Managing Large and Big Data with R

Dhafer Malouche

http://dhafermalouche.net

What's a large and massive data?

- R is not well-suited for working with data structures larger than about 10-20% of a computer's RAM.
- Data exceeding 50% of available RAM are essentially unusable.
- We consider a data set **large** if it exceeds 20% of the RAM on a given machine and **massive** if it exceeds 50%

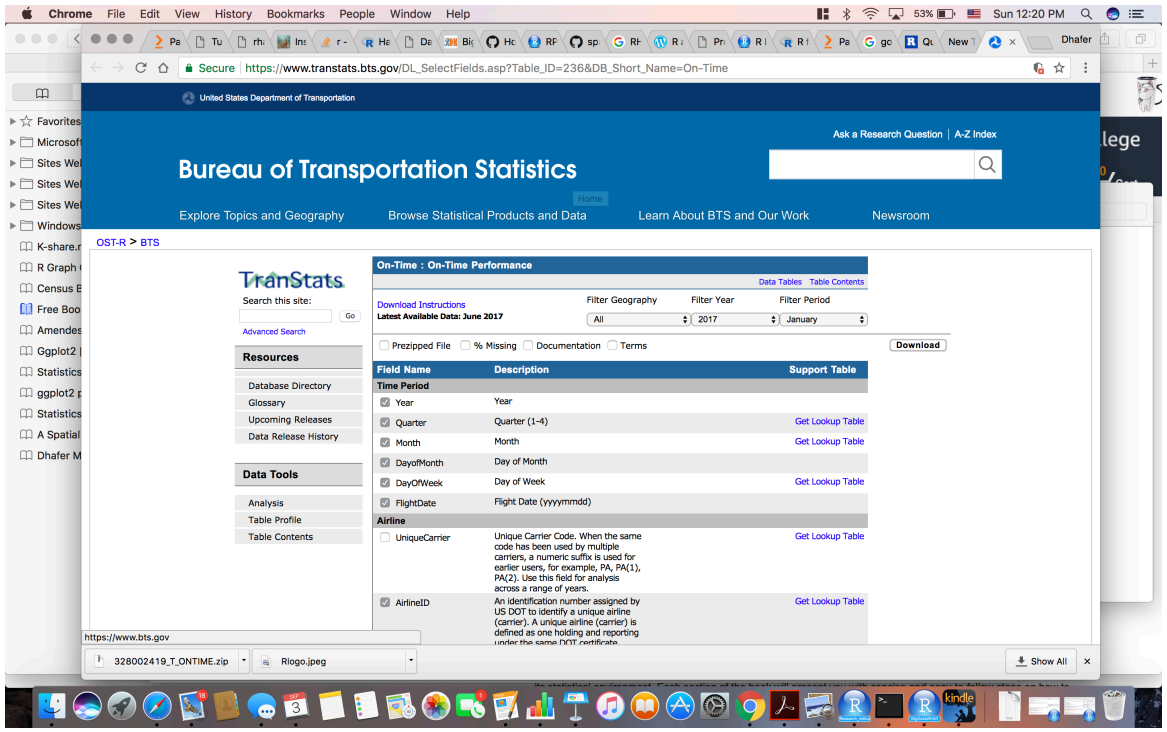
Importing Large data into R: ff package

Installation

```
> install.packages("ff")
> install.packages("ffbase")
> install.packages("ffbase2")
> library(ff)
> library(ffbase)
> library(ffbase2)
```

