Procesamiento de datos masivos

Jesús Morán

viu Universidad Internacional de Valencia

12.01.22

- R vs SparkR
- Instalación SparkR
- Operaciones SparkR
- Visualización de datos
- Machine Learning

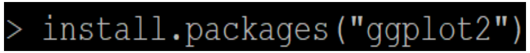
R

- R:
  - No soporta grandes cantidades de datos
  - Muy utilizado por analistas de datos
  - Muchas librerías
  - data.frame

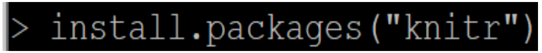
- SparkR:
  - □ Soporta grandes cantidades de datos
  - □ Tiene objeto DataFrame
    - ~ data.frame de R pero limitado
    - Varias Row + Schema
    - Es distribuido
    - Se puede convertir en data.frame
  - □ Hay operaciones Spark que no se pueden hacer con SparkR

SparkR: Instalación

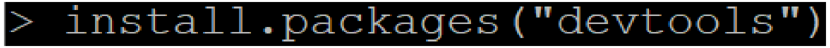
- Instalar Spark
- Instalar R: (en todos los servidores del cluster)
  - □ Repositorio de R: `yum install epel-release`
  - □ Instalar R: `yum install R`
- Instalar librerías: (en todos los servidores del cluster)
  - □ Entrar en R e instalar



```
> install.packages("ggplot2")
```



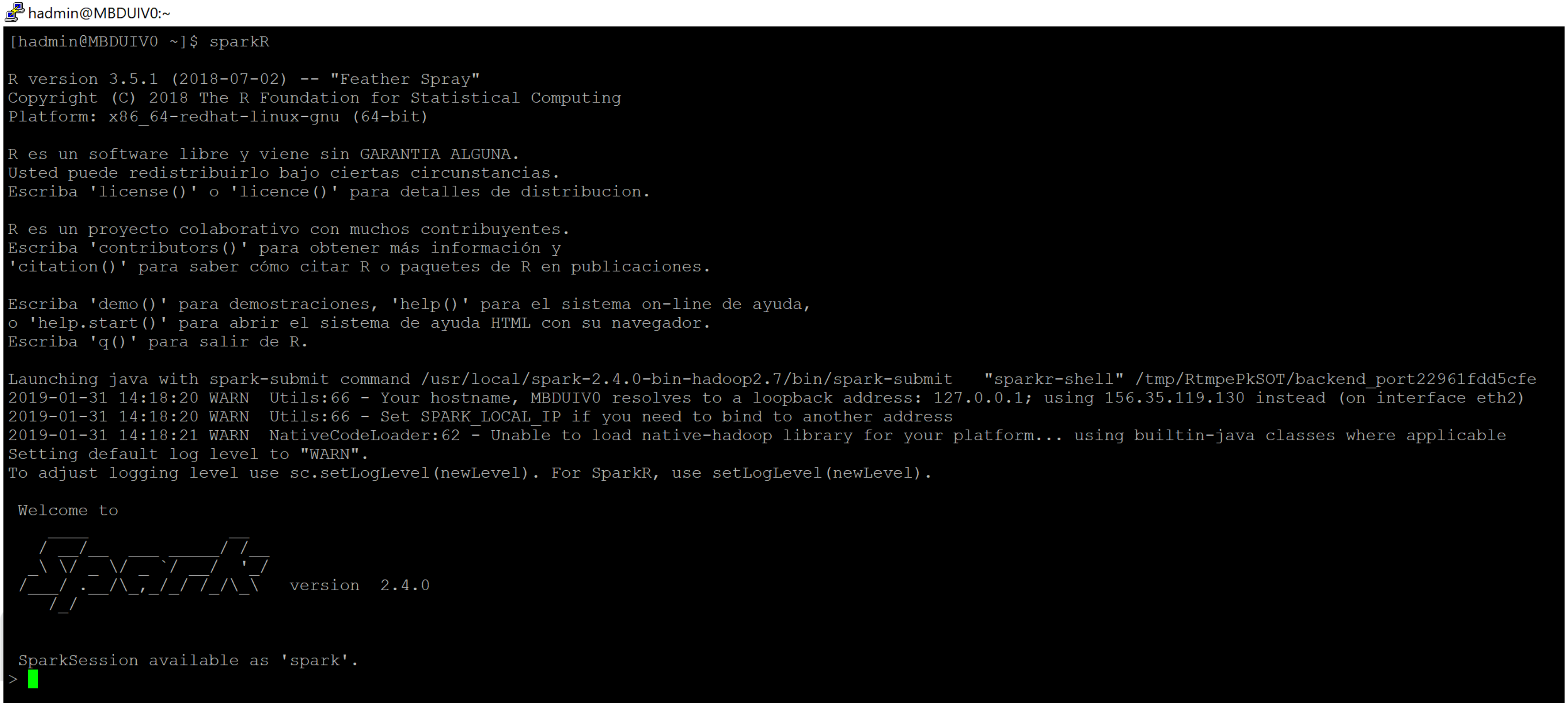
```
> install.packages("knitr")
```



```
> install.packages("devtools")
```

SparkR: Ejecución

- SparkR

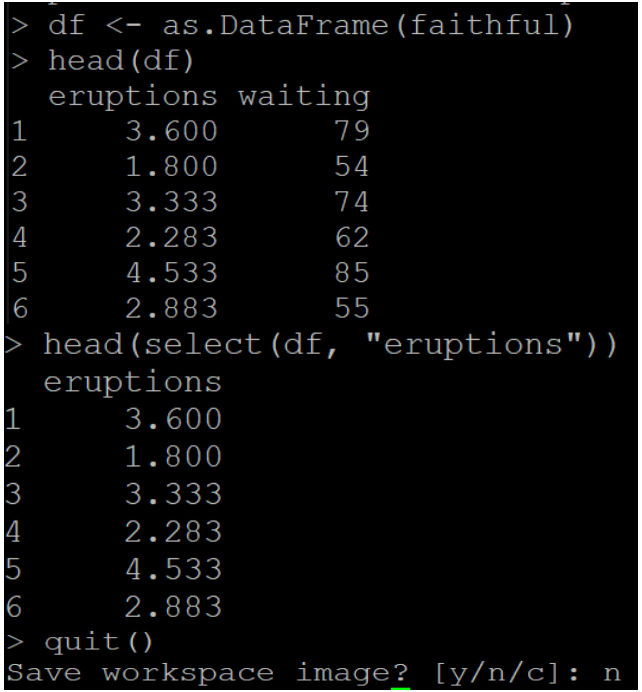


hadmin@MBDUIV0:~

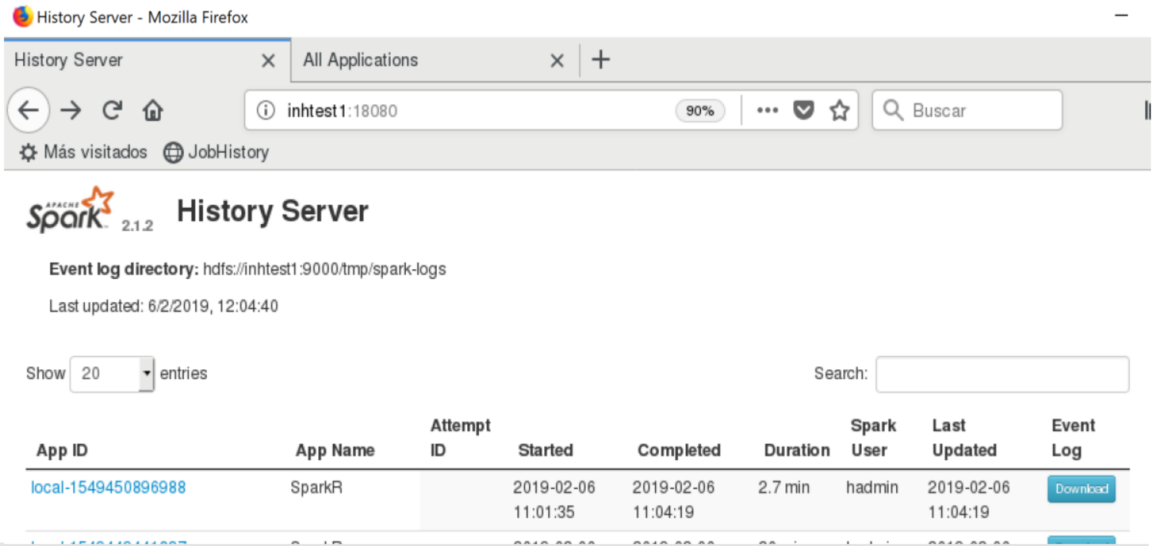
```
[hadmin@MBDUIV0 ~]$ sparkR

R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (64-bit)

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribucion.

R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener más información y
'citation()' para saber cómo citar R o paquetes de R en publicaciones.

Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

Launching java with spark-submit command /usr/local/spark-2.4.0-bin-hadoop2.7/bin/spark-submit   "sparkr-shell" /tmp/RtmpePkSOT/backend_port22961fdd5cfe
2019-01-31 14:18:20 WARN  Utils:66 - Your hostname, MBDUIV0 resolves to a loopback address: 127.0.0.1; using 156.35.119.130 instead (on interface eth2)
2019-01-31 14:18:20 WARN  Utils:66 - Set SPARK_LOCAL_IP if you need to bind to another address
2019-01-31 14:18:21 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

 Welcome to
    ____              __
   / __/__  ___ _____/ /__
  _\ \/ _ \/ _ `/ __/  '_/
 /___/ .__/\_,_/_/ /_/\_\   version  2.4.0
    /_/


 SparkSession available as 'spark'.
>
```

SparkR: Ejecución

- Se puede ejecutar en local o cluster
  - □ sparkR.session(master="local[*]")
  - □ sparkR.session(master="yarn")



```
> df <- as.DataFrame(faithful)
> head(df)
  eruptions waiting
1     3.600      79
2     1.800      54
3     3.333      74
4     2.283      62
5     4.533      85
6     2.883      55
> head(select(df, "eruptions"))
  eruptions
1     3.600
2     1.800
3     3.333
4     2.283
5     4.533
6     2.883
> quit()
Save workspace image? [y/n/c]: n
```



History Server

Event log directory: hdfs://inhtest1:9000/tmp/spark-logs

Last updated: 6/2/2019, 12:04:40

Show 20 entries

Search:

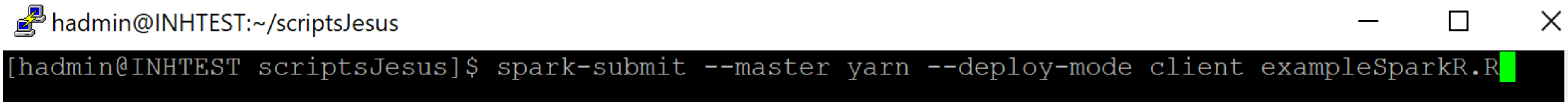
App ID	App Name	Attempt ID	Started	Completed	Duration	Spark User	Last Updated	Event Log
local-1549450896988	SparkR		2019-02-06 11:01:35	2019-02-06 11:04:19	2.7 min	hadmin	2019-02-06 11:04:19	Download

SparkR: Ejecución

- Crear script SparkR: (exampleSparkR.R)

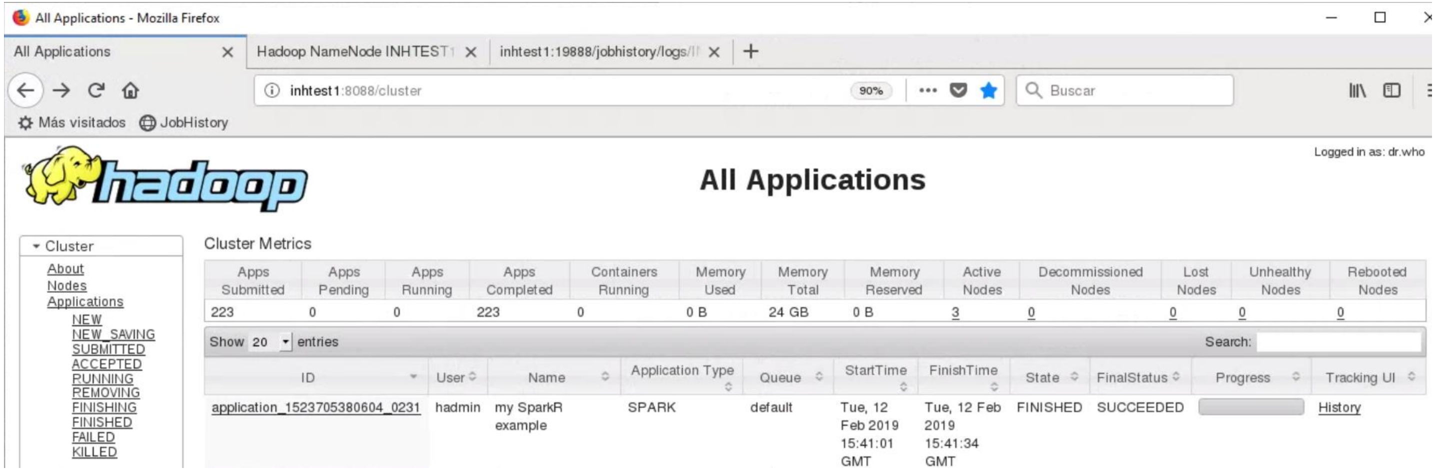
```
sparkR.session(master="yarn", appName="my SparkR  example")
df <- as.DataFrame(faithful)
head(df)
head(select(df, "eruptions"))
```

- Le damos permisos de ejecución: `chmod u+x exampleSparkR.R`
- Ejecución:



```
[hadmin@INHTEST scriptsJesus]$ spark-submit --master yarn --deploy-mode client exampleSparkR.R
```

SparkR: Ejecución



Logged in as: dr.who

All Applications

- Cluster
  - About
  - Nodes
  - Applications
    - NEW
    - NEW_SAVING
    - SUBMITTED
    - ACCEPTED
    - RUNNING
    - REMOVING
    - FINISHING
    - FINISHED
    - FAILED
    - KILLED

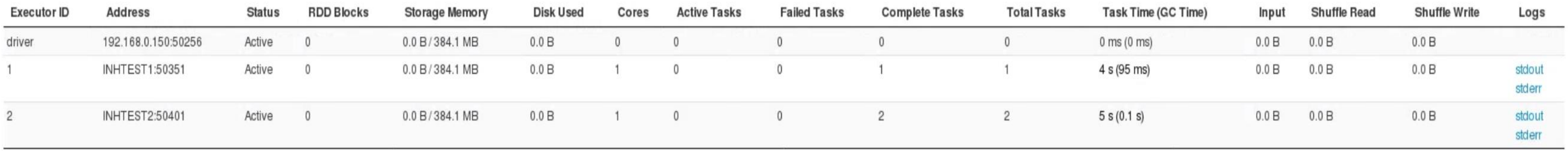
Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
223	0	0	223	0	0 B	24 GB	0 B	3	0	0	0	0

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1523705380604_0231	hadmin	my SparkR example	SPARK	default	Tue, 12 Feb 2019 15:41:01 GMT	Tue, 12 Feb 2019 15:41:34 GMT	FINISHED	SUCCEEDED		History

- Se ejecutó en paralelo en dos servidores



Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
driver	192.168.0.150:50256	Active	0	0.0 B / 384.1 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	
1	INHTEST1:50351	Active	0	0.0 B / 384.1 MB	0.0 B	1	0	0	1	1	4 s (95 ms)	0.0 B	0.0 B	0.0 B	stdout stderr
2	INHTEST2:50401	Active	0	0.0 B / 384.1 MB	0.0 B	1	0	0	2	2	5 s (0.1 s)	0.0 B	0.0 B	0.0 B	stdout stderr

SparkR: Cargar datos

- Crear un DataFrame: `read.df`
  - path: ubicación de los datos
  - header: indicamos si hay cabecera
  - source: tipo de datos, ej csv
  - delimiter: separador de columna
  - inferSchema: indicamos si SparkR tiene que inferir el schema
    - Se puede pasar un schema
  - na.strings: valor que se utiliza cuando haya NA

```
myDF <- read.df(path="/user/hadmin/Iowa_Liquor_Sales4G_cleaned.csv", source =
"csv", delimiter=",", inferSchema = "true", header = "true", na.strings = "")
```

