running code through Data Scientist Workbench, such as your own instance of Jupyter Notebooks, you will probably need to initialize the **R environment**. To do so, execute the following line of code. Note that if you are running from the SparkR shell, you do not need to execute this.

```
#Initialize the R environment
#Only needed if you run this outside Data Scientist Workbench or a SparkR shell
sc <- sparkR.init()
```

Initializing the SQL Context **SQL Context** is what enables SparkR to read and manipulate structured data. As such, it is very important to start it up whenever you are utilizing SparkR. You can do this by execute the code in the cell below.

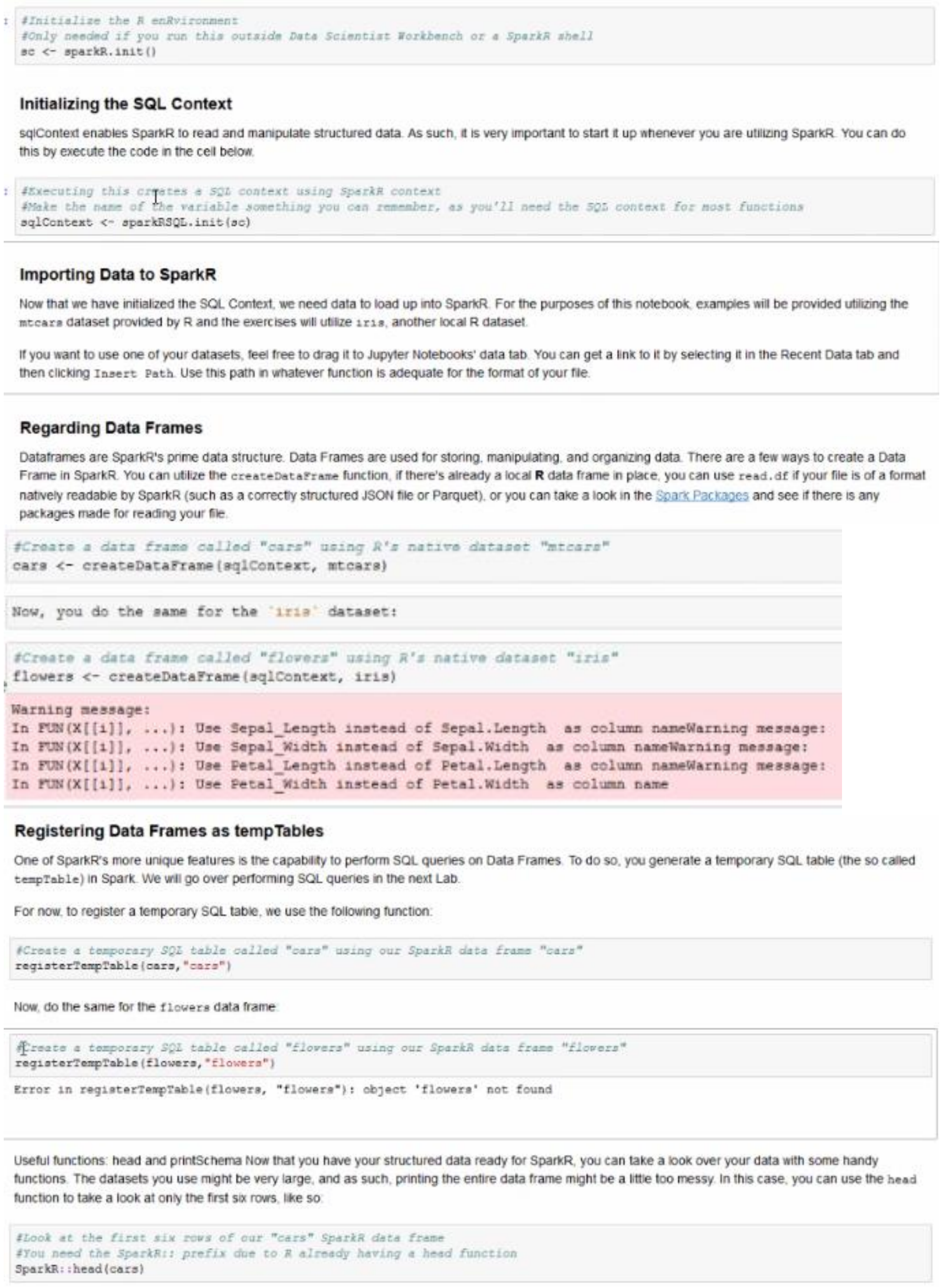
```
#Executing this creates a SQL context using SparkR context
#Make the name of the variable something you can remember, as you'll need the SQL context for most functions
sqlContext <- sparkRSQL.init(sc)
```

Importing Data to SparkR Now that we have initialized the SQL Context, we need data to load up into SparkR. For the purposes of this notebook, examples will be provided utilizing the `mtcars` dataset provided by R and the exercises will utilize `iris`, another local R dataset.

If you want to use one of your datasets, feel free to drag it to Jupyter Notebooks' data tab. You can get a link to it by selecting it in the Recent Data tab and then clicking `Insert Path`. Use this path in whatever function is adequate for the format of your file.

Regarding Data Frames **Data Frames** are SparkR's prime data structure. Data Frames are used for storing, manipulating, and organizing data. There are a few ways to create a Data Frame in SparkR. You can utilize the `createDataFrame` function, if there's already a local **R** data frame in place, you can use `read.df` if your file is of a format natively readable by SparkR (such as a correctly structured JSON file or Parquet), or you can take a look in the Spark Packages and see if there is any packages made for reading your file.

To read and create data from our `mtcars` dataset, we use the `createDataFrame` function, like so:



```
#Initialize the R enRvironment
#Only needed if you run this outside Data Scientist Workbench or a SparkR shell
sc <- sparkR.init()
```

Initializing the SQL Context

sqlContext enables SparkR to read and manipulate structured data. As such, it is very important to start it up whenever you are utilizing SparkR. You can do this by execute the code in the cell below.

```
#Executing this creates a SQL context using SparkR context
#Make the name of the variable something you can remember, as you'll need the SQL context for most functions
sqlContext <- sparkRSQL.init(sc)
```

Importing Data to SparkR

Now that we have initialized the SQL Context, we need data to load up into SparkR. For the purposes of this notebook, examples will be provided utilizing the `mtcars` dataset provided by R and the exercises will utilize `iris`, another local R dataset.

If you want to use one of your datasets, feel free to drag it to Jupyter Notebooks' data tab. You can get a link to it by selecting it in the Recent Data tab and then clicking `Insert Path`. Use this path in whatever function is adequate for the format of your file.

Regarding Data Frames

Dataframes are SparkR's prime data structure. Data Frames are used for storing, manipulating, and organizing data. There are a few ways to create a Data Frame in SparkR. You can utilize the `createDataFrame` function, if there's already a local **R** data frame in place, you can use `read.df` if your file is of a format natively readable by SparkR (such as a correctly structured JSON file or Parquet), or you can take a look in the Spark Packages and see if there is any packages made for reading your file.

```
#Create a data frame called "cars" using R's native dataset "mtcars"
cars <- createDataFrame(sqlContext, mtcars)
```

```
Now, you do the same for the `iris` dataset:
```

```
#Create a data frame called "flowers" using R's native dataset "iris"
flowers <- createDataFrame(sqlContext, iris)
```

```
Warning message:
In FUN(X[[i]], ...): Use Sepal_Length instead of Sepal.Length  as column nameWarning message:
In FUN(X[[i]], ...): Use Sepal_Width instead of Sepal.Width  as column nameWarning message:
In FUN(X[[i]], ...): Use Petal_Length instead of Petal.Length  as column nameWarning message:
In FUN(X[[i]], ...): Use Petal_Width instead of Petal.Width  as column name
```

Registering Data Frames as tempTables

One of SparkR's more unique features is the capability to perform SQL queries on Data Frames. To do so, you generate a temporary SQL table (the so called `tempTable`) in Spark. We will go over performing SQL queries in the next Lab.

For now, to register a temporary SQL table, we use the following function:

```
#Create a temporary SQL table called "cars" using our SparkR data frame "cars"
registerTempTable(cars,"cars")
```