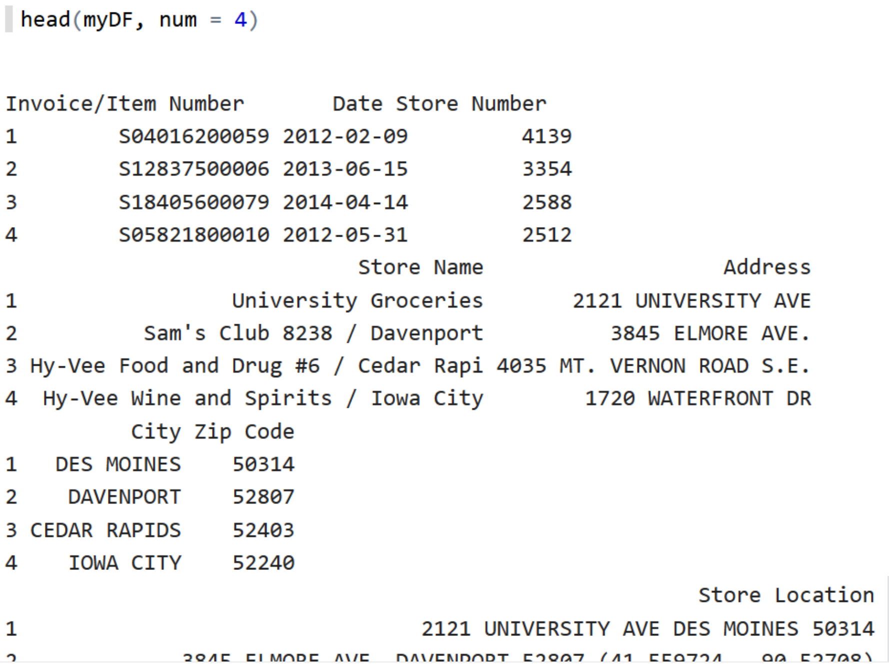
Número de columnas: `colnames`

```
colnames(myDF)
```

```
[1] “Invoice/Item Number”   “Date”
 [3] “Store Number”          “Store Name”
 [5] “Address”               “City”
 [7] “Zip Code”              “Store Location”
 [9] “County Number”         “County”
[11] “Category”              “Category Name”
[13] “Vendor Number”         “Vendor Name”
[15] “Item Number”           “Item Description”
[17] “Pack”                  “Bottle Volume (ml)”
[19] “State Bottle Cost”     “State Bottle Retail”
[21] “Bottles Sold”          “Sale (Dollars)”
[23] “Volume Sold (Liters)”  “Volume Sold (Gallons)”
```

Primeros datos: head

- □ num: número de filas a mostrar

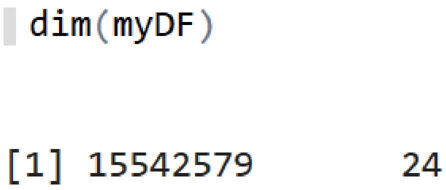


```
head(myDF, num = 4)
```

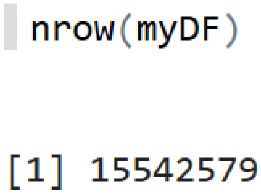
```
Invoice/Item Number       Date Store Number
1        S04016200059 2012-02-09         4139
2        S12837500006 2013-06-15         3354
3        S18405600079 2014-04-14         2588
4        S05821800010 2012-05-31         2512
                            Store Name                   Address
1                 University Groceries       2121 UNIVERSITY AVE
2          Sam's Club 8238 / Davenport          3845 ELMORE AVE.
3 Hy-Vee Food and Drug #6 / Cedar Rapi 4035 MT. VERNON ROAD S.E.
4  Hy-Vee Wine and Spirits / Iowa City        1720 WATERFRONT DR
          City Zip Code
1   DES MOINES    50314
2    DAVENPORT    52807
3 CEDAR RAPIDS    52403
4    IOWA CITY    52240
                                                       Store Location
1                                2121 UNIVERSITY AVE DES MOINES 50314
2            3845 ELMORE AVE.  DAVENPORT 52807 (41.559724, -90.52708)
```

SparkR: Operaciones básicas de resumen

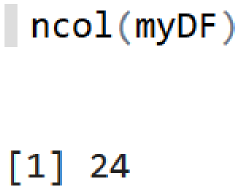
- Dimensiones del dataset: `dim, nrow y ncol`



```
dim(myDF)

[1] 15542579        24
```

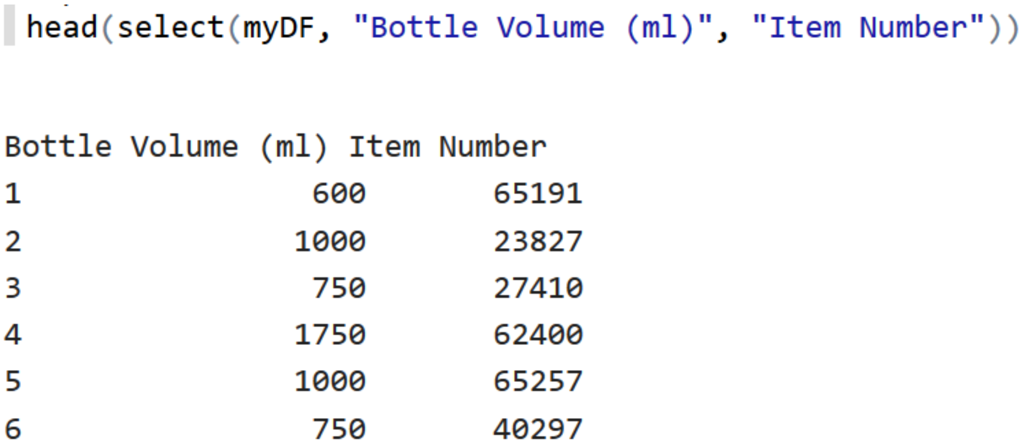


```
nrow(myDF)

[1] 15542579
```



```
ncol(myDF)

[1] 24
```

- Seleccionar columnas: `select`

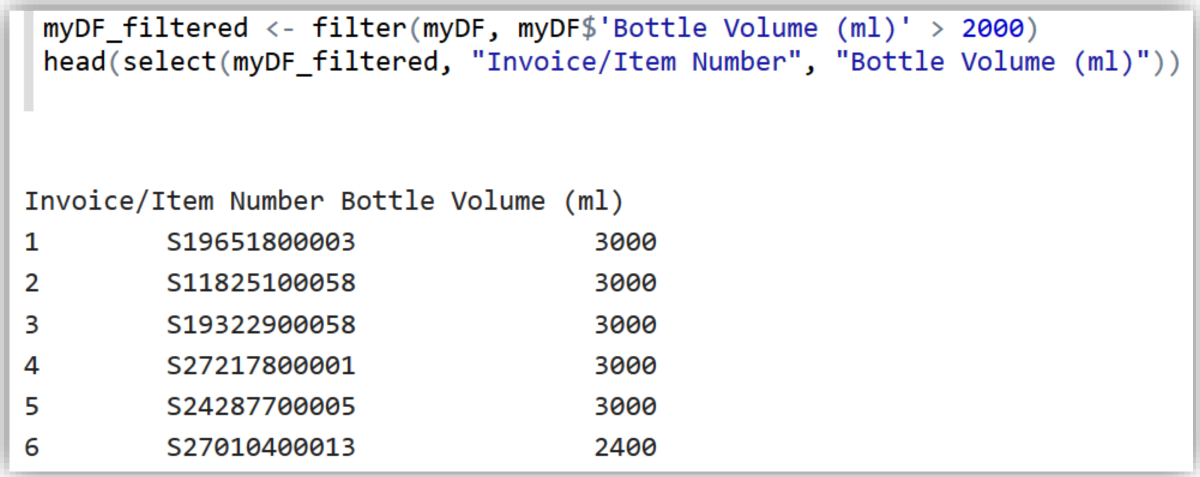


```
head(select(myDF, "Bottle Volume (ml)", "Item Number"))
```

```
Bottle Volume (ml) Item Number
1                600       65191
2               1000       23827
3                750       27410
4               1750       62400
5               1000       65257
6                750       40297
```

SparkR: Operaciones básicas de resumen

- Filtrar: `filter, where`

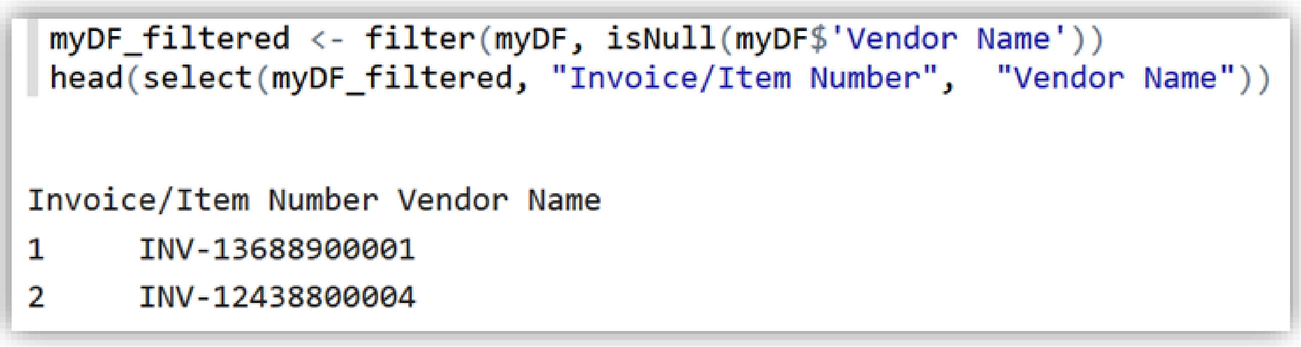


```
myDF_filtered <- filter(myDF, myDF$'Bottle Volume (ml)' > 2000)
head(select(myDF_filtered, "Invoice/Item Number", "Bottle Volume (ml)"))
```

```
Invoice/Item Number Bottle Volume (ml)
1         S19651800003                3000
2         S11825100058                3000
3         S19322900058                3000
4         S27217800001                3000
5         S24287700005                3000
6         S27010400013                2400
```

SparkR: Operaciones básicas de resumen

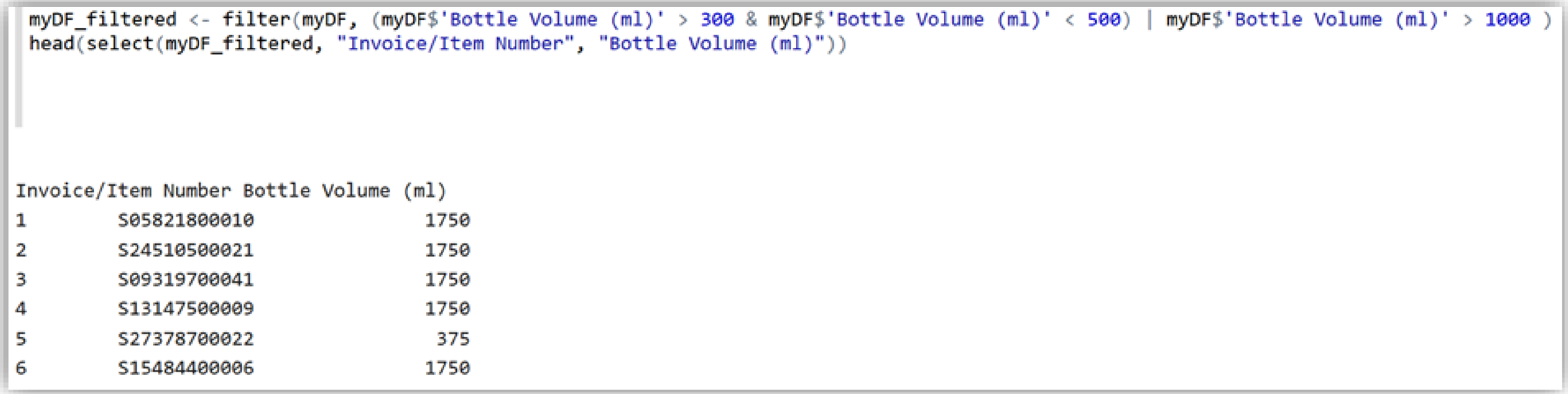
- Filtrar: `filter, where`
  - Comprobar nulls: `isNull`



```
myDF_filtered <- filter(myDF, isNull(myDF$'Vendor Name'))
head(select(myDF_filtered, "Invoice/Item Number",  "Vendor Name"))


Invoice/Item Number Vendor Name
1      INV-13688900001
2      INV-12438800004
```

SparkR: Operaciones básicas de resumen

- Filtrar: `filter, where`
  - Condicionales compuestos: & |

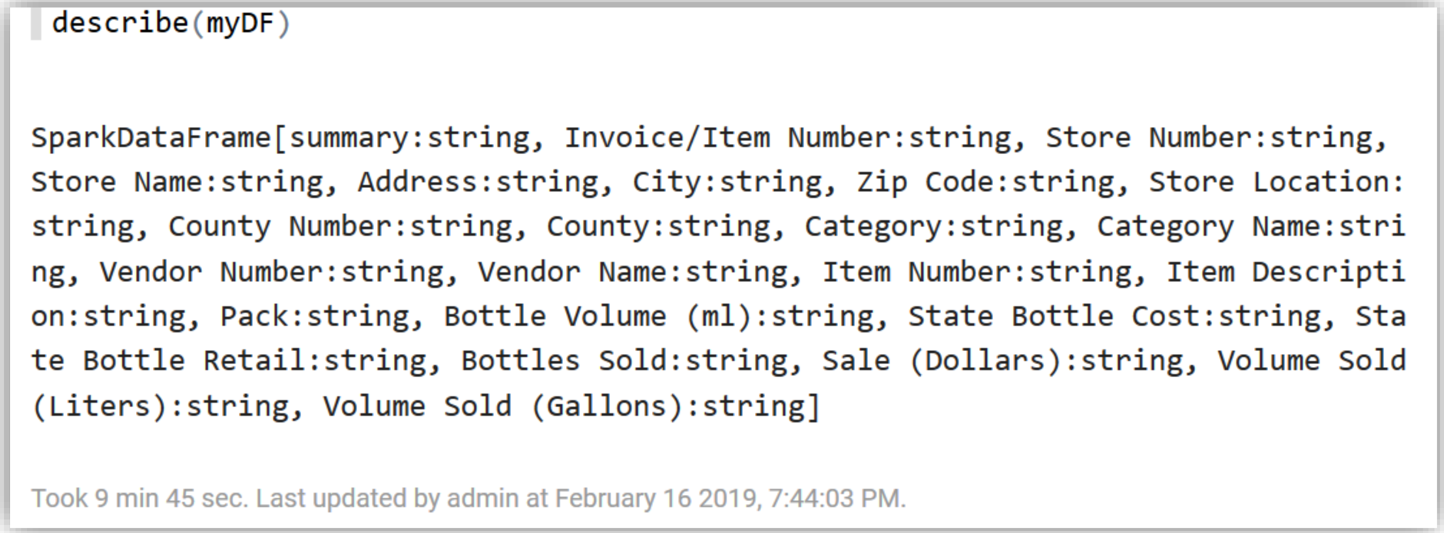


```
mydf_filtered <- filter(myDF, (myDF$'Bottle Volume (ml)' > 300 & myDF$'Bottle Volume (ml)' < 500) | myDF$'Bottle Volume (ml)' > 1000 )
head(select(myDF_filtered, "Invoice/Item Number", "Bottle Volume (ml)"))
```

```
Invoice/Item Number Bottle Volume (ml)
1        S05821800010               1750
2        S24510500021               1750
3        S09319700041               1750
4        S13147500009               1750
5        S27378700022                375
6        S15484400006               1750
```

SparkR: Operaciones básicas de resumen

- Resumen de columnas: `describe, str, printSchema, schema, showDF`



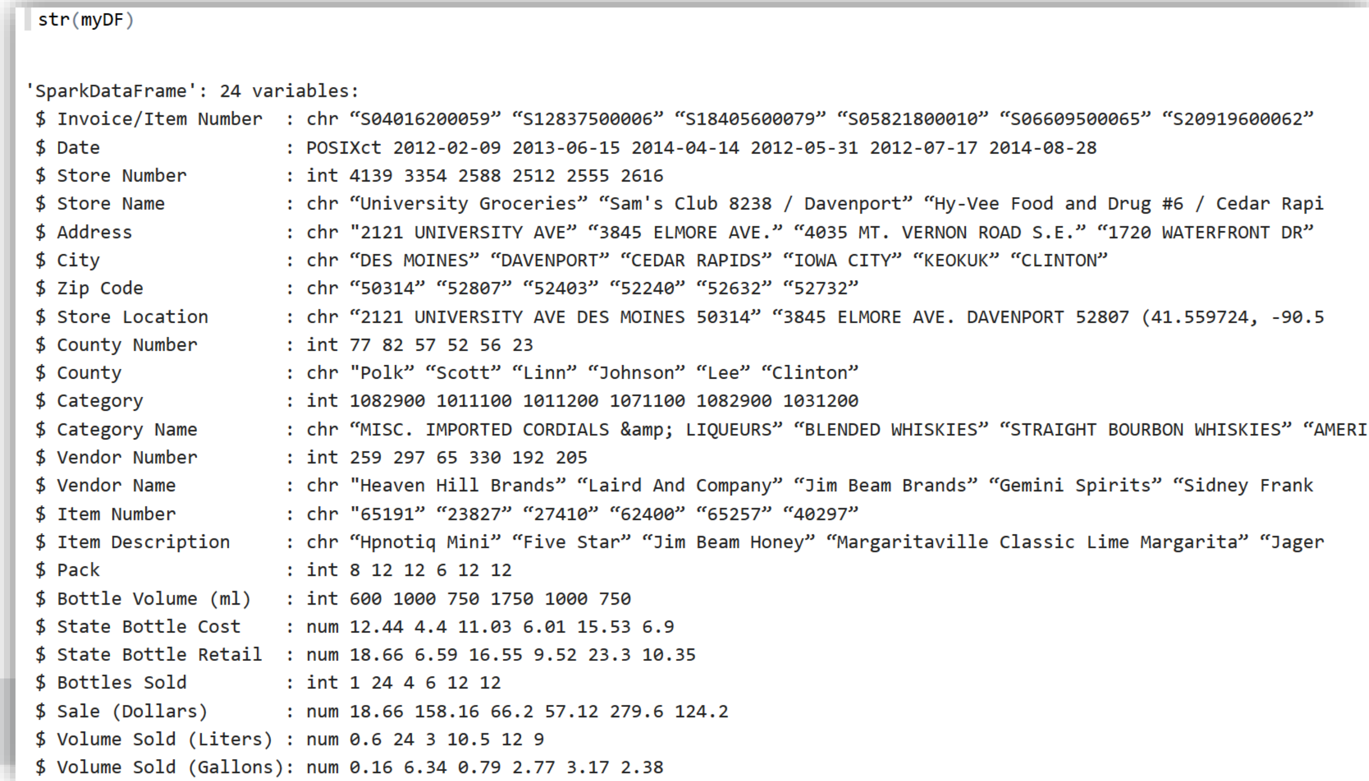
```
describe(myDF)
```

```
SparkDataFrame[summary:string, Invoice/Item Number:string, Store Number:string,
Store Name:string, Address:string, City:string, Zip Code:string, Store Location:
string, County Number:string, County:string, Category:string, Category Name:stri
ng, Vendor Number:string, Vendor Name:string, Item Number:string, Item Descripti
on:string, Pack:string, Bottle Volume (ml):string, State Bottle Cost:string, Sta
te Bottle Retail:string, Bottles Sold:string, Sale (Dollars):string, Volume Sold
(Liters):string, Volume Sold (Gallons):string]
```

Took 9 min 45 sec. Last updated by admin at February 16 2019, 7:44:03 PM.

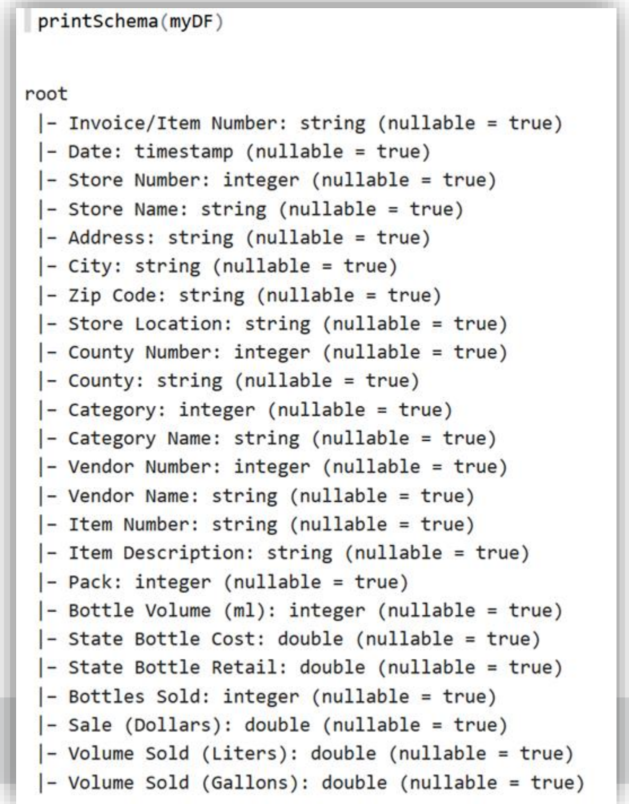
SparkR: Operaciones básicas de resumen

- Resumen de columnas: `describe, str, printSchema, schema, showDF`



```
str(myDF)


'SparkDataFrame': 24 variables:
 $ Invoice/Item Number   : chr “S04016200059” “S12837500006” “S18405600079” “S05821800010” “S06609500065” “S20919600062”
 $ Date                  : POSIXct 2012-02-09 2013-06-15 2014-04-14 2012-05-31 2012-07-17 2014-08-28
 $ Store Number          : int 4139 3354 2588 2512 2555 2616
 $ Store Name            : chr “University Groceries” “Sam's Club 8238 / Davenport” “Hy-Vee Food and Drug #6 / Cedar Rapi
 $ Address               : chr "2121 UNIVERSITY AVE” “3845 ELMORE AVE.” “4035 MT. VERNON ROAD S.E.” “1720 WATERFRONT DR”
 $ City                  : chr “DES MOINES” “DAVENPORT” “CEDAR RAPIDS” “IOWA CITY” “KEOKUK” “CLINTON”
 $ Zip Code              : chr “50314” “52807” “52403” “52240” “52632” “52732”
 $ Store Location        : chr “2121 UNIVERSITY AVE DES MOINES 50314” “3845 ELMORE AVE. DAVENPORT 52807 (41.559724, -90.5
 $ County Number         : int 77 82 57 52 56 23
 $ County                : chr "Polk” “Scott” “Linn” “Johnson” “Lee” “Clinton”
 $ Category              : int 1082900 1011100 1011200 1071100 1082900 1031200
 $ Category Name         : chr “MISC. IMPORTED CORDIALS &amp; LIQUEURS” “BLENDED WHISKIES” “STRAIGHT BOURBON WHISKIES” “AMERI
 $ Vendor Number         : int 259 297 65 330 192 205
 $ Vendor Name           : chr "Heaven Hill Brands” “Laird And Company” “Jim Beam Brands” “Gemini Spirits” “Sidney Frank
 $ Item Number           : chr "65191” “23827” “27410” “62400” “65257” “40297”
 $ Item Description      : chr “Hpnotiq Mini” “Five Star” “Jim Beam Honey” “Margaritaville Classic Lime Margarita” “Jager
 $ Pack                  : int 8 12 12 6 12 12
 $ Bottle Volume (ml)    : int 600 1000 750 1750 1000 750
 $ State Bottle Cost     : num 12.44 4.4 11.03 6.01 15.53 6.9
 $ State Bottle Retail   : num 18.66 6.59 16.55 9.52 23.3 10.35
 $ Bottles Sold          : int 1 24 4 6 12 12
 $ Sale (Dollars)        : num 18.66 158.16 66.2 57.12 279.6 124.2
 $ Volume Sold (Liters)  : num 0.6 24 3 10.5 12 9
 $ Volume Sold (Gallons): num 0.16 6.34 0.79 2.77 3.17 2.38
```

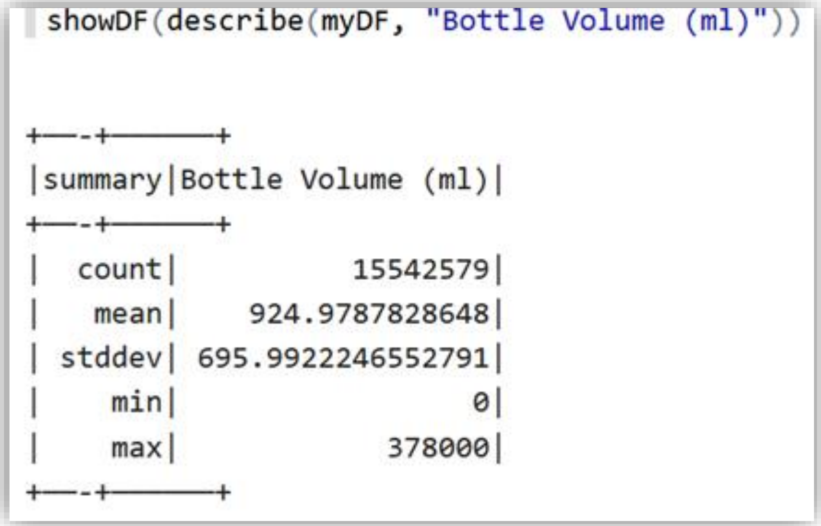
- Resumen de columnas: `describe, str, printSchema, schema, showDF`



```
printSchema(myDF)


root
 |- Invoice/Item Number: string (nullable = true)
 |- Date: timestamp (nullable = true)
 |- Store Number: integer (nullable = true)
 |- Store Name: string (nullable = true)
 |- Address: string (nullable = true)
 |- City: string (nullable = true)
 |- Zip Code: string (nullable = true)
 |- Store Location: string (nullable = true)
 |- County Number: integer (nullable = true)
 |- County: string (nullable = true)
 |- Category: integer (nullable = true)
 |- Category Name: string (nullable = true)
 |- Vendor Number: integer (nullable = true)
 |- Vendor Name: string (nullable = true)
 |- Item Number: string (nullable = true)
 |- Item Description: string (nullable = true)
 |- Pack: integer (nullable = true)
 |- Bottle Volume (ml): integer (nullable = true)
 |- State Bottle Cost: double (nullable = true)
 |- State Bottle Retail: double (nullable = true)
 |- Bottles Sold: integer (nullable = true)
 |- Sale (Dollars): double (nullable = true)
 |- Volume Sold (Liters): double (nullable = true)
 |- Volume Sold (Gallons): double (nullable = true)
```

SparkR: Operaciones básicas de resumen

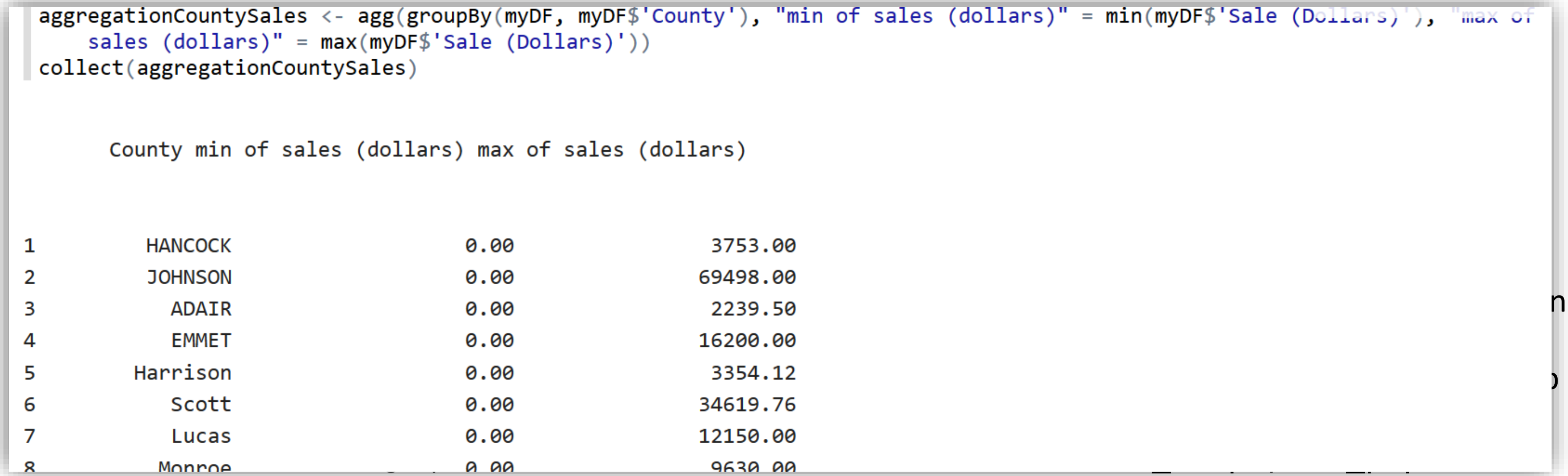
- Resumen de columnas: `describe, str, printSchema, schema, showDF`



```
showDF(describe(myDF, "Bottle Volume (ml)"))


+-------+------------------+
|summary|Bottle Volume (ml)|
+-------+------------------+
|  count|          15542579|
|   mean|    924.9787828648|
| stddev| 695.9922246552791|
|    min|                 0|
|    max|            378000|
+-------+------------------+
```

- Agregación de columnas:
  - □ Agrupación: `groupBy`
  - □ Agregación: `agg`, `sumarize` con:
    - Media: `avg` y `mean`
    - Máximo elemento del grupo: `max`
    - Suma de los elementos del grupo: `sum`
    - Primer elemento del grupo: `first`
    - Último elemento del grupo: `last`
    - Número de elementos: n
    - Número de elementos distintos: `countDistinct` y n_distinct
    - Desviación estándar: `sd`, stddev, stddev_samp y stddev_pop
    - Varianza: `var`, `variance`, `var_samp` y `var_pop`
    - Forma de la distribución: `kurtosis`, `skewness`

Agregación de columnas:



```
aggregationCountySales <- agg(groupBy(myDF, myDF$'County'), "min of sales (dollars)" = min(myDF$'Sale (Dollars)'), "max of
    sales (dollars)" = max(myDF$'Sale (Dollars)'))
collect(aggregationCountySales)
```

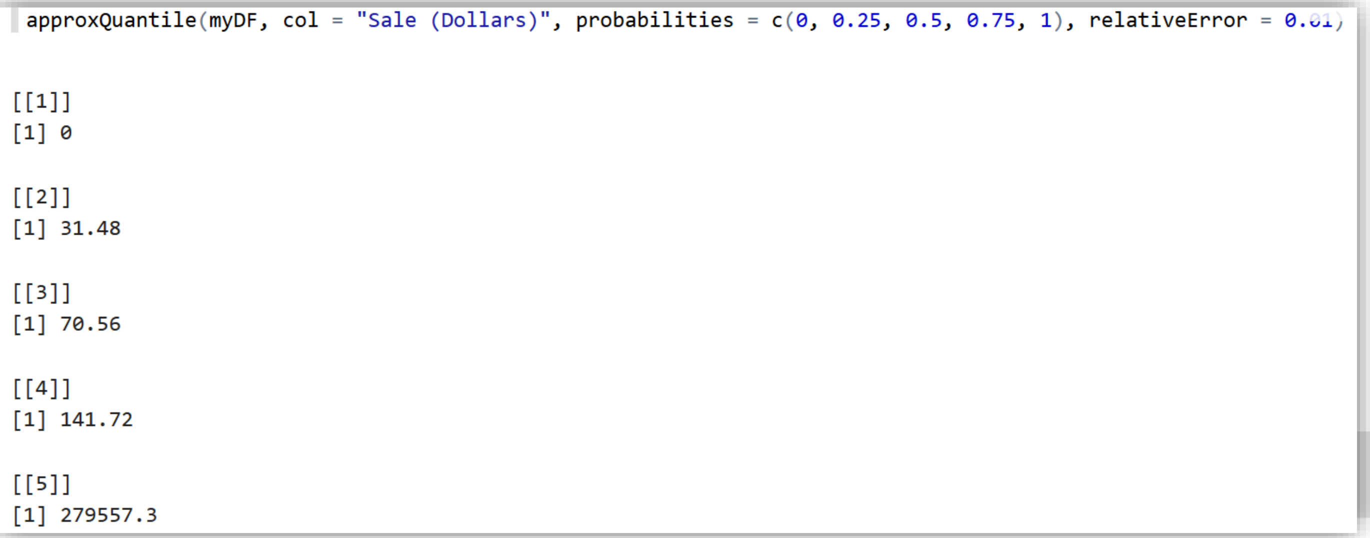
```
     County min of sales (dollars) max of sales (dollars)


1        HANCOCK                 0.00               3753.00
2        JOHNSON                 0.00              69498.00
3          ADAIR                 0.00               2239.50
4          EMMET                 0.00              16200.00
5       Harrison                 0.00               3354.12
6          Scott                 0.00              34619.76
7          Lucas                 0.00              12150.00
8         Monroe                 0.00               9630.00
```

nct

)