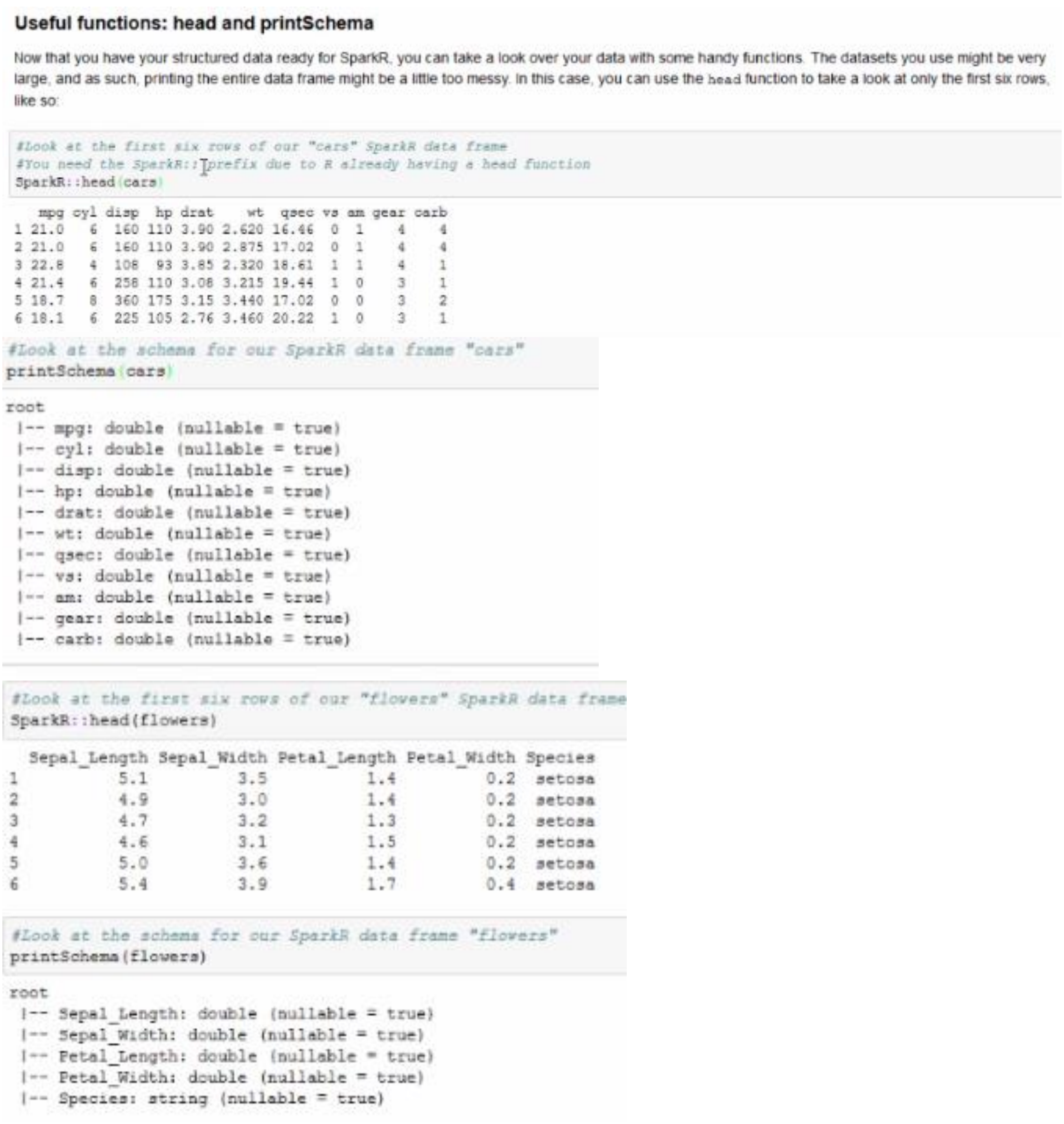
Now, do the same for the `flowers` data frame:

```
#Create a temporary SQL table called "flowers" using our SparkR data frame "flowers"
registerTempTable(flowers,"flowers")
```

```
Error in registerTempTable(flowers, "flowers"): object 'flowers' not found
```

Useful functions: head and printSchema Now that you have your structured data ready for SparkR, you can take a look over your data with some handy functions. The datasets you use might be very large, and as such, printing the entire data frame might be a little too messy. In this case, you can use the `head` function to take a look at only the first six rows, like so:

```
#Look at the first six rows of our "cars" SparkR data frame
#You need the SparkR:: prefix due to R already having a head function
SparkR::head(cars)
```



Useful functions: head and printSchema

Now that you have your structured data ready for SparkR, you can take a look over your data with some handy functions. The datasets you use might be very large, and as such, printing the entire data frame might be a little too messy. In this case, you can use the `head` function to take a look at only the first six rows, like so:

```
#Look at the first six rows of our "cars" SparkR data frame
#You need the SparkR:: prefix due to R already having a head function
SparkR::head(cars)
```

```
   mpg cyl disp  hp drat    wt  qsec vs am gear carb
1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
#Look at the schema for our SparkR data frame "cars"
printSchema(cars)
```

```
root
 |-- mpg: double (nullable = true)
 |-- cyl: double (nullable = true)
 |-- disp: double (nullable = true)
 |-- hp: double (nullable = true)
 |-- drat: double (nullable = true)
 |-- wt: double (nullable = true)
 |-- qsec: double (nullable = true)
 |-- vs: double (nullable = true)
 |-- am: double (nullable = true)
 |-- gear: double (nullable = true)
 |-- carb: double (nullable = true)
```

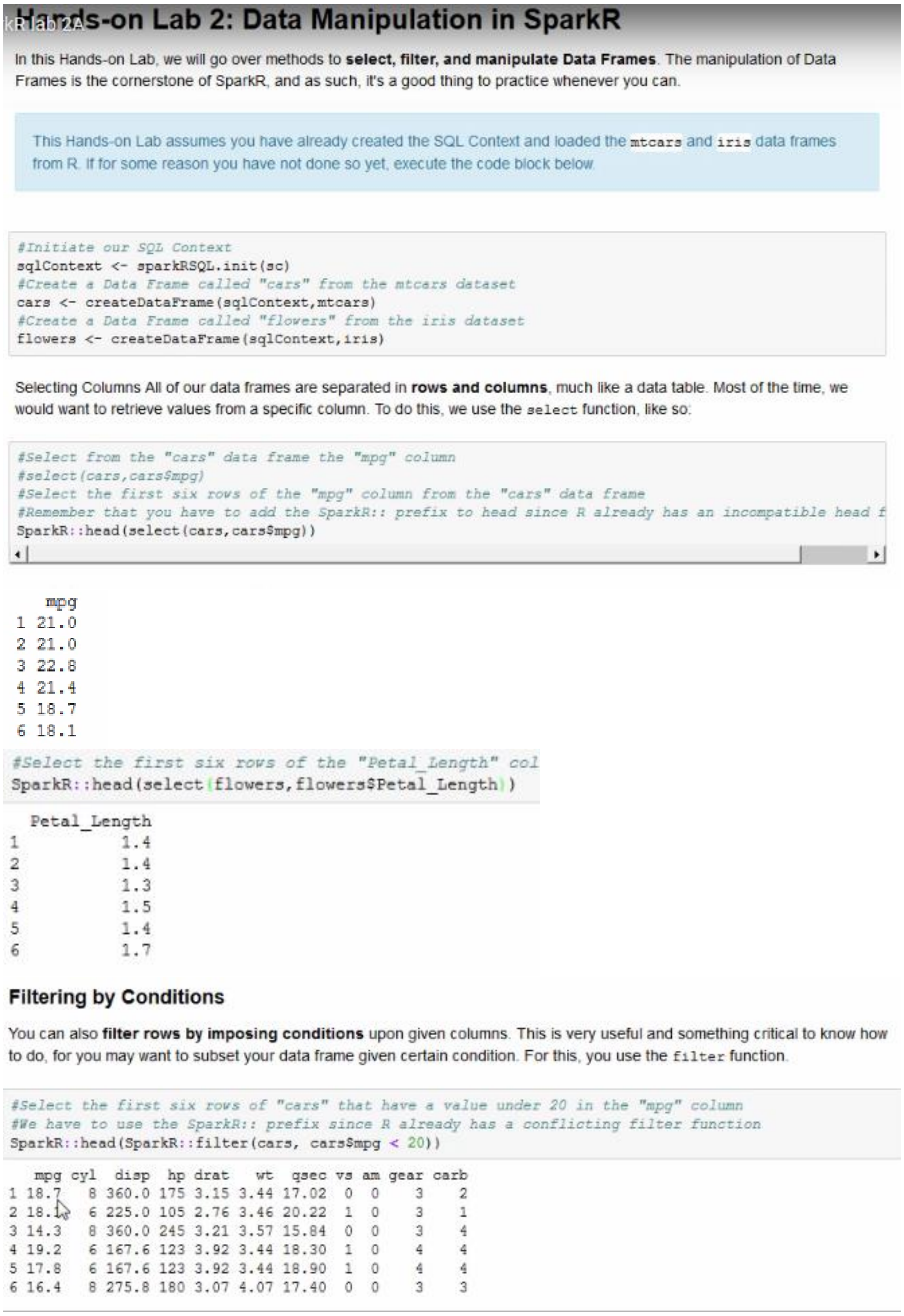
```
#Look at the first six rows of our "flowers" SparkR data frame
SparkR::head(flowers)
```

```
  Sepal_Length Sepal_Width Petal_Length Petal_Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```
#Look at the schema for our SparkR data frame "flowers"
printSchema(flowers)
```

```
root
 |-- Sepal_Length: double (nullable = true)
 |-- Sepal_Width: double (nullable = true)
 |-- Petal_Length: double (nullable = true)
 |-- Petal_Width: double (nullable = true)
 |-- Species: string (nullable = true)
```

>> Lab2:



Hands-on Lab 2: Data Manipulation in SparkR

In this Hands-on Lab, we will go over methods to **select, filter, and manipulate Data Frames**. The manipulation of Data Frames is the cornerstone of SparkR, and as such, it's a good thing to practice whenever you can.

This Hands-on Lab assumes you have already created the SQL Context and loaded the `mtcars` and `iris` data frames from R. If for some reason you have not done so yet, execute the code block below.

```
#Initiate our SQL Context
sqlContext <- sparkRSQL.init(sc)
#Create a Data Frame called "cars" from the mtcars dataset
cars <- createDataFrame(sqlContext,mtcars)
#Create a Data Frame called "flowers" from the iris dataset
flowers <- createDataFrame(sqlContext,iris)
```

Selecting Columns All of our data frames are separated in **rows and columns**, much like a data table. Most of the time, we would want to retrieve values from a specific column. To do this, we use the `select` function, like so:

```
#Select from the "cars" data frame the "mpg" column
#select(cars,cars$mpg)
#Select the first six rows of the "mpg" column from the "cars" data frame
#Remember that you have to add the SparkR:: prefix to head since R already has an incompatible head f
SparkR::head(select(cars,cars$mpg))
```

```
   mpg
1 21.0
2 21.0
3 22.8
4 21.4
5 18.7
6 18.1
```

```
#Select the first six rows of the "Petal_Length" col
SparkR::head(select(flowers,flowers$Petal_Length))
```

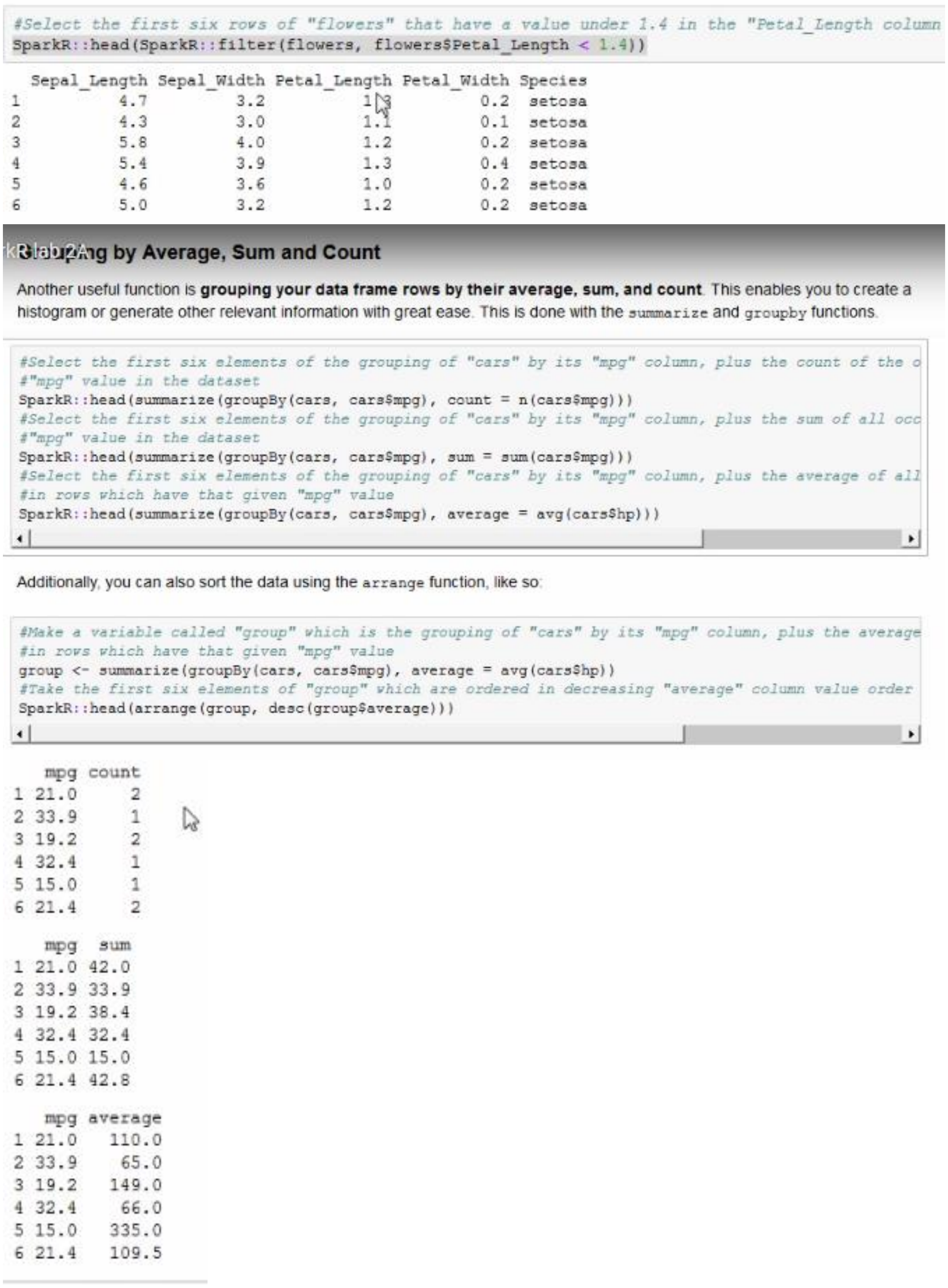
```
  Petal_Length
1          1.4
2          1.4
3          1.3
4          1.5
5          1.4
6          1.7
```

Filtering by Conditions

You can also **filter rows by imposing conditions** upon given columns. This is very useful and something critical to know how to do, for you may want to subset your data frame given certain condition. For this, you use the `filter` function.

```
#Select the first six rows of "cars" that have a value under 20 in the "mpg" column
#We have to use the SparkR:: prefix since R already has a conflicting filter function
SparkR::head(SparkR::filter(cars, cars$mpg < 20))
```

```
   mpg cyl  disp  hp drat   wt  qsec vs am gear carb
1 18.7   8 360.0 175 3.15 3.44 17.02  0  0    3    2
2 18.1   6 225.0 105 2.76 3.46 20.22  1  0    3    1
3 14.3   8 360.0 245 3.21 3.57 15.84  0  0    3    4
4 19.2   6 167.6 123 3.92 3.44 18.30  1  0    4    4
5 17.8   6 167.6 123 3.92 3.44 18.90  1  0    4    4
6 16.4   8 275.8 180 3.07 4.07 17.40  0  0    3    3
```



```
#Select the first six rows of "flowers" that have a value under 1.4 in the "Petal_Length column
SparkR::head(SparkR::filter(flowers, flowers$Petal_Length < 1.4))
```

```
  Sepal_Length Sepal_Width Petal_Length Petal_Width Species
1          4.7         3.2          1.3         0.2  setosa
2          4.3         3.0          1.1         0.1  setosa
3          5.8         4.0          1.2         0.2  setosa
4          5.4         3.9          1.3         0.4  setosa
5          4.6         3.6          1.0         0.2  setosa
6          5.0         3.2          1.2         0.2  setosa
```

Grouping by Average, Sum and Count

Another useful function is **grouping your data frame rows by their average, sum, and count**. This enables you to create a histogram or generate other relevant information with great ease. This is done with the `summarize` and `groupby` functions.

```
#Select the first six elements of the grouping of "cars" by its "mpg" column, plus the count of the o
#"mpg" value in the dataset
SparkR::head(summarize(groupBy(cars, cars$mpg), count = n(cars$mpg)))
#Select the first six elements of the grouping of "cars" by its "mpg" column, plus the sum of all occ
#"mpg" value in the dataset
SparkR::head(summarize(groupBy(cars, cars$mpg), sum = sum(cars$mpg)))
#Select the first six elements of the grouping of "cars" by its "mpg" column, plus the average of all
#in rows which have that given "mpg" value
SparkR::head(summarize(groupBy(cars, cars$mpg), average = avg(cars$hp)))
```

Additionally, you can also sort the data using the `arrange` function, like so:

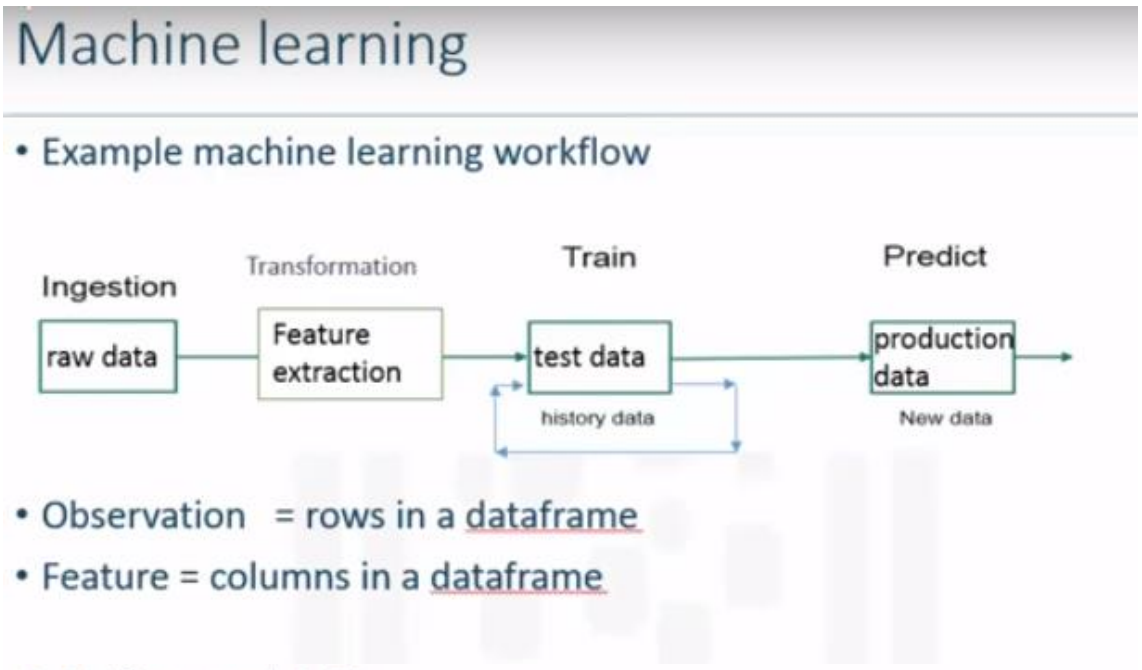
```
#Make a variable called "group" which is the grouping of "cars" by its "mpg" column, plus the average
#in rows which have that given "mpg" value
group <- summarize(groupBy(cars, cars$mpg), average = avg(cars$hp))
#Take the first six elements of "group" which are ordered in decreasing "average" column value order
SparkR::head(arrange(group, desc(group$average)))
```

```
   mpg count
1 21.0     2
2 33.9     1
3 19.2     2
4 32.4     1
5 15.0     1
6 21.4     2

   mpg  sum
1 21.0 42.0
2 33.9 33.9
3 19.2 38.4
4 32.4 32.4
5 15.0 15.0
6 21.4 42.8

   mpg average
1 21.0   110.0
2 33.9    65.0
3 19.2   149.0
4 32.4    66.0
5 15.0   335.0
6 21.4   109.5
```

welcome to lab 2 data manipulation in SparkR in this lab we will go over the methods to select,filter and manipulate data frames The manipulation of data frames is the cornerstone of SparkR we've seen previously and as such a good thing to practice this hands on lab assumes you've already created the sqlContext and loaded the mtcars and iris data frames from R if some reason you haven't done so please execute the code block below so that you will be in a position to do the rest of the lab so we'll run this code just to