- Último elemento del grupo: `last`
- Forma de la distribución: `kurtosis`, `skewness`

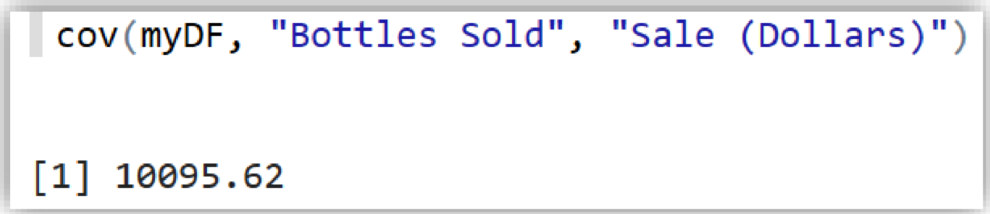
- Resumen estadístico: `approxQuantile`
  - □ Nos da una aproximación que puede tener un error
  - □ `relativeError`



```
approxQuantile(myDF, col = "Sale (Dollars)", probabilities = c(0, 0.25, 0.5, 0.75, 1), relativeError = 0.01)


[[1]]
[1] 0

[[2]]
[1] 31.48

[[3]]
[1] 70.56

[[4]]
[1] 141.72

[[5]]
[1] 279557.3
```

SparkR: Operaciones básicas

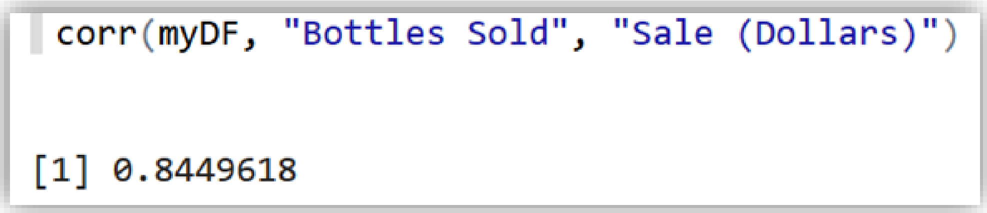
- Resumen estadístico varias columnas:
  - □ Covarianzas: cov, covar_samp y covar_pop
  - □ Correlaciones: corr
  - □ Tablas cruzadas: crosstab



```
cov(myDF, "Bottles Sold", "Sale (Dollars)")


[1] 10095.62
```

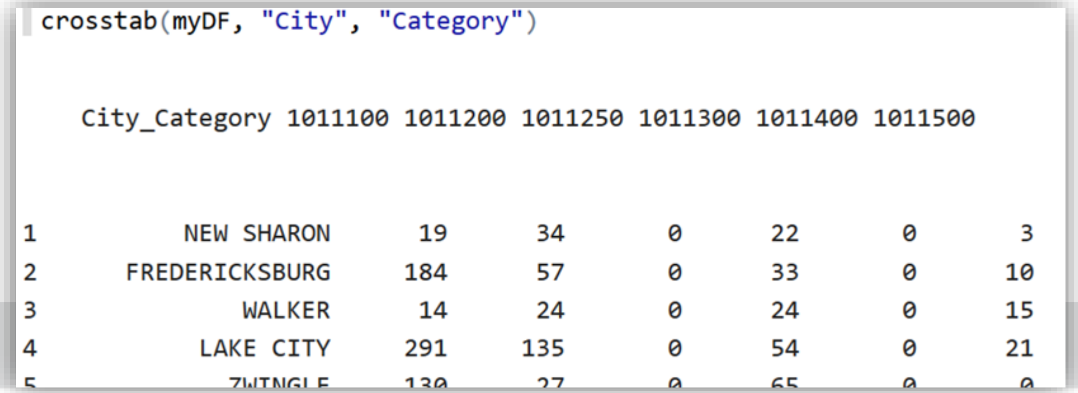
SparkR: Operaciones básicas

- Resumen estadístico varias columnas:
  - □ Covarianzas: `cov`, `covar_samp` y `covar_pop`
  - □ Correlaciones: corr
  - □ Tablas cruzadas: `crosstab`



```
corr(myDF, "Bottles Sold", "Sale (Dollars)")


[1] 0.8449618
```

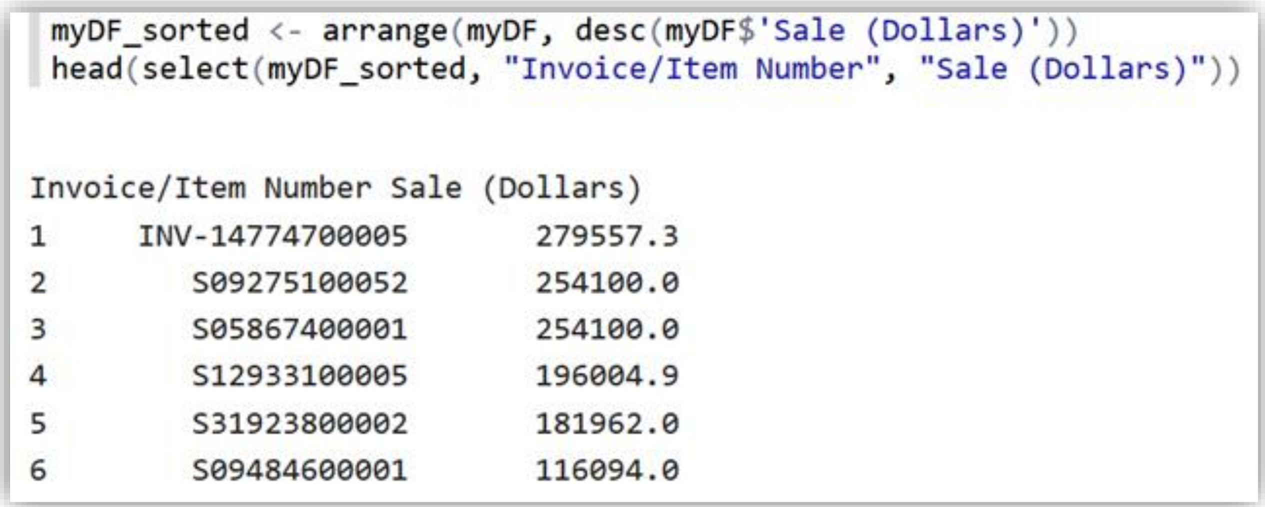
- Resumen estadístico varias columnas:
  - □ Covarianzas: `cov`, `covar_samp` y `covar_pop`
  - □ Correlaciones: corr
  - □ Tablas cruzadas: `crosstab`



```
crosstab(myDF, "City", "Category")

   City_Category 1011100 1011200 1011250 1011300 1011400 1011500

1        NEW SHARON      19      34       0      22       0       3
2    FREDERICKSBURG     184      57       0      33       0      10
3            WALKER      14      24       0      24       0      15
4         LAKE CITY     291     135       0      54       0      21
5          ZWINGLE     130      27       0      65       0       0
```

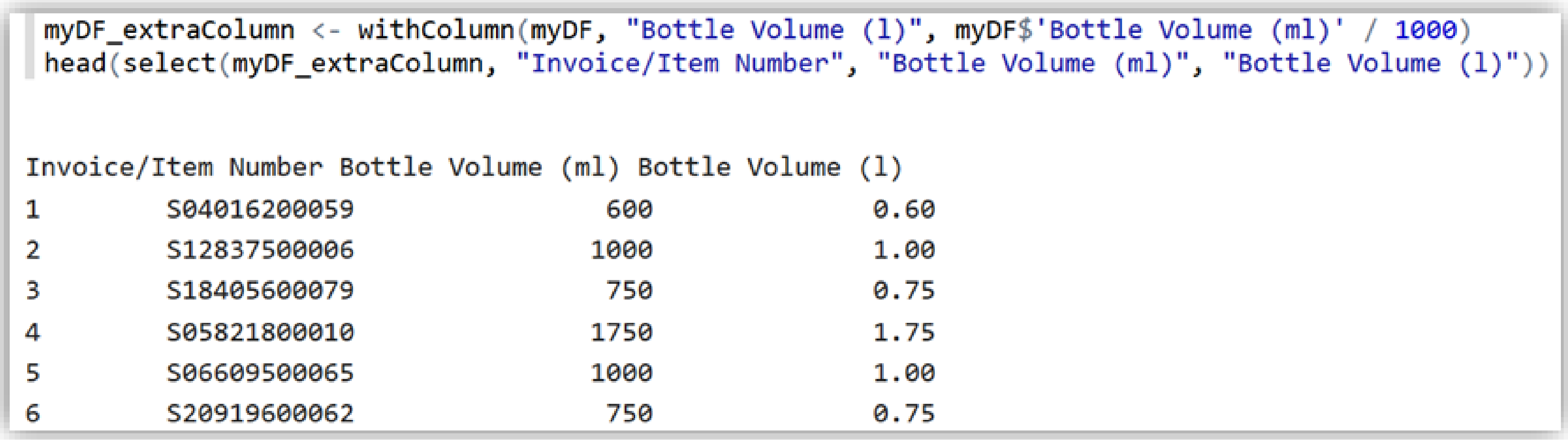
Ordenar columna: arrange, orderBy

- Le indicamos asc o desc



```
myDF_sorted <- arrange(myDF, desc(myDF$'Sale (Dollars)'))
head(select(myDF_sorted, "Invoice/Item Number", "Sale (Dollars)"))


Invoice/Item Number Sale (Dollars)
1      INV-14774700005       279557.3
2         S09275100052       254100.0
3         S05867400001       254100.0
4         S12933100005       196004.9
5         S31923800002       181962.0
6         S09484600001       116094.0
```

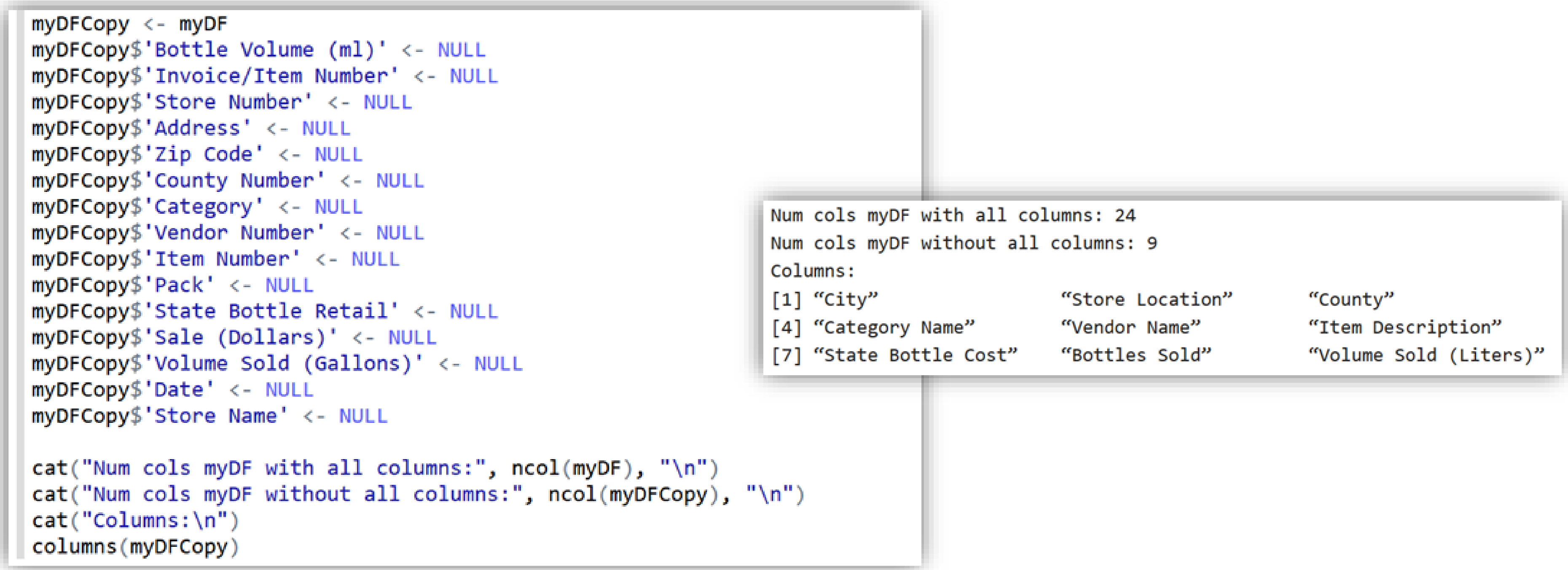
Añadir columna: `withColumn`



```
myDF_extraColumn <- withColumn(myDF, "Bottle Volume (l)", myDF$'Bottle Volume (ml)' / 1000)
head(select(myDF_extraColumn, "Invoice/Item Number", "Bottle Volume (ml)", "Bottle Volume (l)"))
```

```
Invoice/Item Number Bottle Volume (ml) Bottle Volume (l)
1        S04016200059                600              0.60
2        S12837500006               1000              1.00
3        S18405600079                750              0.75
4        S05821800010               1750              1.75
5        S06609500065               1000              1.00
6        S20919600062                750              0.75
```

Eliminar columna: NULL

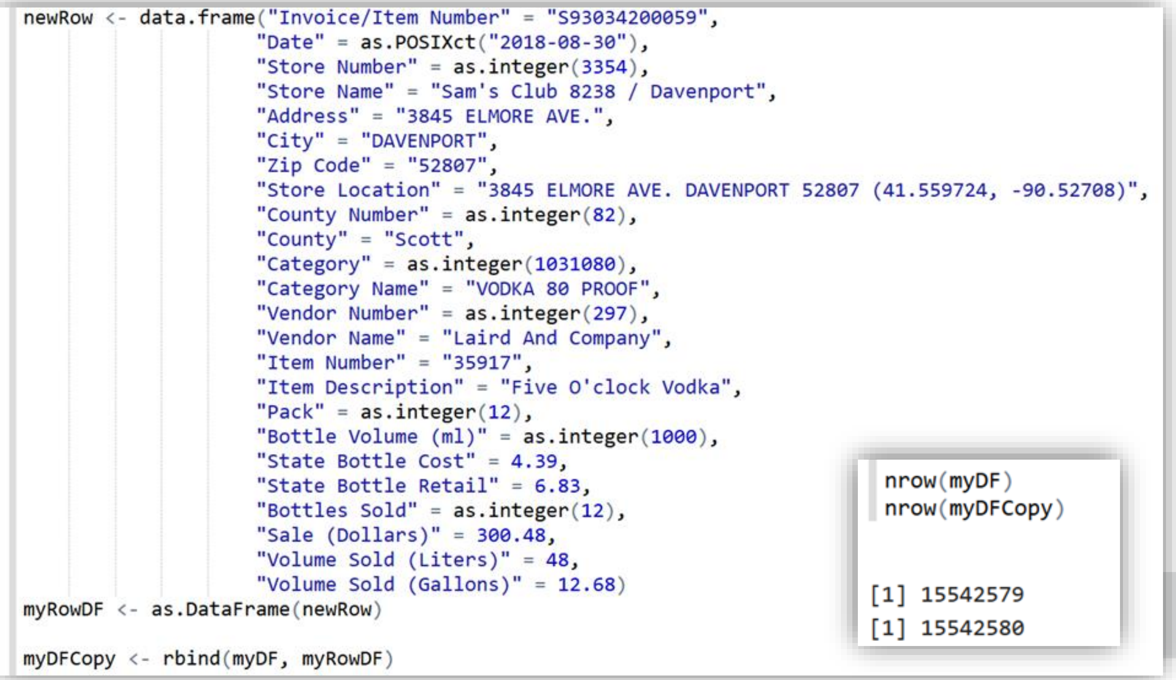


```r
myDFCopy <- myDF
myDFCopy$'Bottle Volume (ml)' <- NULL
myDFCopy$'Invoice/Item Number' <- NULL
myDFCopy$'Store Number' <- NULL
myDFCopy$'Address' <- NULL
myDFCopy$'Zip Code' <- NULL
myDFCopy$'County Number' <- NULL
myDFCopy$'Category' <- NULL
myDFCopy$'Vendor Number' <- NULL
myDFCopy$'Item Number' <- NULL
myDFCopy$'Pack' <- NULL
myDFCopy$'State Bottle Retail' <- NULL
myDFCopy$'Sale (Dollars)' <- NULL
myDFCopy$'Volume Sold (Gallons)' <- NULL
myDFCopy$'Date' <- NULL
myDFCopy$'Store Name' <- NULL

cat("Num cols myDF with all columns:", ncol(myDF), "\n")
cat("Num cols myDF without all columns:", ncol(myDFCopy), "\n")
cat("Columns:\n")
columns(myDFCopy)
```

```
Num cols myDF with all columns: 24
Num cols myDF without all columns: 9
Columns:
[1] “City”               “Store Location”      “County”
[4] “Category Name”      “Vendor Name”         “Item Description”
[7] “State Bottle Cost”  “Bottles Sold”        “Volume Sold (Liters)”
```

Añadir filas:

- □ Unión de dos DataFrames
- □ rbind

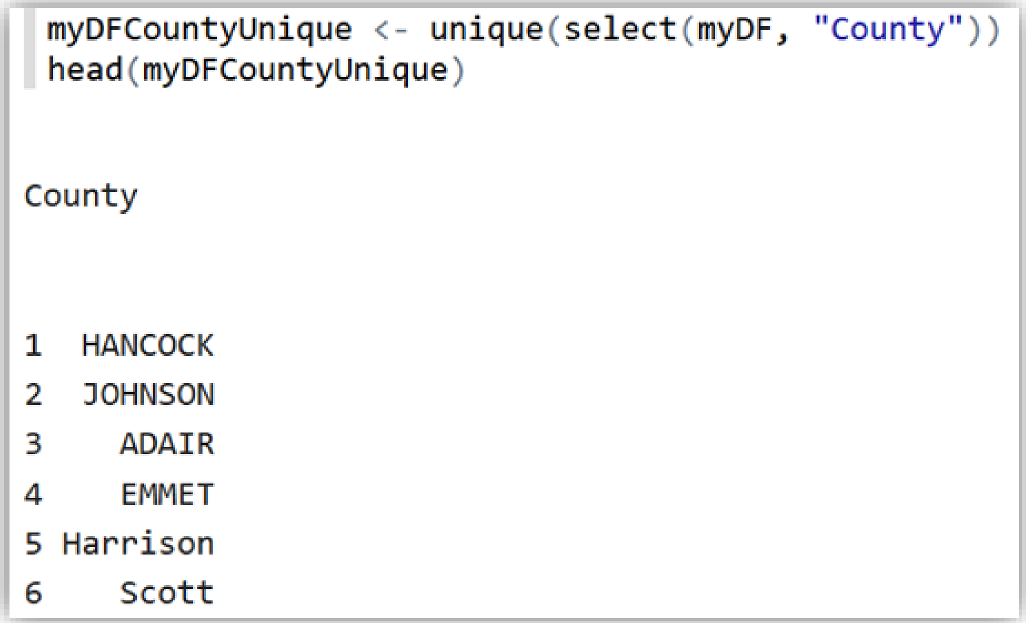


```
newRow <- data.frame("Invoice/Item Number" = "S93034200059",
                     "Date" = as.POSIXct("2018-08-30"),
                     "Store Number" = as.integer(3354),
                     "Store Name" = "Sam's Club 8238 / Davenport",
                     "Address" = "3845 ELMORE AVE.",
                     "City" = "DAVENPORT",
                     "Zip Code" = "52807",
                     "Store Location" = "3845 ELMORE AVE. DAVENPORT 52807 (41.559724, -90.52708)",
                     "County Number" = as.integer(82),
                     "County" = "Scott",
                     "Category" = as.integer(1031080),
                     "Category Name" = "VODKA 80 PROOF",
                     "Vendor Number" = as.integer(297),
                     "Vendor Name" = "Laird And Company",
                     "Item Number" = "35917",
                     "Item Description" = "Five O'clock Vodka",
                     "Pack" = as.integer(12),
                     "Bottle Volume (ml)" = as.integer(1000),
                     "State Bottle Cost" = 4.39,
                     "State Bottle Retail" = 6.83,
                     "Bottles Sold" = as.integer(12),
                     "Sale (Dollars)" = 300.48,
                     "Volume Sold (Liters)" = 48,
                     "Volume Sold (Gallons)" = 12.68)
myRowDF <- as.DataFrame(newRow)

myDFCopy <- rbind(myDF, myRowDF)
```

```
nrow(myDF)
nrow(myDFCopy)


[1] 15542579
[1] 15542580
```

SparkR: Operaciones básicas

- Obtener datos únicos (conjunto de datos): unique

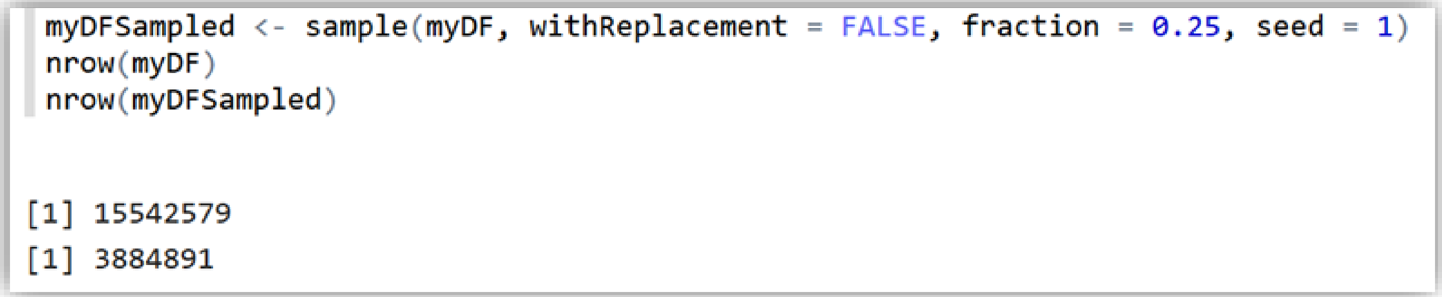


```
myDFCountyUnique <- unique(select(myDF, "County"))
head(myDFCountyUnique)
```

```
County



1   HANCOCK
2   JOHNSON
3     ADAIR
4     EMMET
5  Harrison
6     Scott
```

SparkR: Operaciones básicas

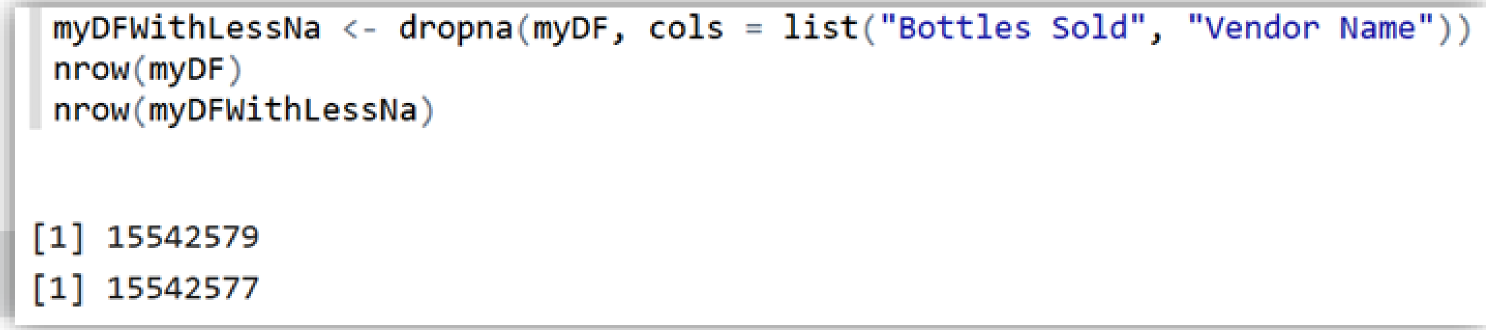
- Muestreo:
  - □ Big Data: `sample`
  - □ Datos pequeños: `takeSample`



```
myDFSampled <- sample(myDF, withReplacement = FALSE, fraction = 0.25, seed = 1)
nrow(myDF)
nrow(myDFSampled)


[1] 15542579
[1] 3884891
```

SparkR: Operaciones básicas

- Eliminar valores nulos: dropNa
  - □ cols: columnas en las que analizar Nulls
  - □ how: condiciones para eliminar la fila:
    - all: se elimina la fila si todas sus cols tienen Nulls
    - any: se elimina la fila si alguna de sus cols tiene NULL
  - □ minNonNulls: Número mínimo de valores no nulos posibles en la fila

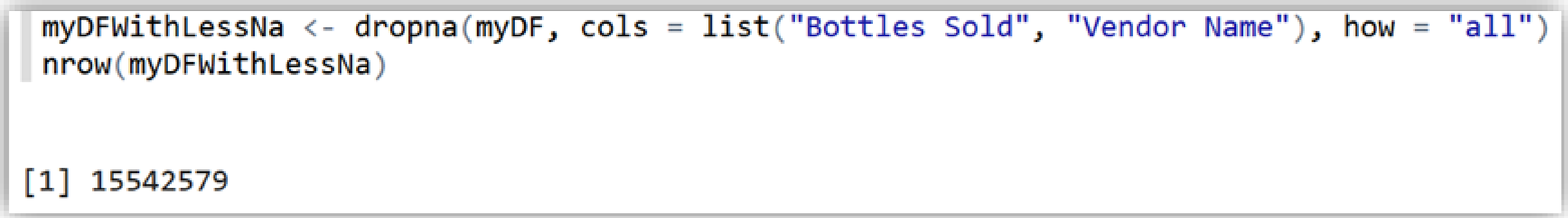


```
myDFWithLessNa <- dropna(myDF, cols = list("Bottles Sold", "Vendor Name"))
nrow(myDF)
nrow(myDFWithLessNa)
```

```
[1] 15542579
[1] 15542577
```

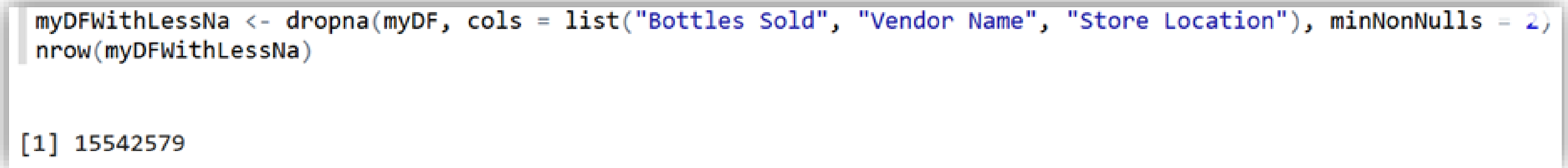
SparkR: Operaciones básicas

- Eliminar valores nulos: dropNa
  - □ cols: columnas en las que analizar Nulls
  - □ how: condiciones para eliminar la fila:
    - all: se elimina la fila si todas sus cols tienen Nulls
    - any: se elimina la fila si alguna de sus cols tiene NULL
  - □ minNonNulls: Número mínimo de valores no nulos posibles en la fila



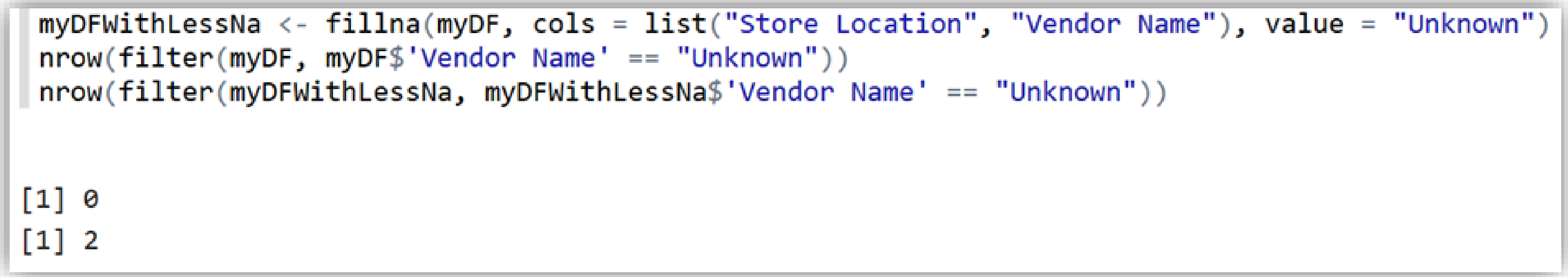
```
myDFWithLessNa <- dropna(myDF, cols = list("Bottles Sold", "Vendor Name"), how = "all")
nrow(myDFWithLessNa)


[1] 15542579
```

- Eliminar valores nulos: dropNa
  - □ cols: columnas en las que analizar Nulls
  - □ how: condiciones para eliminar la fila:
    - all: se elimina la fila si todas sus cols tienen Nulls
    - any: se elimina la fila si alguna de sus cols tiene NULL
  - □ minNonNulls: Número mínimo de valores no nulos posibles en la fila



```
myDFWithLessNa <- dropna(myDF, cols = list("Bottles Sold", "Vendor Name", "Store Location"), minNonNulls = 2)
nrow(myDFWithLessNa)


[1] 15542579
```

- Filtrar/cambiar valores nulos: `fillNA`
  - □ `cols`: columnas a analizar
  - □ `value`: valor por el que se sustituirá el `NULL`

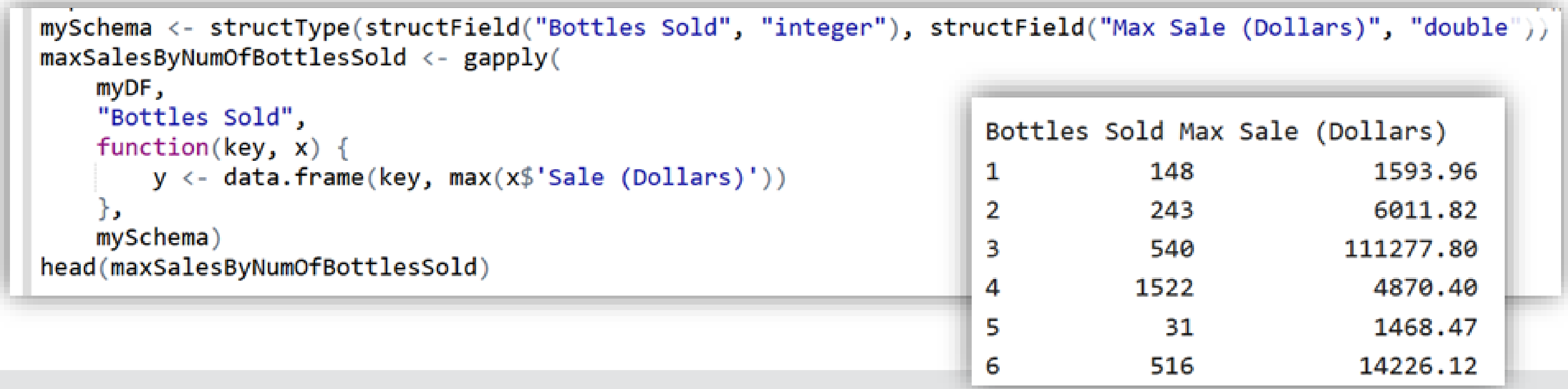


```
myDFWithLessNa <- fillna(myDF, cols = list("Store Location", "Vendor Name"), value = "Unknown")
nrow(filter(myDF, myDF$'Vendor Name' == "Unknown"))
nrow(filter(myDFWithLessNa, myDFWithLessNa$'Vendor Name' == "Unknown"))


[1] 0
[1] 2
```

- `dapply`
  - □ Aplica una función a cada partición
    - Recibe la partición como un data.frame de R
  - □ ~ mapPartitions
  - □ Genera un data.frame de R que se transforma en la partición
  - □ Operación costosa: serializa y deserializa información entre JVM y R

gapply

- Función ejecutada por cada agrupación
  - Columna para agrupar
  - Función para ejecutar en cada grupo



```
mySchema <- structType(structField("Bottles Sold", "integer"), structField("Max Sale (Dollars)", "double"))
maxSalesByNumOfBottlesSold <- gapply(
    myDF,
    "Bottles Sold",
    function(key, x) {
        y <- data.frame(key, max(x$'Sale (Dollars)'))
    },
    mySchema)
head(maxSalesByNumOfBottlesSold)
```