make sure we all start off from the same place and there we are successfully
finished now and we have the warning messages about the columns names which
we got before
So selecting columns, all the data frames are separated in rows and columns so
much like a data table
most of the time we want to retrieve values from a specific column and to do
this we use the Select function as per the following block of code
so what we're doing here is selecting from the cars data frame which we
created earlier on the mpg miles per gallon column and that's shown us the
Select cars cars MPG column
to select the first six rows of the mpg column we use the head function and
remember because of the conflicts the head function in R and SparkR we
have to use the SPARKR:: so we'll run that
and there we are
there are the first six rows or the mpg column from the first six rows so now you
can go and do the same thing for the flowers data frame
okay assuming that you've tried to to the code yourself
it should look like this should look the same as this actually
so let's run that and there we have the first six rows on the column of
petal_length
now we're going to look at how to filter rows by imposing a condition on a
particular column or set of columns
it's crucial to know how to do this so that you can sub set your data frame for
a given set of conditions
to do this we use the filter function here we are here we have the cell
and what we're going to do select the first rows of cars that have a value of
less than 20 in the miles per gallon column and you can see here there we
have the condition of less than 20
so let's run that and there we are
there's the miles per gallon column and we haven't restricted in this case
to just that column we have the whole row and here we can see the first six rows
that have a miles per gallon of less than 20
ok so now it's your turn to do a similar thing with the flowers data frame in
this case having a condition of the petal length column being less than 1.4
ok again assuming that you've done that exercise
this is what your code should look the same as will run that
and there we are petal length less than 1.4 now we're going to look at another
useful function and that is the grouping of the data frame rows by their
average or sum or count enabling you to create a histogram or other relevant
information and this is done with the summarized and group by functions
so here we are the examples that we are going to use and the first one is
selecting the first six elements of the grouping of cars by the mpg column
plus a count of the number of MPG values in that data set
after that I'm going to select the first six elements of the group of cars by the
mpg column plus the sum rather than the count of the occurrences and then finally
going to select the first six elements of the grouping plus the average of all
the mpgs in that row of a given value
ok so let's let's run that
it's running and now we have the outputs as you can see here is the count of each
particular grouping
so there's two at 21 miles per gallon, one at 33.9 to at 19.2 and so
on and then we have the sum so 2 times 21 is 42, 1 times 39 and so on

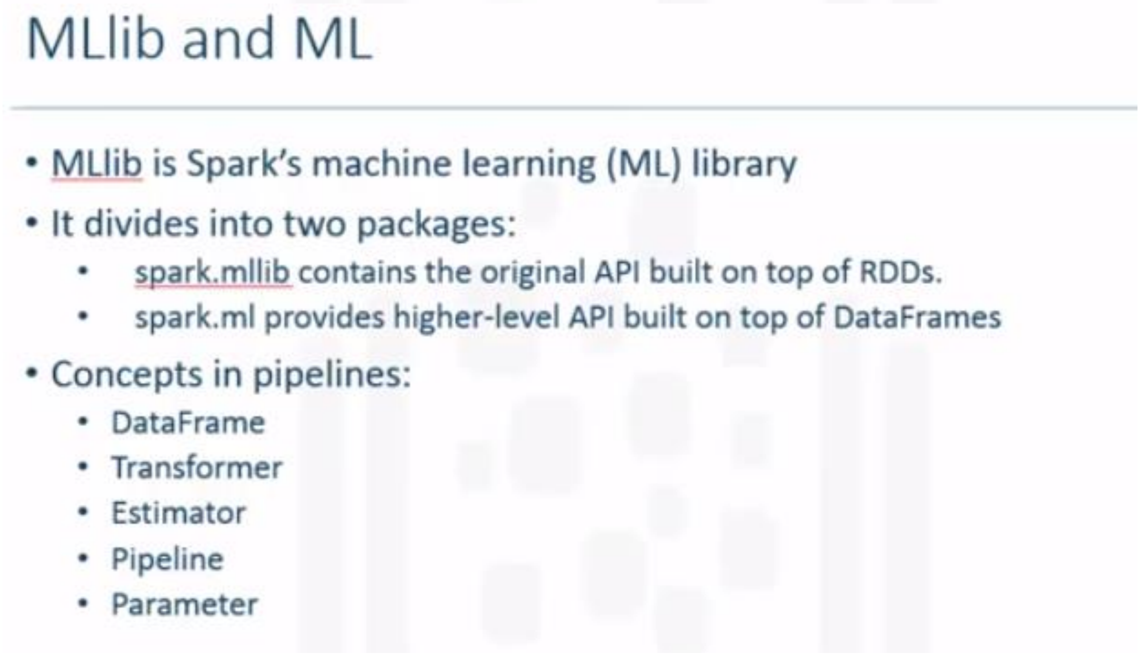
MODULE 3 MACHINE LEARNING IN SPARKR



Machine learning

- Example machine learning workflow

- Observation = rows in a dataframe
- Feature = columns in a dataframe

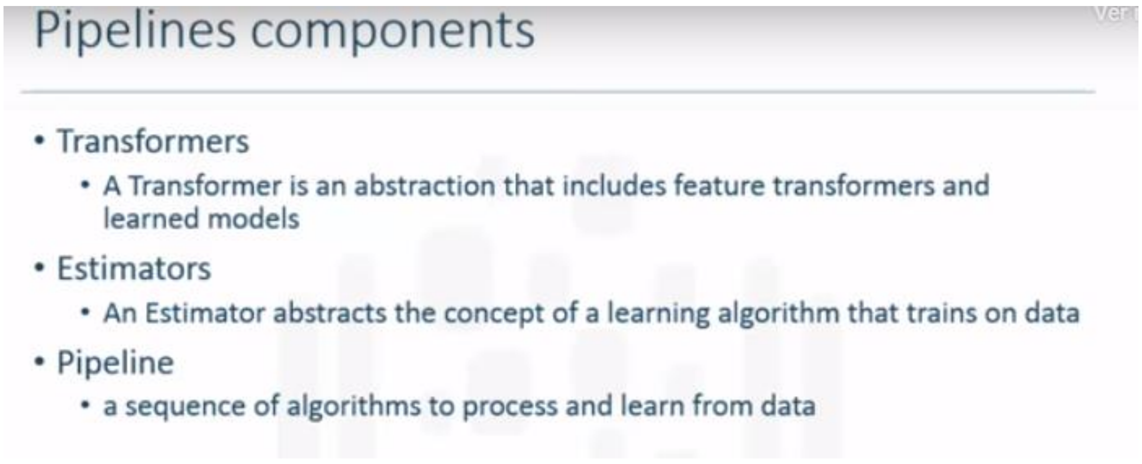


MLlib and ML

- MLlib is Spark's machine learning (ML) library
- It divides into two packages:
  - spark.mllib contains the original API built on top of RDDs.
  - spark.ml provides higher-level API built on top of DataFrames
- Concepts in pipelines:
  - DataFrame
  - Transformer
  - Estimator
  - Pipeline
  - Parameter

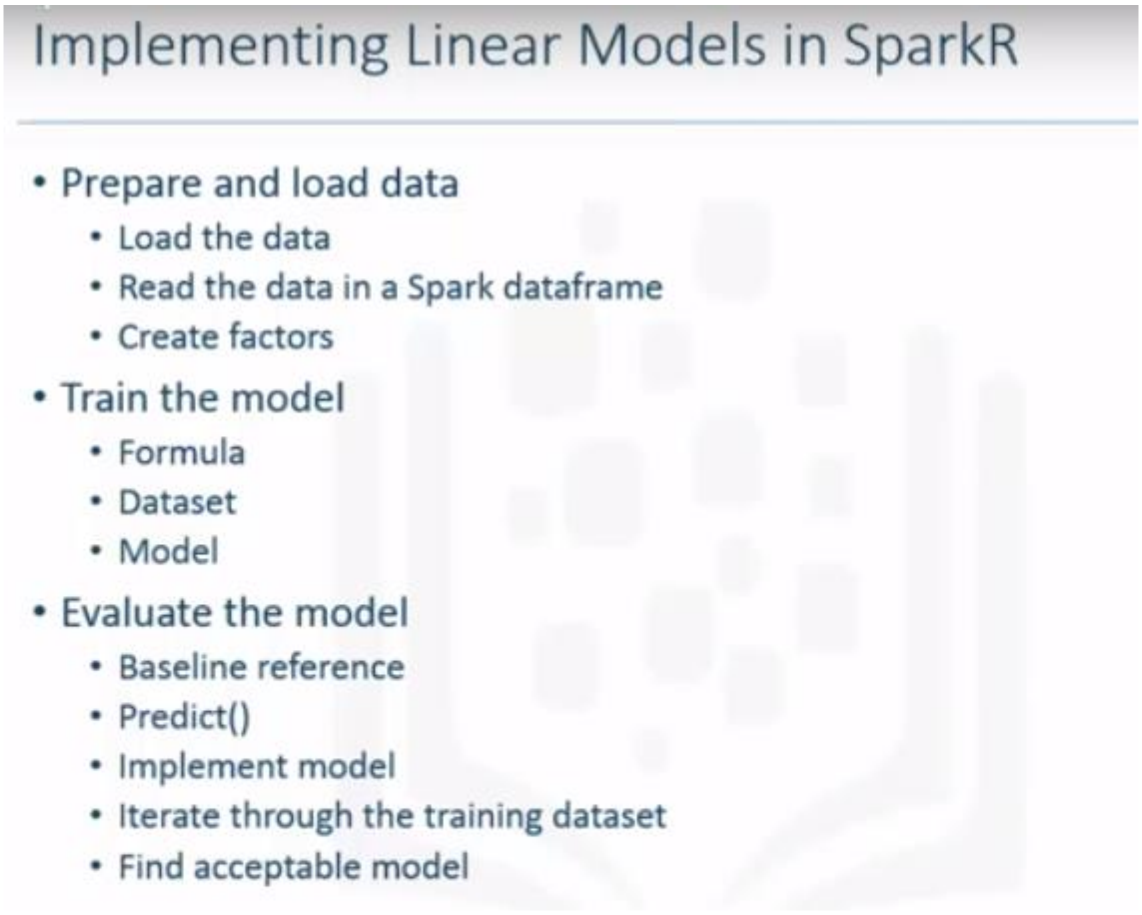
Welcome back to the course, Introduction to analyzing Big Data in R using Apache Spark. In this lesson, lesson 3, we're going to be looking at machine learning in SparkR. The aim of this lesson is to understand machine learning, and also how to use the GLM, or generalized linear models within SparkR. Within SparkR for machine learning, we have Spark MLlib. This is a module, library, an extension of Apache Spark to provide machine learning algorithms on top of SparkRs RDD abstraction. SparkML is a module to provide distributed machine learning algorithms on top of Spark dataframes. There is a distinction between the two there. A machine learning algorithm take data as input and it produces a model of that data as output. This model can then be used to make predictions out of a variety of data sets. Machine learning uses large datasets to identify or infer patterns and make decisions. This automated decision making is what makes machine learning so efficient at predicting future events and trends. Machine learning workflows involve using test data