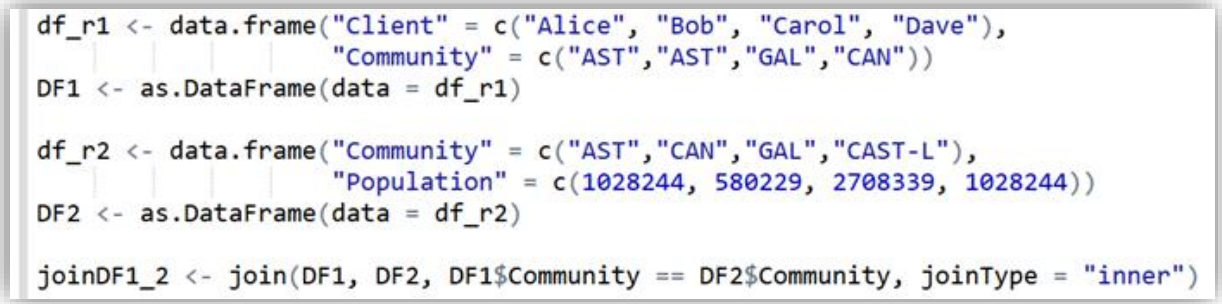
```
Bottles Sold Max Sale (Dollars)
1          148            1593.96
2          243            6011.82
3          540          111277.80
4         1522            4870.40
5           31            1468.47
6          516           14226.12
```

SparkR: Funciones definidas por el usuario

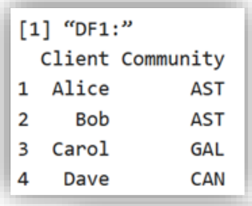
- `lapply`
  - Ejecutar varias tareas en paralelo
  - Tiene que ser capaz de ejecutar cada tarea en un único servidor
    - Ejemplo: entrenar varios modelos con diferentes parámetros
      - Crear lista con los parámetros del modelo
      - lapply entrena en un servidor un modelo, en otro servidor otro modelo, ...
      - Tiene que ser capaz de entrenar cada modelo en un único servidor

SparkR: operaciones con varios DataFrames

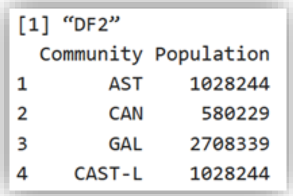
- Unión de columnas: `join`
  - □ DataFrames a unir
  - □ `joinExpr`: Columna que queremos unir
  - □ `joinType`: inner, outer, full, fullouter, leftouter, left_outer, left, right_outer, rightouter, right, y leftsemi



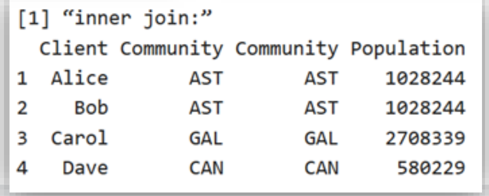
```
df_r1 <- data.frame("Client" = c("Alice", "Bob", "Carol", "Dave"),
                    "Community" = c("AST","AST","GAL","CAN"))
DF1 <- as.DataFrame(data = df_r1)

df_r2 <- data.frame("Community" = c("AST","CAN","GAL","CAST-L"),
                    "Population" = c(1028244, 580229, 2708339, 1028244))
DF2 <- as.DataFrame(data = df_r2)

joinDF1_2 <- join(DF1, DF2, DF1$Community == DF2$Community, joinType = "inner")
```



```
[1] “DF1:”
  Client Community
1  Alice       AST
2    Bob       AST
3  Carol       GAL
4   Dave       CAN
```



```
[1] “DF2”
  Community Population
1       AST    1028244
2       CAN     580229
3       GAL    2708339
4    CAST-L    1028244
```

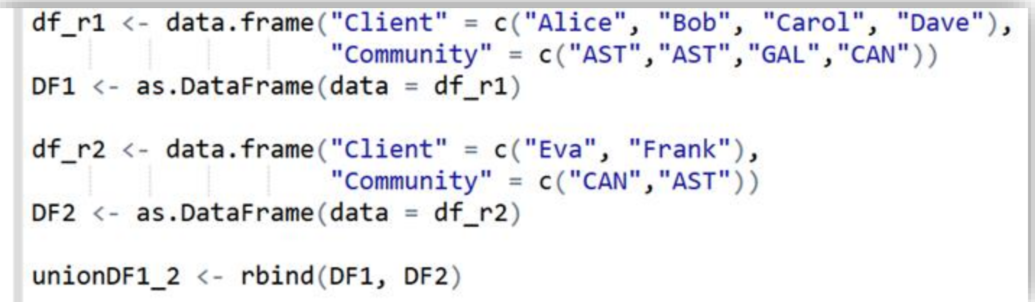


```
[1] “inner join:”
  Client Community Community Population
1  Alice       AST       AST    1028244
2    Bob       AST       AST    1028244
3  Carol       GAL       GAL    2708339
4   Dave       CAN       CAN     580229
```

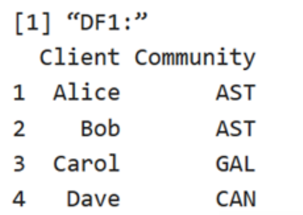
SparkR: operaciones con varios DataFrames

Unión de filas: `rbind`

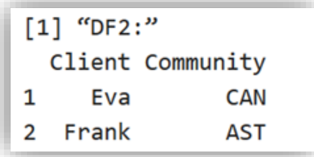
- Añade filas de un DataFrame a otro



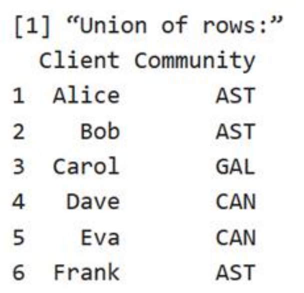
```
df_r1 <- data.frame("Client" = c("Alice", "Bob", "Carol", "Dave"),
                    "Community" = c("AST","AST","GAL","CAN"))
DF1 <- as.DataFrame(data = df_r1)

df_r2 <- data.frame("Client" = c("Eva", "Frank"),
                    "Community" = c("CAN","AST"))
DF2 <- as.DataFrame(data = df_r2)

unionDF1_2 <- rbind(DF1, DF2)
```



```
[1] “DF1:”
  Client Community
1  Alice       AST
2    Bob       AST
3  Carol       GAL
4   Dave       CAN
```

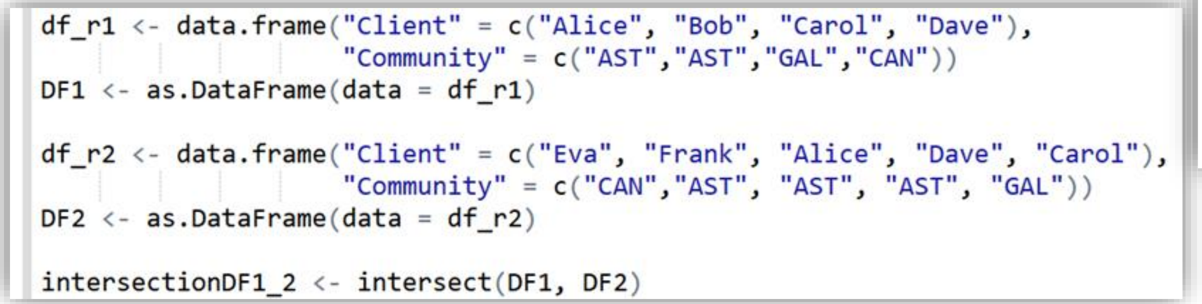


```
[1] “DF2:”
  Client Community
1    Eva       CAN
2  Frank       AST
```

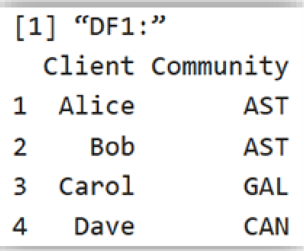


```
[1] “Union of rows:”
  Client Community
1  Alice       AST
2    Bob       AST
3  Carol       GAL
4   Dave       CAN
5    Eva       CAN
6  Frank       AST
```

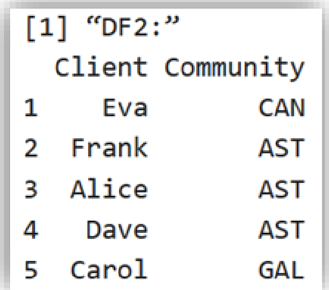
- Intersección: `intersect`
  - □ Obtenemos las filas que están en ambos DataFrames



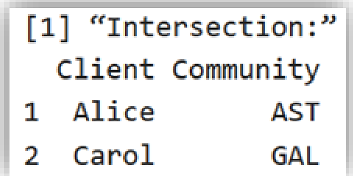
```r
df_r1 <- data.frame("Client" = c("Alice", "Bob", "Carol", "Dave"),
                    "Community" = c("AST","AST","GAL","CAN"))
DF1 <- as.DataFrame(data = df_r1)

df_r2 <- data.frame("Client" = c("Eva", "Frank", "Alice", "Dave", "Carol"),
                    "Community" = c("CAN","AST", "AST", "AST", "GAL"))
DF2 <- as.DataFrame(data = df_r2)

intersectionDF1_2 <- intersect(DF1, DF2)
```



```
[1] “DF1:”
  Client Community
1  Alice       AST
2    Bob       AST
3  Carol       GAL
4   Dave       CAN
```



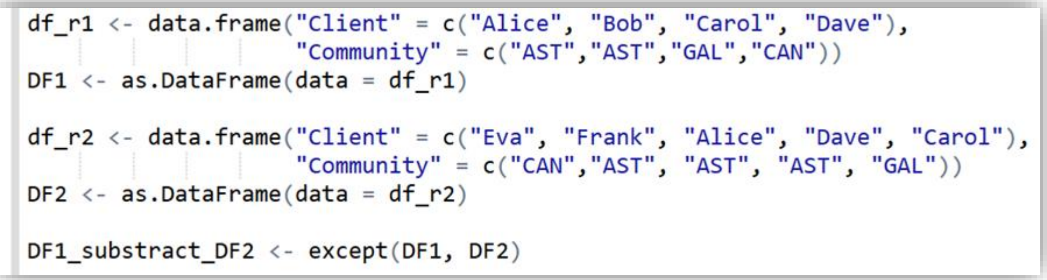
```
[1] “DF2:”
  Client Community
1    Eva       CAN
2  Frank       AST
3  Alice       AST
4   Dave       AST
5  Carol       GAL
```



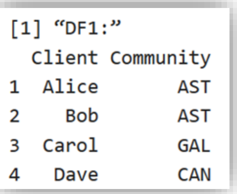
```
[1] “Intersection:”
  Client Community
1  Alice       AST
2  Carol       GAL
```

SparkR: operaciones con varios DataFrames

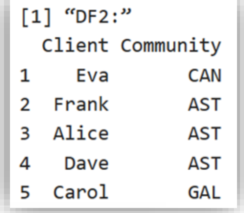
- Resta de filas: `except`
  - □ Quitar las filas que estén en otro DataFrame
  - □ Puede utilizarse para dividir el datasaet en train y test
    - Train: con un muestreo
    - Test: resta del DataFrame original menos el train
  - □ Para dividir un DataFrame en train y test es mejor utilizar `randomSplit` (a partir de la versión 2.0 de Spark)



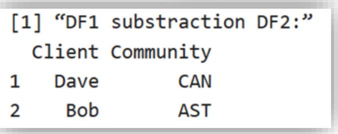
```
df_r1 <- data.frame("Client" = c("Alice", "Bob", "Carol", "Dave"),
                    "Community" = c("AST","AST","GAL","CAN"))
DF1 <- as.DataFrame(data = df_r1)

df_r2 <- data.frame("Client" = c("Eva", "Frank", "Alice", "Dave", "Carol"),
                    "Community" = c("CAN","AST", "AST", "AST", "GAL"))
DF2 <- as.DataFrame(data = df_r2)

DF1_substract_DF2 <- except(DF1, DF2)
```



```
[1] “DF1:”
  Client Community
1  Alice       AST
2    Bob       AST
3  Carol       GAL
4   Dave       CAN
```



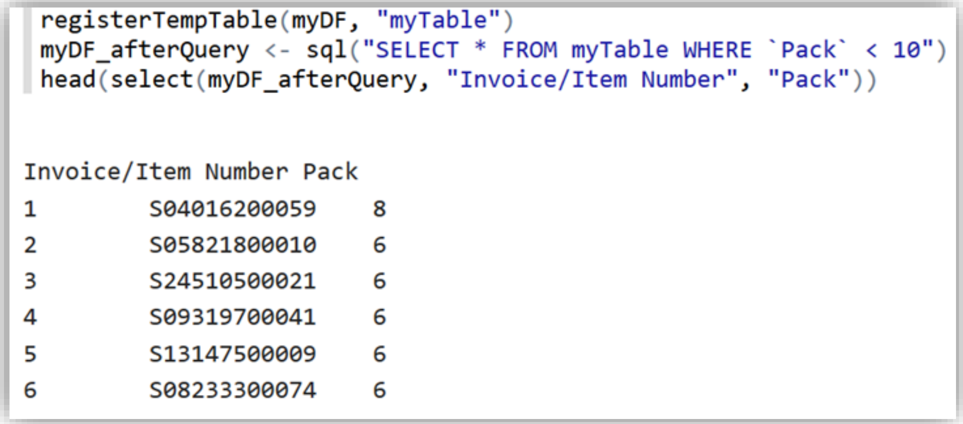
```
[1] “DF2:”
  Client Community
1    Eva       CAN
2  Frank       AST
3  Alice       AST
4   Dave       AST
5  Carol       GAL
```



```
[1] “DF1 substraction DF2:”
  Client Community
1   Dave       CAN
2    Bob       AST
```

SQL

- Crear tabla temporal: `registerTempTable`
- Consulta: `sql`

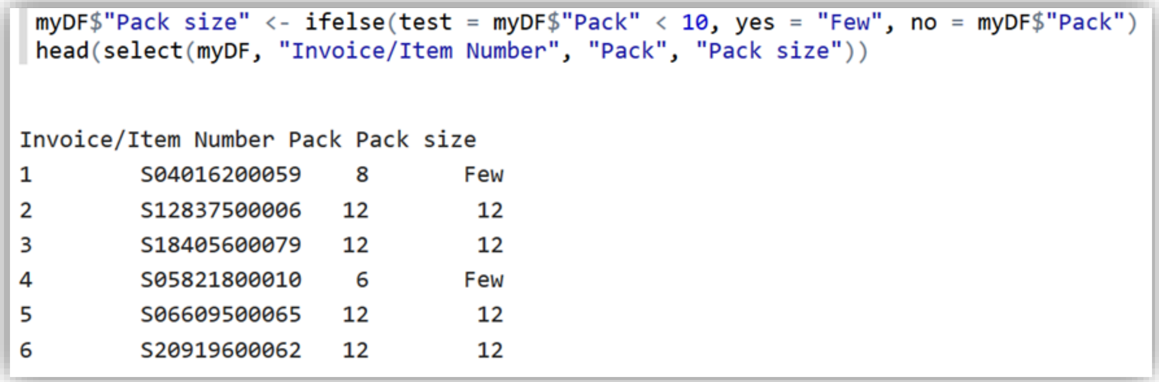


```
registerTempTable(myDF, "myTable")
myDF_afterQuery <- sql("SELECT * FROM myTable WHERE `Pack` < 10")
head(select(myDF_afterQuery, "Invoice/Item Number", "Pack"))
```

```
Invoice/Item Number Pack
1        S04016200059    8
2        S05821800010    6
3        S24510500021    6
4        S09319700041    6
5        S13147500009    6
6        S08233300074    6
```

Sustituir datos: `ifelse`

- Crear una columna con un valor u otro dependiendo de una condición
  - Condición que se evalúa por cada dato
  - Yes: valor por el que se debe sustituir en caso de que el dato sea sí
  - No: idem para el no



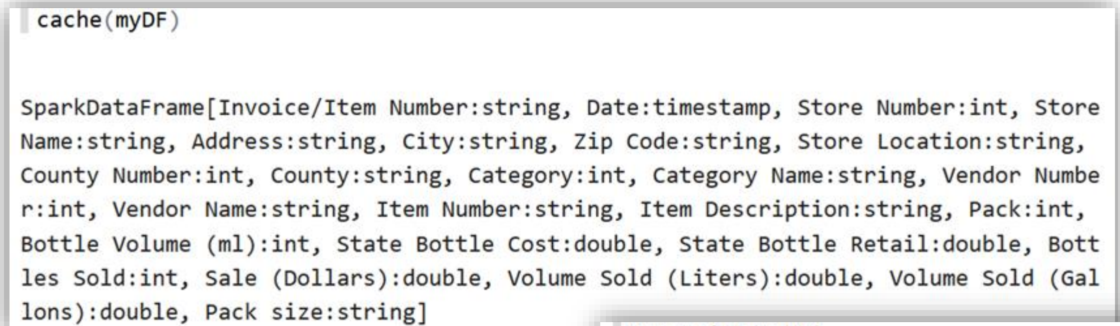
```
myDF$"Pack size" <- ifelse(test = myDF$"Pack" < 10, yes = "Few", no = myDF$"Pack")
head(select(myDF, "Invoice/Item Number", "Pack", "Pack size"))
```

```
Invoice/Item Number Pack Pack size
1         S04016200059    8       Few
2         S12837500006   12        12
3         S18405600079   12        12
4         S05821800010    6       Few
5         S06609500065   12        12
6         S20919600062   12        12
```

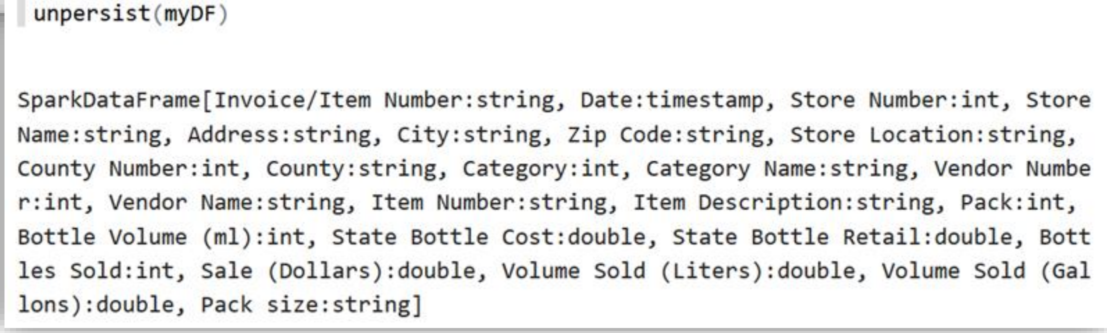
Guardar DataFrame: `write.df`

- DataFrame
- path
- source: tipo de dato para que lo guarde en ese formato
- mode: tipo de guardado:
  - overwrite: si hay un archivo lo elimina
  - append: si hay un archivo, guarda el DataFrame a continuación
  - error: si hay un archivo, salta una excepción
  - ignore: si hay un archivo, no guarda nada.
    - Sólo se guarda el DataFrame si no existe nada

- Cacheo y descacheo: `cache` y `unpersist`

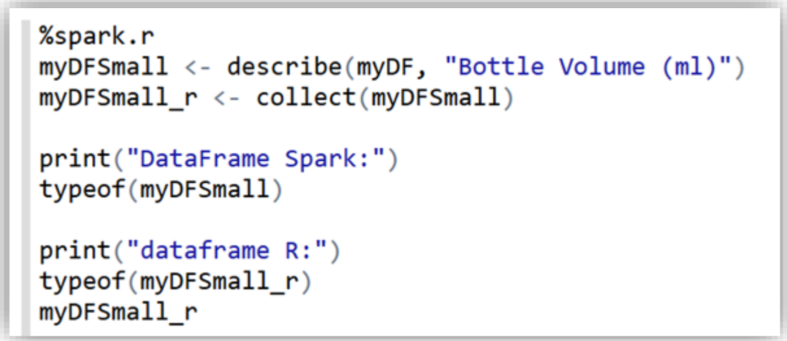


```
cache(myDF)


SparkDataFrame[Invoice/Item Number:string, Date:timestamp, Store Number:int, Store
Name:string, Address:string, City:string, Zip Code:string, Store Location:string,
County Number:int, County:string, Category:int, Category Name:string, Vendor Numbe
r:int, Vendor Name:string, Item Number:string, Item Description:string, Pack:int,
Bottle Volume (ml):int, State Bottle Cost:double, State Bottle Retail:double, Bott
les Sold:int, Sale (Dollars):double, Volume Sold (Liters):double, Volume Sold (Gal
lons):double, Pack size:string]
```

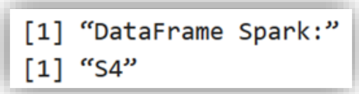


```
unpersist(myDF)


SparkDataFrame[Invoice/Item Number:string, Date:timestamp, Store Number:int, Store
Name:string, Address:string, City:string, Zip Code:string, Store Location:string,
County Number:int, County:string, Category:int, Category Name:string, Vendor Numbe
r:int, Vendor Name:string, Item Number:string, Item Description:string, Pack:int,
Bottle Volume (ml):int, State Bottle Cost:double, State Bottle Retail:double, Bott
les Sold:int, Sale (Dollars):double, Volume Sold (Liters):double, Volume Sold (Gal
lons):double, Pack size:string]
```

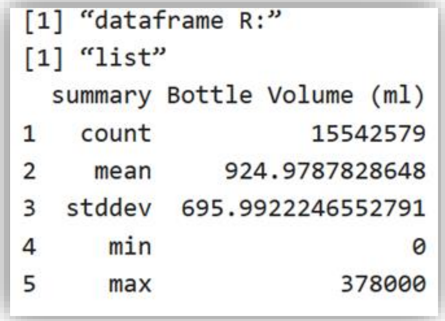
- Crear data.frame: `collect`
  - □ Sólo cuando tenemos pocos datos
  - □ data.frame de R no soporta grandes cantidades de datos



```
%spark.r
myDFSmall <- describe(myDF, "Bottle Volume (ml)")
myDFSmall_r <- collect(myDFSmall)

print("DataFrame Spark:")
typeof(myDFSmall)

print("dataframe R:")
typeof(myDFSmall_r)
myDFSmall_r
```



```
[1] “DataFrame Spark:”
[1] “S4”
```



```
[1] “dataframe R:”
[1] “list”
  summary Bottle Volume (ml)
1   count           15542579
2    mean     924.9787828648
3  stddev   695.992246552791
4     min                  0
5     max             378000
```

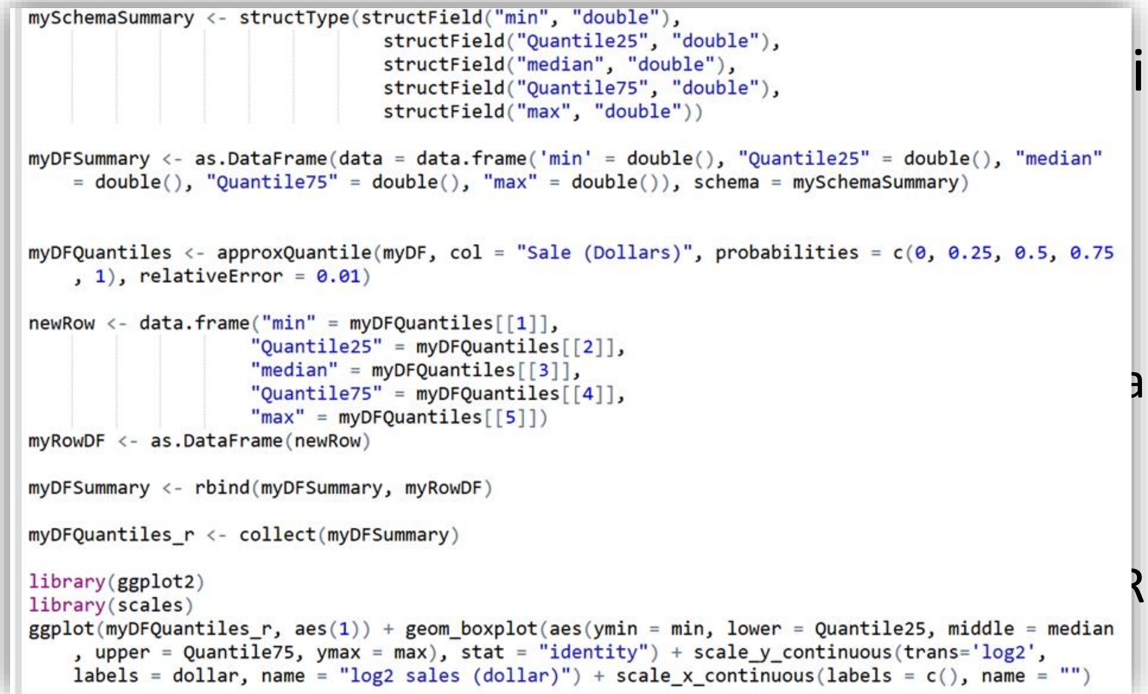
Visualización de datos

- □ Es difícil con grandes cantidades de Datos
- □ SparkR: libraría `ggplot2.SparkR`
  - Utiliza DataFrames de Spark
  - Sólo disponible en versiones antiguas de Spark
  - Extrae todo lo que necesita el gráfico utilizando Spark y luego genera el gráfico con `ggplot` de R

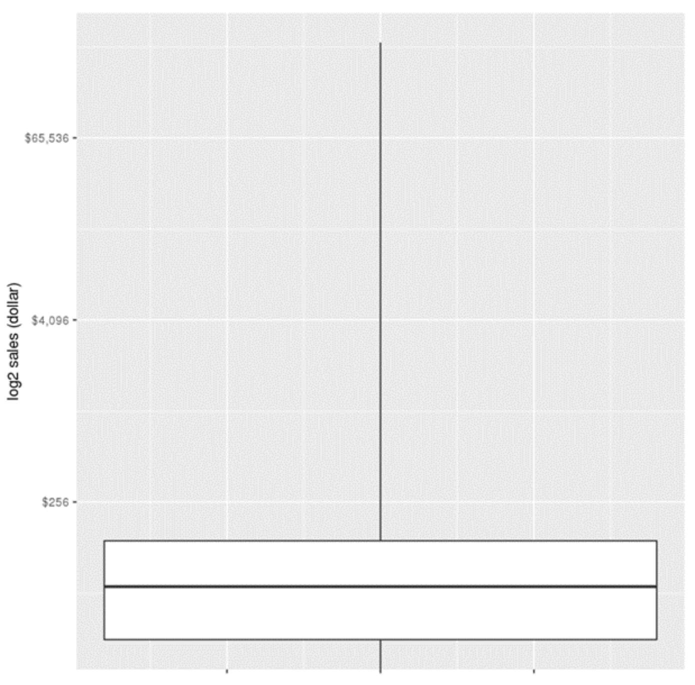
Boxplot

- □ Se necesitan 5 datos: mínimo, primer cuartil, mediana, tercer cuartil y máximo
- □ Pasos:
  - Transformar el DataFrame a un DataFrame con los 5 datos
  - Convertirlo en `data.frame` de R
  - Crear el gráfico con las librerías de R

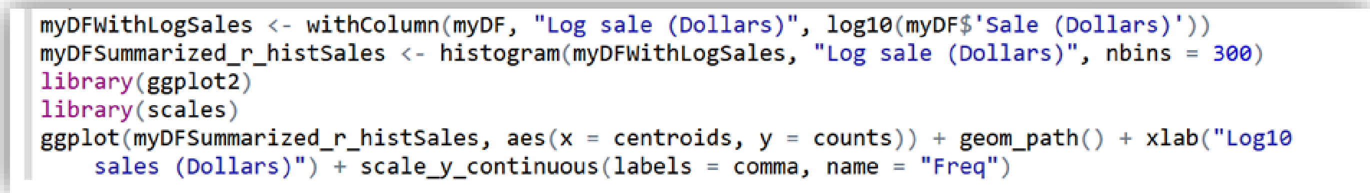
- Boxplot



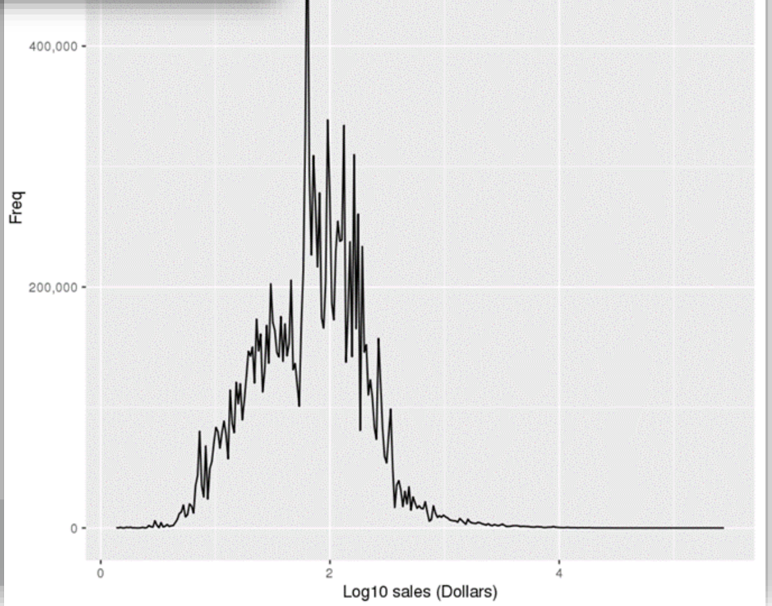
```
mySchemaSummary <- structType(structField("min", "double"),
                              structField("Quantile25", "double"),
                              structField("median", "double"),
                              structField("Quantile75", "double"),
                              structField("max", "double"))

myDFSummary <- as.DataFrame(data = data.frame('min' = double(), "Quantile25" = double(), "median"
    = double(), "Quantile75" = double(), "max" = double()), schema = mySchemaSummary)


myDFQuantiles <- approxQuantile(myDF, col = "Sale (Dollars)", probabilities = c(0, 0.25, 0.5, 0.75
    , 1), relativeError = 0.01)

newRow <- data.frame("min" = myDFQuantiles[[1]],
                     "Quantile25" = myDFQuantiles[[2]],
                     "median" = myDFQuantiles[[3]],
                     "Quantile75" = myDFQuantiles[[4]],
                     "max" = myDFQuantiles[[5]])
myRowDF <- as.DataFrame(newRow)

myDFSummary <- rbind(myDFSummary, myRowDF)

myDFQuantiles_r <- collect(myDFSummary)

library(ggplot2)
library(scales)
ggplot(myDFQuantiles_r, aes(1)) + geom_boxplot(aes(ymin = min, lower = Quantile25, middle = median
    , upper = Quantile75, ymax = max), stat = "identity") + scale_y_continuous(trans='log2',
    labels = dollar, name = "log2 sales (dollar)") + scale_x_continuous(labels = c(), name = "")
```



Histograma

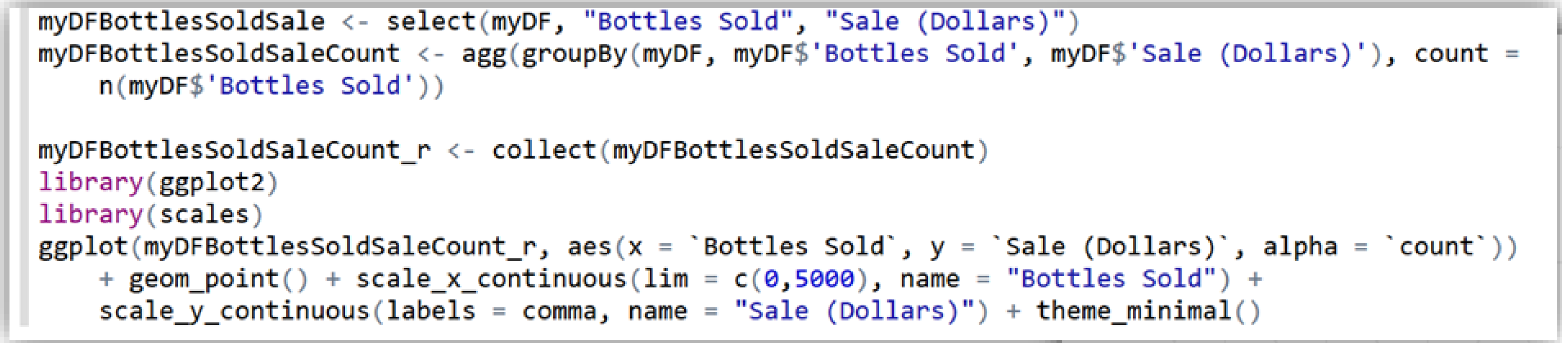


```
myDFWithLogSales <- withColumn(myDF, "Log sale (Dollars)", log10(myDF$'Sale (Dollars)'))
myDFSummarized_r_histSales <- histogram(myDFWithLogSales, "Log sale (Dollars)", nbins = 300)
library(ggplot2)
library(scales)
ggplot(myDFSummarized_r_histSales, aes(x = centroids, y = counts)) + geom_path() + xlab("Log10
    sales (Dollars)") + scale_y_continuous(labels = comma, name = "Freq")
```