to evaluate various models until a best model is found and this can then be applied to the prediction data. An observation is used to learn about or evaluate an items target value. Spark models observations as rows in a dataframe. A features is an attribute of an observation. Spark models features as a column in a dataframe. MLlib is Sparks machine learning, ML library.
It's goal is to make practical machine learning scalable and easy. It consists of common learning
algorithms and utilities including classification regression, clustering, collaborative filtering, dimensionality reduction, as well as lower level optimization in primitives, and higher level pipeline APIs. It is divided into two packages, MLlib and ML. spark mllib contains the original API built on top of RDDs. spark ml pprovides higher-level API built on top of DataFrames. Spark ML standardizes APIs from machine learning algorithms to make it easier to combine multiple algorithms into a single pipeline, or workflow. Using spark.ml is recommended because with dataframes, the API is more versatile, flexible and faster. So we will concentrate on ML in this lesson. A pipeline is specified as a sequence of stages and each stage is either a transformer or an estimator. These stages are in an order with the input dataframe being transformed as it passes through each stage. Let's look at the concepts in pipelines. Firstly we have the dataframe. Spark ML uses the dataframe from SparkSQL as an ML dataset. This can hold a variety of data types, for example a dataframe
can add different columns storing text, feature vectors, tool labels, and predictions. Next we have the transformer. A transformer is an algorithm which can transform one dataframe into another dataframe. For example, an ML model is a transformer, which transforms a dataframe with features into a dataframe with predictions. An estimator is an algorithm which can fit onto a datraframe to produce a transformer. For example, a learning algorithm is an estimator which trains on a dataframe and produces a model. A pipeline is a chain of multiple transformers and estimators which together specify an ML workflow. Finally we have a parameter. All transformers and estimators now share a common API for specifying parameters.



Pipelines components

- Transformers
  - A Transformer is an abstraction that includes feature transformers and learned models
- Estimators
  - An Estimator abstracts the concept of a learning algorithm that trains on data
- Pipeline
  - a sequence of algorithms to process and learn from data

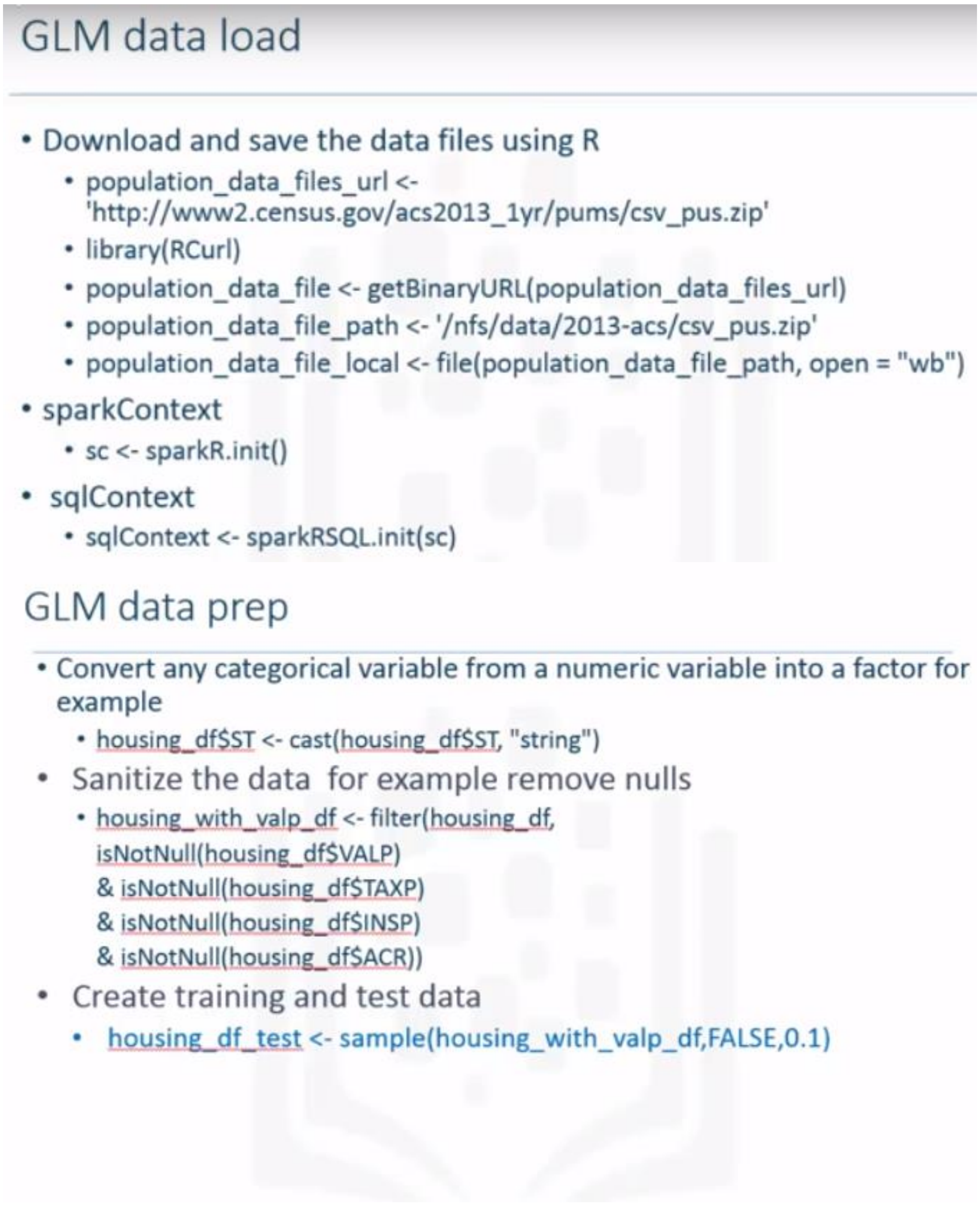


Implementing Linear Models in SparkR

- Prepare and load data
  - Load the data
  - Read the data in a Spark dataframe
  - Create factors
- Train the model
  - Formula
  - Dataset
  - Model
- Evaluate the model
  - Baseline reference
  - Predict()
  - Implement model
  - Iterate through the training dataset
  - Find acceptable model

Previous
Next
SPARKR LESSON 3 PART 2 (5:04)
Skip to a navigable version of this video's transcript.

Pause4:51 / 5:03SpeedSpeed0.50xVolumeMaximum Volume.HDHigh Definition offFill browserTurn off transcriptsSkip to end of transcript.
Now we're going to look more in depth at the pipeline components. Transformers are an abstraction
that includes feature transformers and learned models. Technically a transformer implements a method, transform, which converts one dataframe into another dataframe, generally by appending
one or more columns. For example, a feature transformer might take a dataframe, read a column for example text column, and map it into a new column, for example feature vector, and output a new dataframe with the mapped column appended. A learned model might take a dataframe, read a column containing the featured vectors, predict the label for each feature vector, and output a new dataframe with predicted labels appended as a column. Estimators abstracts the concept of a learning algorithm or any algorithm that fits or trains on data. Technically an estimator implements a method, fit, which accepts a dataframe and produces a model, which is a transformer. For example, a learning algorithm such as logistic regression is an estimator and calling fit, or the method fit, trains a logistic regression model, which is a model and hence a transformer. And lastly we have a pipeline. In machine learning, it is common to run a sequence of machine learning algorithms to process and learn from data. For example, a simple text document processing workflow might include several stages such as split each documents text into words, convert each documents words into numerical feature vectors, learn a prediction model using the feature vectors and labels. Spark ML represents such a workflow as a pipeline, which consists of a sequence of pipeline stages, with transformers and estimators to be run in a specific order. So let's look at implementing linear models in SparkR. Generalized linear models, or GLMs, unify various statistical models such as linear and logistic regression through the specification of a model family and link function. In R, such models can be fitted by passing an R model formula family and training dataset to the GLM function. So let's look at the steps to implement a GLM in SparkR. As per previous lessons, the first steps to prepare and load the data are to create the SparkContext and SQLContext, then load the data, sanitize the data, read the data into a Spark dataframe, and convert any categorical variable from a numeric variable into a factor. Next we have to train the model. In this, we use the MLlib by calling the GLM method with a formula specifying the model variables. MLlib caches the input dataframe and launches a series of Spark jobs to fit the model over the distributed dataset. We call the GLM method with the following parameters, a formula, the dataset we want
to use to train the model, and the type of model, Gaussian or binomial. To evaluate the model, we would first obtain a value that we can use as a reference of a base predictor model, then we can use the predict method just like in R. We can pass in the training data or another dataframe that contains test data. And the return dataframe contains the original columns in addition to the label, the features, and the predict value. And we will keep using this predict method on this test dataset for our evaluation. Before implementing
the model into production, we would iterate through the training and evaluation cycles with different parameters until a model is found with acceptable results. Obviously there is no point implementing a model which doesn't reflect the real world.

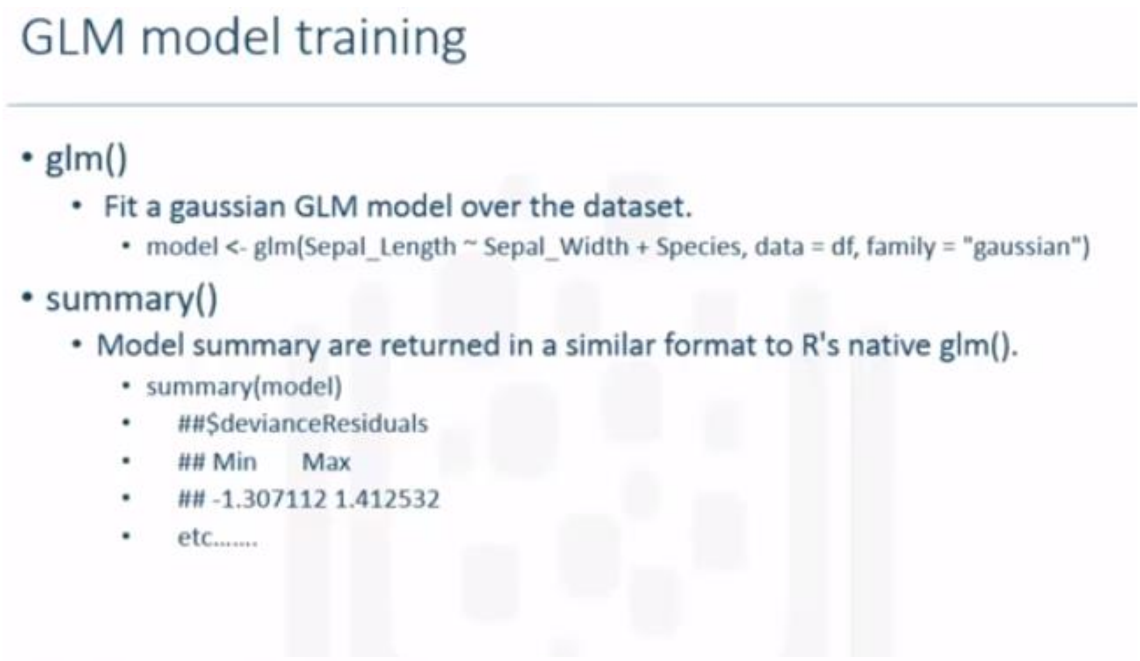


GLM data load

- Download and save the data files using R
  - population_data_files_url <- 'http://www2.census.gov/acs2013_1yr/pums/csv_pus.zip'
  - library(RCurl)
  - population_data_file <- getBinaryURL(population_data_files_url)
  - population_data_file_path <- '/nfs/data/2013-acs/csv_pus.zip'
  - population_data_file_local <- file(population_data_file_path, open = "wb")
- sparkContext
  - sc <- sparkR.init()
- sqlContext
  - sqlContext <- sparkRSQL.init(sc)

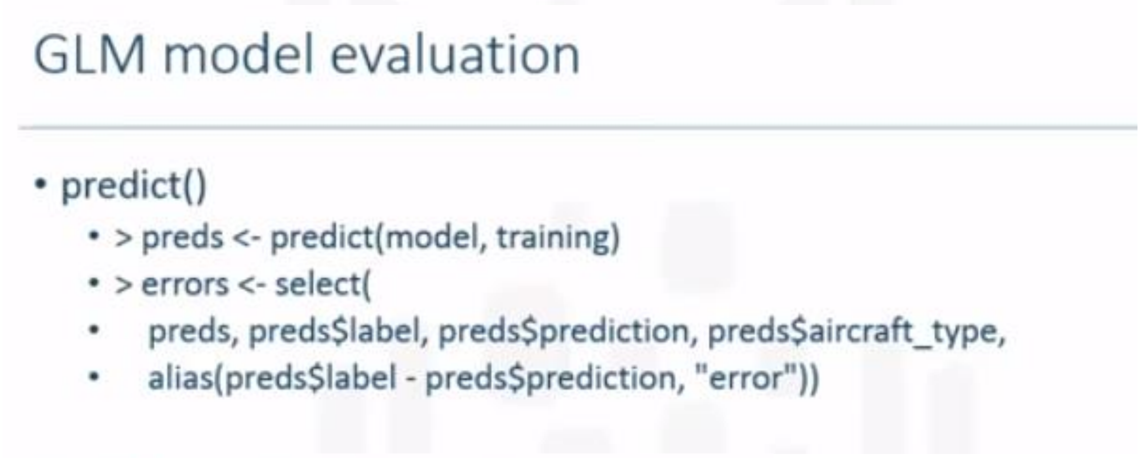
GLM data prep

- Convert any categorical variable from a numeric variable into a factor for example
  - housing_df$ST <- cast(housing_df$ST, "string")
- Sanitize the data for example remove nulls
  - housing_with_valp_df <- filter(housing_df,
    isNotNull(housing_df$VALP)
    & isNotNull(housing_df$TAXP)
    & isNotNull(housing_df$INSP)
    & isNotNull(housing_df$ACR))
- Create training and test data
  - housing_df_test <- sample(housing_with_valp_df,FALSE,0.1)



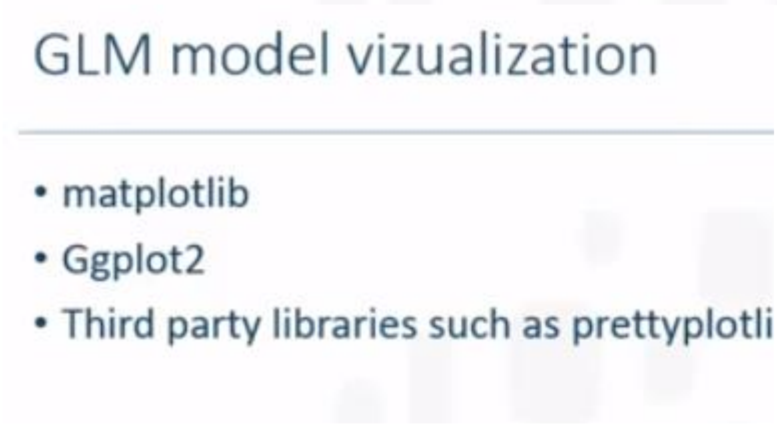
GLM model training

- glm()
  - Fit a gaussian GLM model over the dataset.
    - model <- glm(Sepal_Length ~ Sepal_Width + Species, data = df, family = "gaussian")
- summary()
  - Model summary are returned in a similar format to R's native glm().
    - summary(model)
    - ##$devianceResiduals
    - ## Min   Max
    - ## -1.307112 1.412532
    - etc.......



GLM model evaluation

- predict()
  - > preds <- predict(model, training)
  - > errors <- select(
  - preds, preds$label, preds$prediction, preds$aircraft_type,
  - alias(preds$label - preds$prediction, "error"))



GLM model vizualization

- matplotlib
- Ggplot2
- Third party libraries such as prettyplotli

About this course
Module 1: Introduction to SparkR
Module 2: Data manipulation in SparkR
Module3: Machine learning in SparkR
Learning Objectives

SparkR lesson 3 part 1 (5:10)

SparkR lesson 3 part 2 (5:04)

SparkR lesson 3 part 3 (5:33)current section

Hands on Lab 3

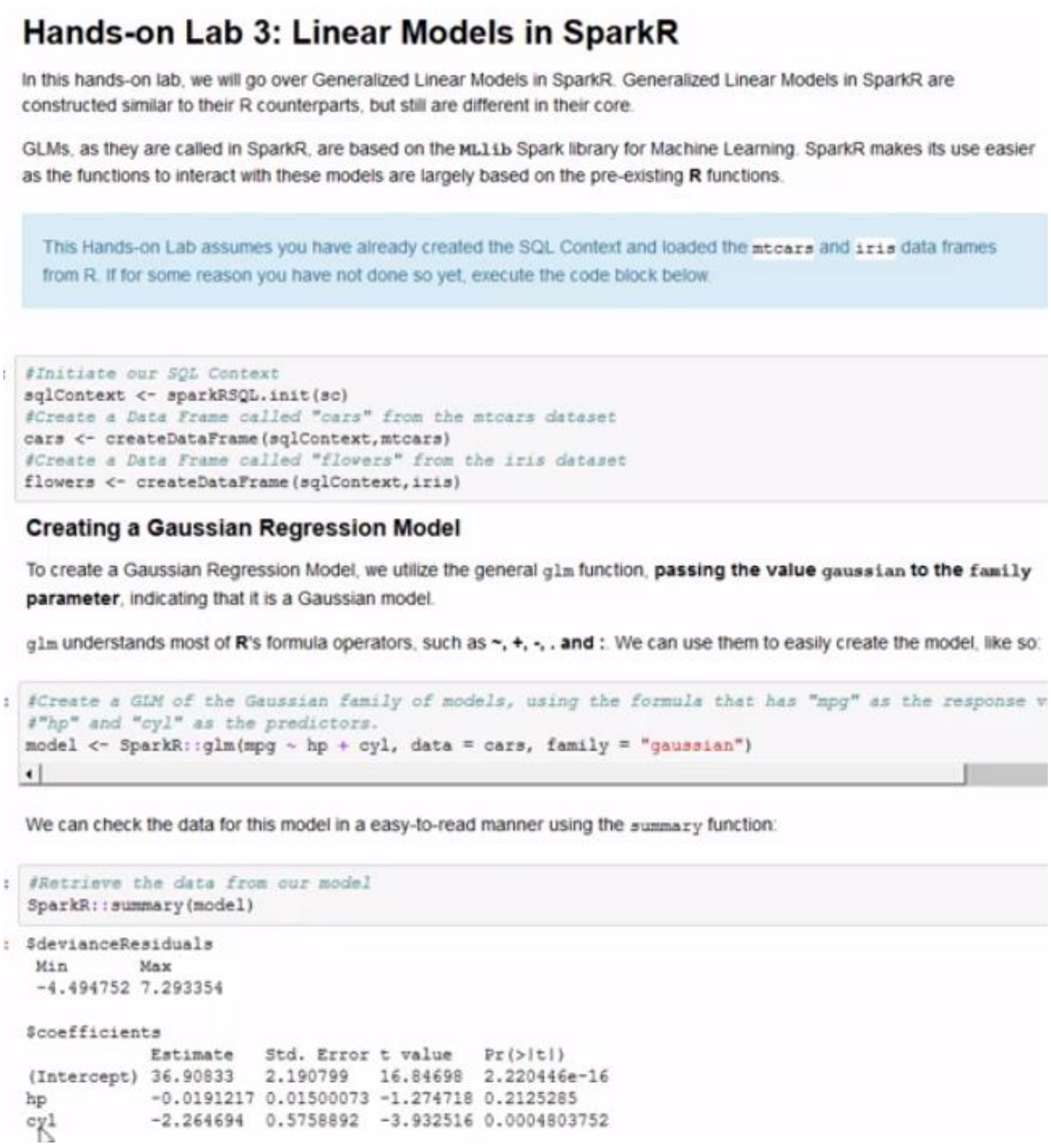
Graded Review Questions

Review QuestionsThis content is graded
Final Exam
Completion Certificate
Course Survey and Feedback
Previous
Next
SPARKR LESSON 3 PART 3 (5:33)
Skip to a navigable version of this video's transcript.