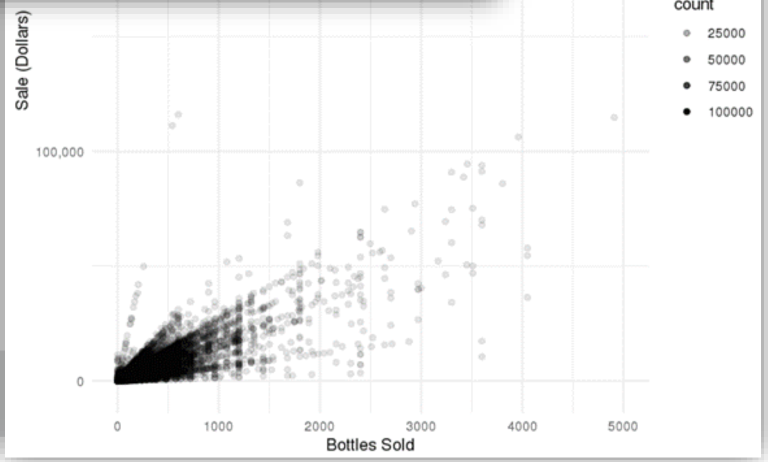


SparkR: Visualización de datos

Dispersión



```
myDFBottlesSoldSale <- select(myDF, "Bottles Sold", "Sale (Dollars)")
myDFBottlesSoldSaleCount <- agg(groupBy(myDF, myDF$'Bottles Sold', myDF$'Sale (Dollars)'), count =
    n(myDF$'Bottles Sold'))

myDFBottlesSoldSaleCount_r <- collect(myDFBottlesSoldSaleCount)
library(ggplot2)
library(scales)
ggplot(myDFBottlesSoldSaleCount_r, aes(x = `Bottles Sold`, y = `Sale (Dollars)`, alpha = `count`))
    + geom_point() + scale_x_continuous(lim = c(0,5000), name = "Bottles Sold") +
    scale_y_continuous(labels = comma, name = "Sale (Dollars)") + theme_minimal()
```



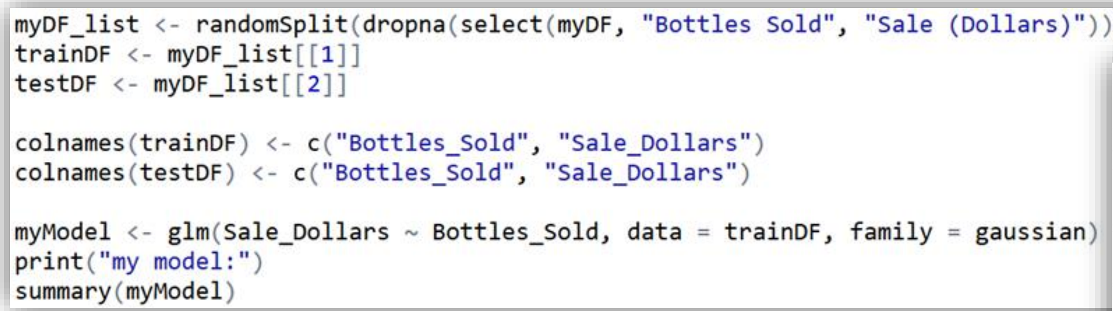
- No todos los modelos de ML son paralelizables
- Spark:
  - Spark ML y MLlib
  - Mahout: https://mahout.apache.org/
  - …

SparkR: Machine Learning

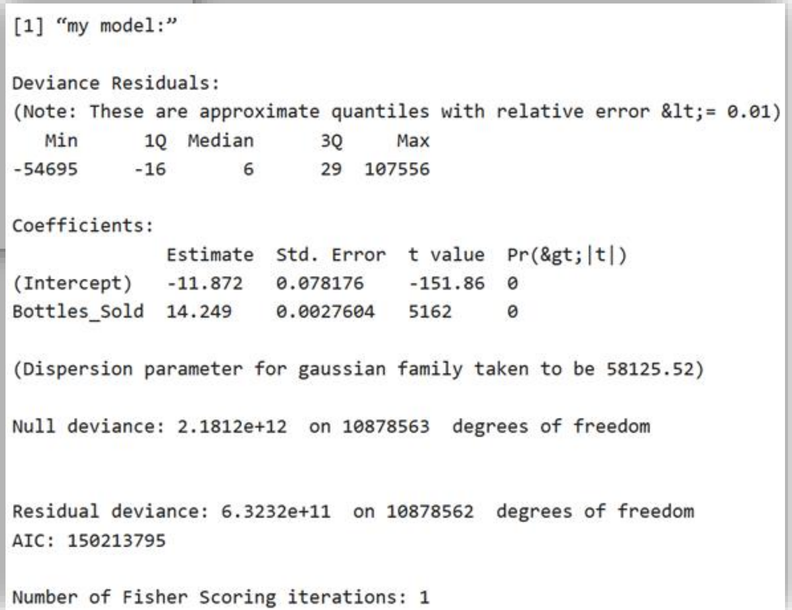
- SparkR
  - □ Classification:
    - `spark.logit`: Logistic Regression
    - `spark.mlp`: Multilayer Perceptron (MLP)
    - `spark.naiveBayes`: Naive Bayes
    - `spark.svmLinear`: Linear Support Vector Machine
    - `spark.fmClassifier`: Factorization Machines classifier
  - □ Regression
    - `spark.survreg`: Accelerated Failure Time (AFT) Survival Model
    - `spark.glm` o `glm`: Generalized Linear Model (GLM)
    - `spark.isoreg`: Isotonic Regression
    - `spark.lm`: Linear Regression
    - `spark.fmRegressor`: Factorization Machines regressor
  - □ Tree
    - `spark.decisionTree`: Decision Tree for Regression and Classification
    - `spark.gbt`: Gradient Boosted Trees for Regression and Classification
    - `spark.randomForest`: Random Forest for Regression and Classification
  - □ Clustering
    - `spark.bisectingKmeans`: Bisecting k-means
    - `spark.gaussianMixture`: Gaussian Mixture Model (GMM)
    - `spark.kmeans`: K-Means
    - `spark.lda`: Latent Dirichlet Allocation (LDA)
    - `spark.powerIterationClustering` (PIC): Power Iteration Clustering (PIC)
  - □ Collaborative Filtering
    - `spark.als`: Alternating Least Squares (ALS)
  - □ Frequent Pattern Mining
    - `spark.fpGrowth`: FP-growth
    - `spark.prefixSpan` : PrefixSpan
  - □ Statistics
    - `spark.kstest`: Kolmogorov-Smirnov Test

Obtenido de la página oficial de Spark (Actualizado a 12/01/2022): https://spark.apache.org/docs/latest/sparkr.html#machine-learning

Modelo de regresión lineal

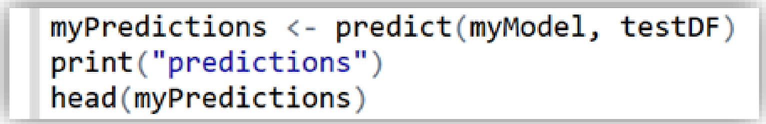


```
myDF_list <- randomSplit(dropna(select(myDF, "Bottles Sold", "Sale (Dollars)")), c(7,3), 2)
trainDF <- myDF_list[[1]]
testDF <- myDF_list[[2]]

colnames(trainDF) <- c("Bottles_Sold", "Sale_Dollars")
colnames(testDF) <- c("Bottles_Sold", "Sale_Dollars")

myModel <- glm(Sale_Dollars ~ Bottles_Sold, data = trainDF, family = gaussian)
print("my model:")
summary(myModel)
```

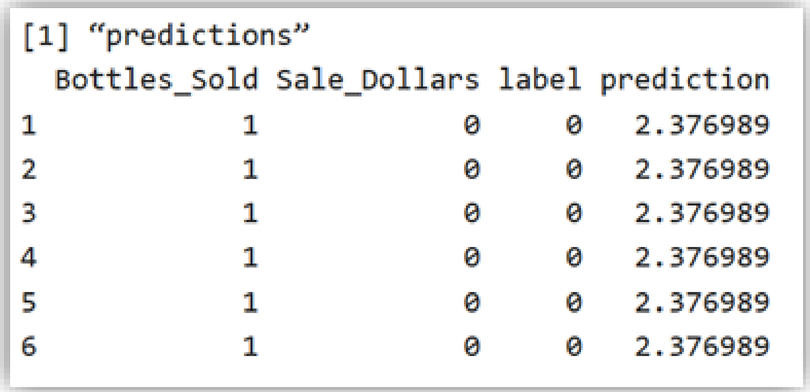


```
[1] “my model:”

Deviance Residuals:
(Note: These are approximate quantiles with relative error &lt;= 0.01)
   Min      1Q  Median      3Q     Max
-54695     -16       6      29  107556

Coefficients:
             Estimate  Std. Error  t value  Pr(&gt;|t|)
(Intercept)  -11.872   0.078176    -151.86  0
Bottles_Sold 14.249    0.0027604   5162     0

(Dispersion parameter for gaussian family taken to be 58125.52)

Null deviance: 2.1812e+12  on 10878563  degrees of freedom


Residual deviance: 6.3232e+11  on 10878562  degrees of freedom
AIC: 150213795

Number of Fisher Scoring iterations: 1
```

Modelo de regresión lineal

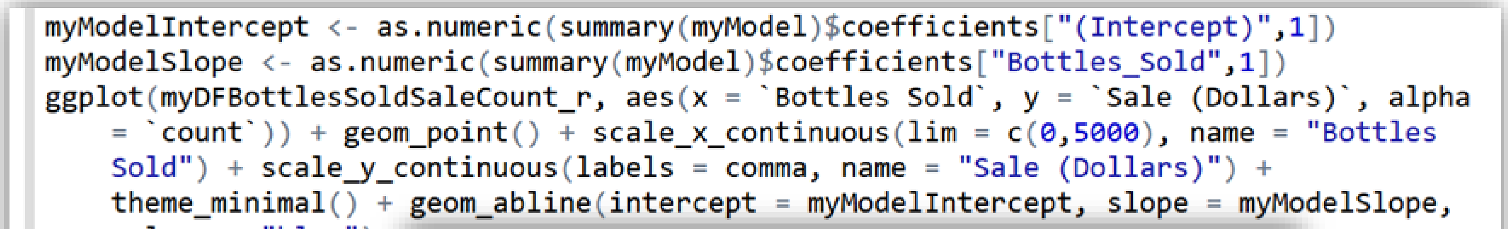


```
myPredictions <- predict(myModel, testDF)
print("predictions")
head(myPredictions)
```

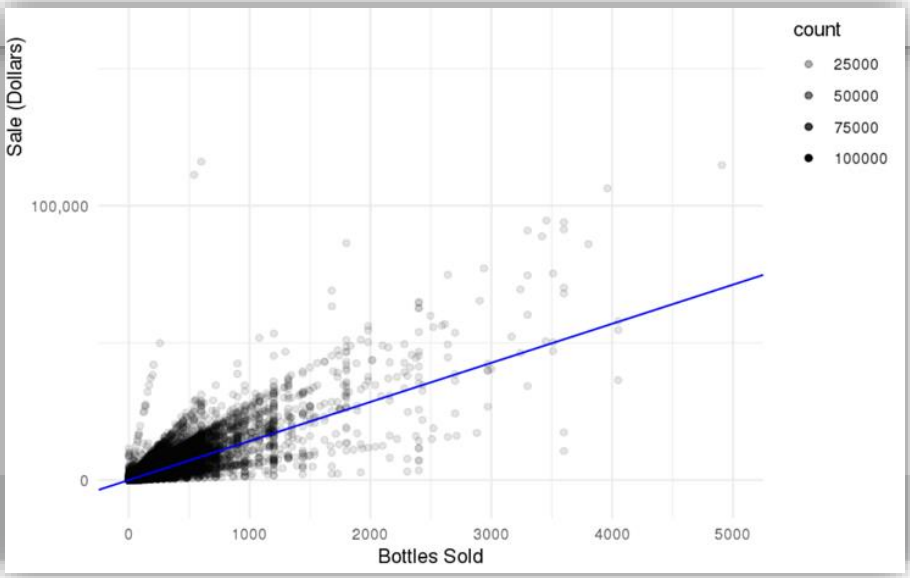


```
[1] “predictions”
  Bottles_Sold Sale_Dollars label prediction
1            1            0     0   2.376989
2            1            0     0   2.376989
3            1            0     0   2.376989
4            1            0     0   2.376989
5            1            0     0   2.376989
6            1            0     0   2.376989
```

Modelo de regresión lineal

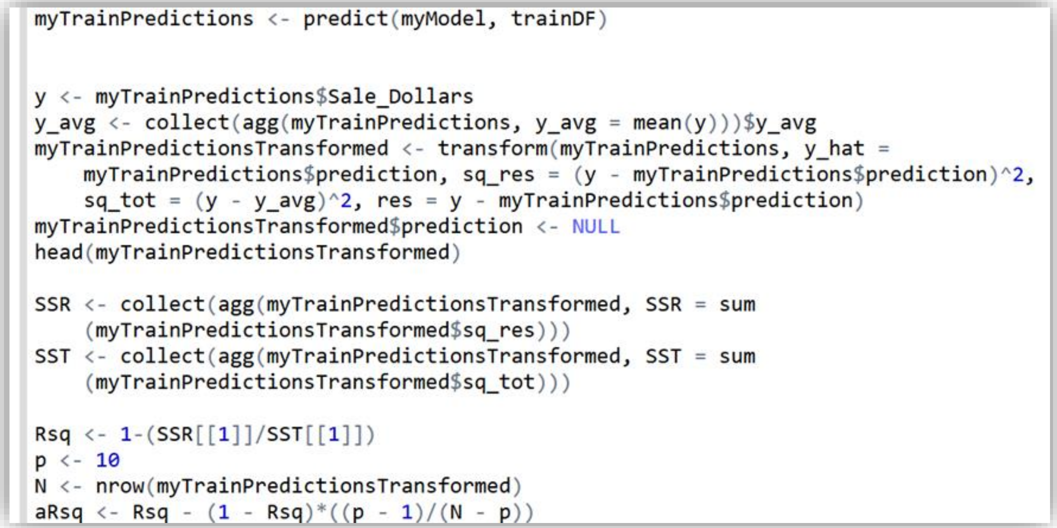


```
myModelIntercept <- as.numeric(summary(myModel)$coefficients["(Intercept)",1])
myModelSlope <- as.numeric(summary(myModel)$coefficients["Bottles_Sold",1])
ggplot(myDFBottlesSoldSaleCount_r, aes(x = `Bottles Sold`, y = `Sale (Dollars)`, alpha
    = `count`)) + geom_point() + scale_x_continuous(lim = c(0,5000), name = "Bottles
    Sold") + scale_y_continuous(labels = comma, name = "Sale (Dollars)") +
    theme_minimal() + geom_abline(intercept = myModelIntercept, slope = myModelSlope,
    colour = "blue")
```

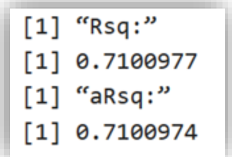


Modelo de regresión lineal

- Rsquared: (código obtenido de https://github.com/UrbanInstitute/sparkr-tutorials/blob/master/09_glm.md)



```
myTrainPredictions <- predict(myModel, trainDF)


y <- myTrainPredictions$Sale_Dollars
y_avg <- collect(agg(myTrainPredictions, y_avg = mean(y)))$y_avg
myTrainPredictionsTransformed <- transform(myTrainPredictions, y_hat =
    myTrainPredictions$prediction, sq_res = (y - myTrainPredictions$prediction)^2,
    sq_tot = (y - y_avg)^2, res = y - myTrainPredictions$prediction)
myTrainPredictionsTransformed$prediction <- NULL
head(myTrainPredictionsTransformed)

SSR <- collect(agg(myTrainPredictionsTransformed, SSR = sum
    (myTrainPredictionsTransformed$sq_res)))
SST <- collect(agg(myTrainPredictionsTransformed, SST = sum
    (myTrainPredictionsTransformed$sq_tot)))

Rsq <- 1-(SSR[[1]]/SST[[1]])
p <- 10
N <- nrow(myTrainPredictionsTransformed)
aRsq <- Rsq - (1 - Rsq)*((p - 1)/(N - p))
```



```
[1] “Rsq:”
[1] 0.7100977
[1] “aRsq:”
[1] 0.7100974
```

SparkR

- R vs SparkR
- Instalación SparkR
- Operaciones SparkR
- Visualización de datos
- Machine Learning

Gracias

viu Universidad Internacional de Valencia