Pause5:26 / 5:32SpeedSpeed0.50xVolumeMaximum Volume.HDHigh Definition offFill browserTurn off transcriptsSkip to end of transcript.
The following slides will guide you through the steps necessary to create a GLM. It assumes you are already familiar with loading the data as per the previous lessons. Accordingly, this slide is a recap of previous information. If the data is from a data source, then we would download the data file using R. It's possible that the data source is local, in which case a download would not be necessary. In order to explore our data, we need to load it into a Spark SQL dataframe. To do that, we would need to initialize a SparkContext, followed by initializing a SQLContext. Depending on the shell we will be using, we may or may not have to. If we are using a R shell, SparkR shell, these contexts would already be initialized. The SparkR implementation tries to create a language that is familiar to R users, not by just using a dataframe abstraction, but by defining a series of functions equivalent to the regular R ones. In addition, SparkR dataframes are distributed data structures that open the door to scalable data analysis. We need to create the factors, the data might need to be sanitized, remove any invalid data or characters. Unfortunately, there are no split functions in SparkR, but we can use the sample function to prepare two data sets for training and testing. SparkR allows us to fit a generalized linear model over dataframes using the GLM function. Under the hood, SparkR is using MLlib to train a model with the specified
family. Currently the Gaussian and Binomial families are supported, supports a subset of the available R formula operators for model fitting including full stops, colons, pluses, and minuses. When we train a model, I call it GLM function with the parameters that we talked about previously, a formula, a data set to train a model, and the type of model, Gaussian or Binomial. The summary function gives a summary of a model produced by GLM method as with Rs native models, coefficients can be retrieved using the summary functions. For Gaussian GLMs, it returns a list with deviance residuals and coefficients components. For Binomial GLMs, it returns a list with coefficients components, which gives the estimated coefficients. To evaluate our models, we use the predict function just like in R. We can pass in the training data, or another dataframe that contains test data that we need to use. And finally having to have gone through all the steps and implemented a model, since the return dataframe contains the original columns, in addition to the labels, features, and predictive
value, it's easy to inspect the results. It's also relatively easy to create plots quickly using Pandas and matplotlib. Ggplot2 is one of the most popular visualization packages for R. It makes it easy to produce high quality graphs using data represented in R dataframes.

Ggplot2.sparkR is an extension to the original Ggplot2 package and can seamlessly handle both R dataframes and SparkR dataframes with no modifications necessary to the original API. Ggplot2.sparkR requires no additional training for existing R users. If you want to make your plots, your visualizations even better in terms of presentation, it's possible to install third party libraries to help you with this. One of the libraries is the prettyplotlib, which enhances the matplotlib plots with better default colors, etc. This sort of thing can be extremely valuable if your notebook is to be used for presentations. That brings us to the end of lesson three. We should now be able to describe machine learning and be able to use the GLM model within SparkR. Thank you for attending this course.

>>Lab:



Hands-on Lab 3: Linear Models in SparkR

In this hands-on lab, we will go over Generalized Linear Models in SparkR. Generalized Linear Models in SparkR are constructed similar to their R counterparts, but still are different in their core.

GLMs, as they are called in SparkR, are based on the `MLlib` Spark library for Machine Learning. SparkR makes its use easier as the functions to interact with these models are largely based on the pre-existing **R** functions.

This Hands-on Lab assumes you have already created the SQL Context and loaded the `mtcars` and `iris` data frames from R. If for some reason you have not done so yet, execute the code block below.

```
#Initiate our SQL Context
sqlContext <- sparkRSQL.init(sc)
#Create a Data Frame called "cars" from the mtcars dataset
cars <- createDataFrame(sqlContext,mtcars)
#Create a Data Frame called "flowers" from the iris dataset
flowers <- createDataFrame(sqlContext,iris)
```

Creating a Gaussian Regression Model

To create a Gaussian Regression Model, we utilize the general `glm` function, **passing the value `gaussian` to the `family` parameter**, indicating that it is a Gaussian model.

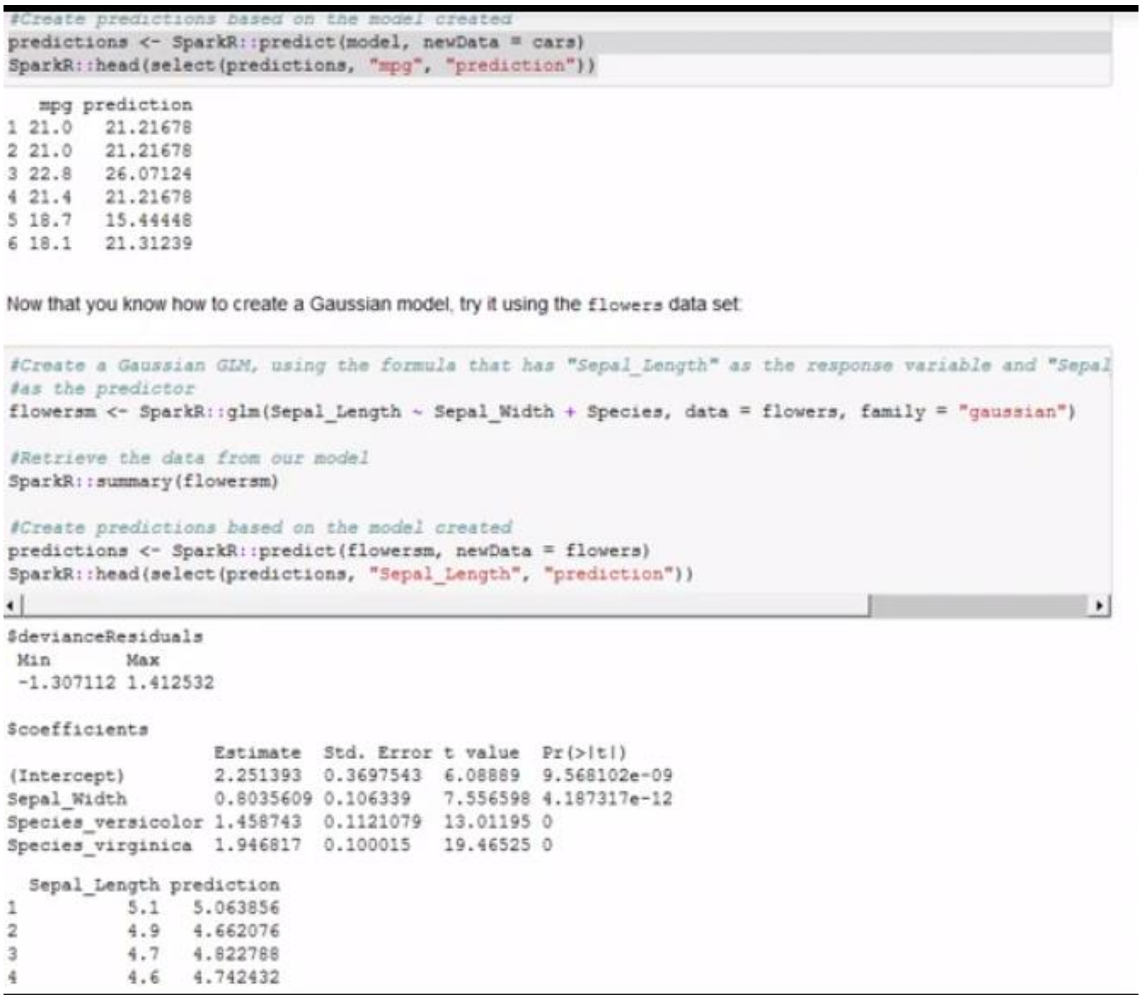
`glm` understands most of **R**'s formula operators, such as **~, +, -, . and :**. We can use them to easily create the model, like so:

```
#Create a GLM of the Gaussian family of models, using the formula that has "mpg" as the response v
#"hp" and "cyl" as the predictors.
model <- SparkR::glm(mpg ~ hp + cyl, data = cars, family = "gaussian")
```

We can check the data for this model in a easy-to-read manner using the `summary` function:

```
#Retrieve the data from our model
SparkR::summary(model)
```

```
$devianceResiduals
 Min       Max
 -4.494752 7.293354

$coefficients
            Estimate   Std. Error t value   Pr(>|t|)
(Intercept) 36.90833   2.190799   16.84698  2.220446e-16
hp          -0.0191217 0.01500073 -1.274718 0.2125285
cyl         -2.264694  0.5758892  -3.932516 0.0004803752
```



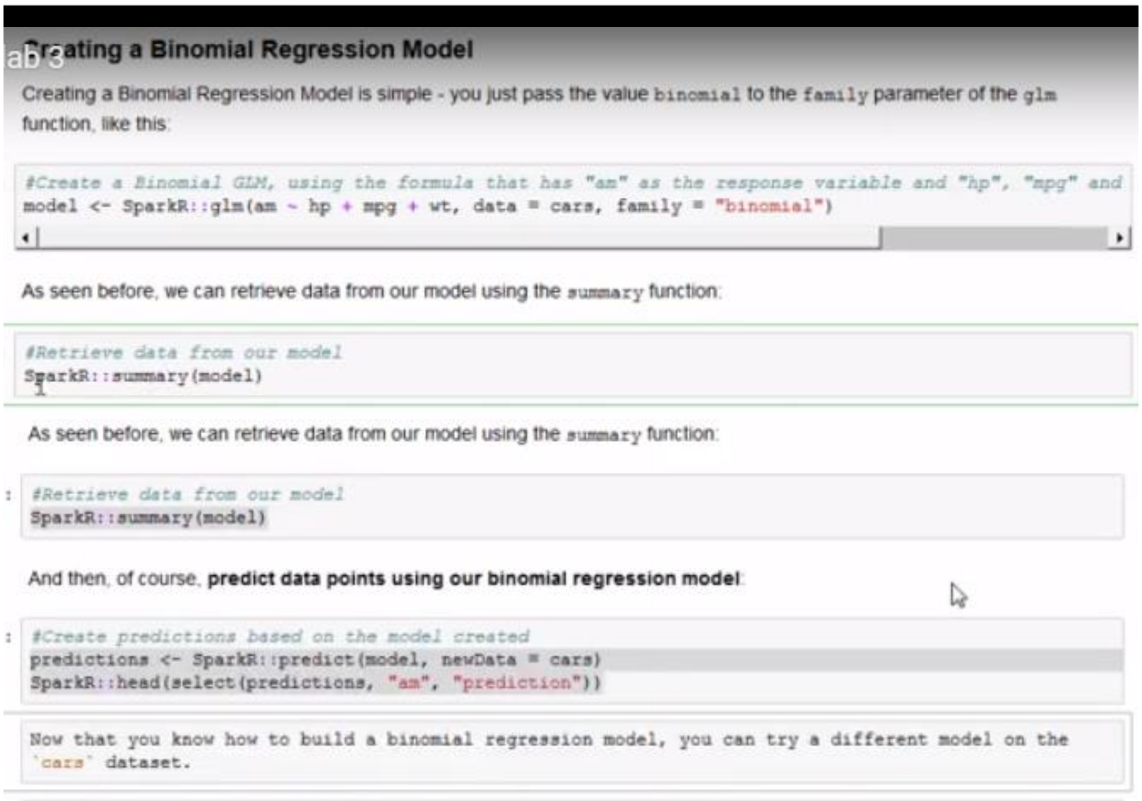
```
#Create predictions based on the model created
predictions <- SparkR::predict(model, newData = cars)
SparkR::head(select(predictions, "mpg", "prediction"))
```

```
   mpg prediction
1 21.0   21.21678
2 21.0   21.21678
3 22.8   26.07124
4 21.4   21.21678
5 18.7   15.44448
6 18.1   21.31239
```

Now that you know how to create a Gaussian model, try it using the `flowers` data set:

```
#Create a Gaussian GLM, using the formula that has "Sepal_Length" as the response variable and "Sepal
#as the predictor
flowersm <- SparkR::glm(Sepal_Length ~ Sepal_Width + Species, data = flowers, family = "gaussian")

#Retrieve the data from our model
SparkR::summary(flowersm)

#Create predictions based on the model created
predictions <- SparkR::predict(flowersm, newData = flowers)
SparkR::head(select(predictions, "Sepal_Length", "prediction"))
```

```
$devianceResiduals
 Min       Max
 -1.307112 1.412532

$coefficients
                   Estimate  Std. Error t value  Pr(>|t|)
(Intercept)        2.251393  0.3697543  6.08889  9.568102e-09
Sepal_Width        0.8035609 0.106339   7.556598 4.187317e-12
Species_versicolor 1.458743  0.1121079  13.01195 0
Species_virginica  1.946817  0.100015   19.46525 0

  Sepal_Length prediction
1          5.1   5.063856
2          4.9   4.662076
3          4.7   4.822788
4          4.6   4.742432
```



Creating a Binomial Regression Model

Creating a Binomial Regression Model is simple - you just pass the value `binomial` to the `family` parameter of the `glm` function, like this:

```
#Create a Binomial GLM, using the formula that has "am" as the response variable and "hp", "mpg" and
model <- SparkR::glm(am ~ hp + mpg + wt, data = cars, family = "binomial")
```

As seen before, we can retrieve data from our model using the `summary` function:

```
#Retrieve data from our model
SparkR::summary(model)
```

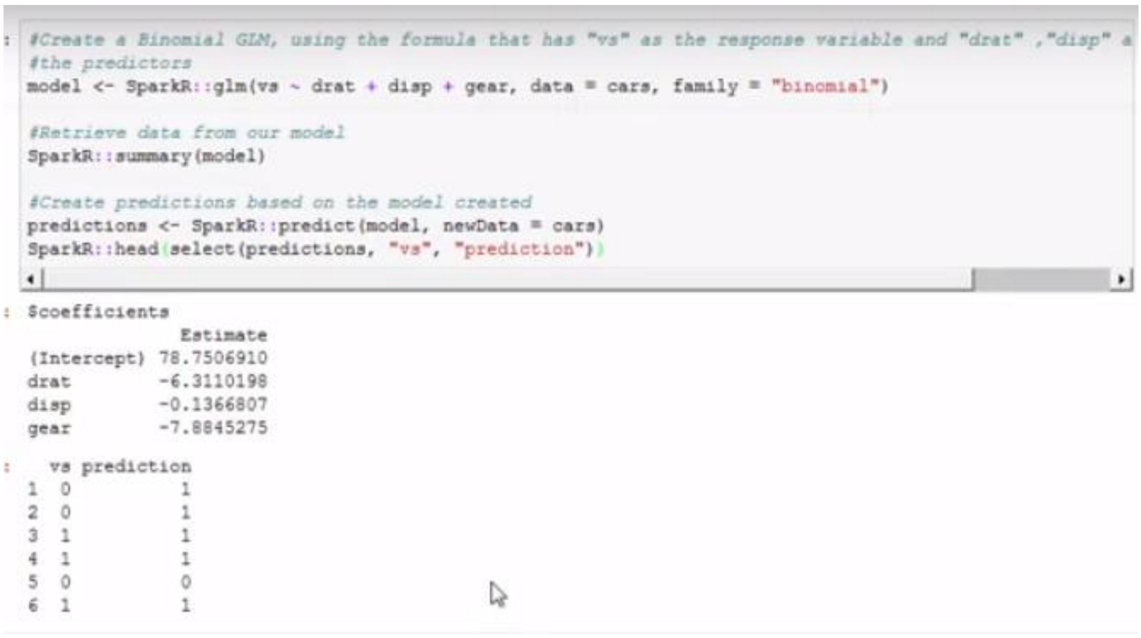
As seen before, we can retrieve data from our model using the `summary` function:

```
#Retrieve data from our model
SparkR::summary(model)
```

And then, of course, **predict data points using our binomial regression model**:

```
#Create predictions based on the model created
predictions <- SparkR::predict(model, newData = cars)
SparkR::head(select(predictions, "am", "prediction"))
```

```
Now that you know how to build a binomial regression model, you can try a different model on the
`cars` dataset.
```



```
#Create a Binomial GLM, using the formula that has "vs" as the response variable and "drat" ,"disp" a
#the predictors
model <- SparkR::glm(vs ~ drat + disp + gear, data = cars, family = "binomial")

#Retrieve data from our model
SparkR::summary(model)

#Create predictions based on the model created
predictions <- SparkR::predict(model, newData = cars)
SparkR::head(select(predictions, "vs", "prediction"))
```

```
$coefficients
              Estimate
(Intercept) 78.7506910
drat        -6.3110198
disp        -0.1366807
gear        -7.8845275
```

```
  vs prediction
1  0          1
2  0          1
3  1          1
4  1          1
5  0          0
6  1          1
```

welcome to the third and final lab linear models in SparkR in this lab we
are going to look at the generalized linear models in SparR or GLMs
as they're known for short
these are based on their R counterparts so there's a lot of
similarities in the construction but there are obviously some differences
since we're running in the spark environment
GLMs are based on the MLlib spark library for machine learning and SparkR
makes it easy to use these functions interacting direct with them rather than the
existing R functions
that's because of the easier
usability of the spark environment as before we assume that you've already
done the previous exercises but just in case we have this cell here which will
do with all that work for you so that you can just do lab three shift and enter
there we are
so now we are ready to begin lab three first section we're going to do is
creating a Gaussian regression model to create a Gaussian regression model we
utilize the general glm function we pass the value of gaussian to the
family parameter indicating that this is a Gaussian model glm understands most of
R formula operators such as tild, plus, minus, and, etc. we
can use them easily to create model like in this following cell
so here we are going to create a gaussian model using the formula that
has the mpg column as the response variable and horsepower and cylinders as
the predictors
here you can see the family parameter specifically saying that this is going
to be a gaussian model and here we have the predictors and here we have the data
frame that we are going to use so let's run that
now thats completed as we can see 2 is there not the star and that shows that
we have completed and there are no error messages
so now we're going to check the data in the model using the summary function
there we are the min and the max and we have the coefficients for the horsepower
and the cylinder
so now that we know how to create a model we can use it for predicting data
points using the predict function so here we are at this cell here we are going

to create predictions based on the model created or just created
there's the model we're talking about the new data being the cars data frame
and there's the predict and then we're going to use the head function to look
at the first six rows so let's run that now we are
everything looks correct so now let's try and use it on the flowers data set is
where you can put your own code in get back to you when you have finished
okay assuming that you have done the exercise here is the answer
and your should show the same
there we are the answer as we expected so now we're going to look at creating
a binomial regression model and this is simple because following the same
constructed before instead of gaussian we're going to pass binomial to the
family parameter of the glm function as we can see here so
run this
that has run, now we're going to retrieve the data exactly the same as before the
coefficients the horsepower miles per gallon
weight so then where we're going to do predict the data points using the
binomial regression model again this is all very similar to what we've just done
and should be very familiar to R users
now that you know how to build a binomial model you can try and create a
different model on the cars date set in the following cell
assuming the do did this exercise then here is the answer using vs as our
response variable and drat, disp and gear as the predictors there we are
there we have the result
so that wraps up lab 3 & wraps up the SparkR course
if you wish to learn any more about the documentation you can always go to the
Apache spark website or you can go to Big Data University where there are lots of courses to
help
you
thank youmml