First Example

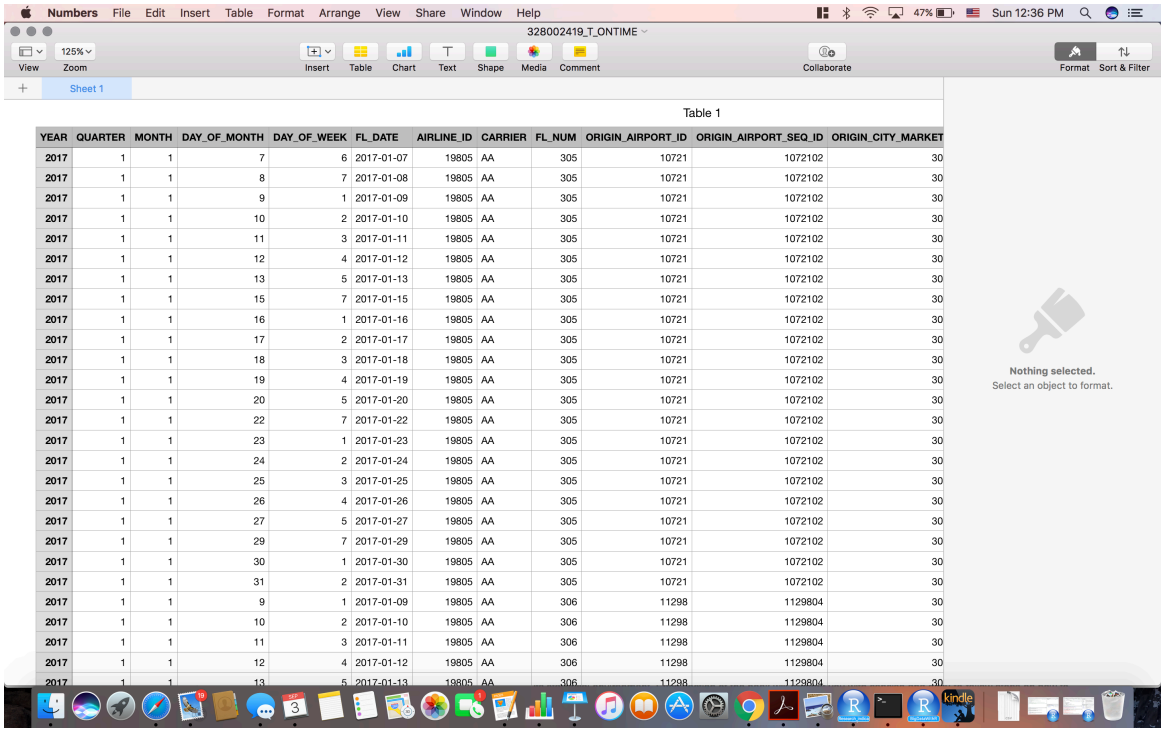
Importing data into R



Source: Link to download the data.

A data with size 3.4MB

We extract data for one day



YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	AIRLINE_ID	CARRIER	FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN_AIRPORT_SEQ_ID	ORIGIN_CITY_MARKET
2017	1	1	7	6	2017-01-07	19805	AA	305	10721	1072102	30
2017	1	1	8	7	2017-01-08	19805	AA	305	10721	1072102	30
2017	1	1	9	1	2017-01-09	19805	AA	305	10721	1072102	30
2017	1	1	10	2	2017-01-10	19805	AA	305	10721	1072102	30
2017	1	1	11	3	2017-01-11	19805	AA	305	10721	1072102	30
2017	1	1	12	4	2017-01-12	19805	AA	305	10721	1072102	30
2017	1	1	13	5	2017-01-13	19805	AA	305	10721	1072102	30
2017	1	1	15	7	2017-01-15	19805	AA	305	10721	1072102	30
2017	1	1	16	1	2017-01-16	19805	AA	305	10721	1072102	30
2017	1	1	17	2	2017-01-17	19805	AA	305	10721	1072102	30
2017	1	1	18	3	2017-01-18	19805	AA	305	10721	1072102	30
2017	1	1	19	4	2017-01-19	19805	AA	305	10721	1072102	30
2017	1	1	20	5	2017-01-20	19805	AA	305	10721	1072102	30
2017	1	1	22	7	2017-01-22	19805	AA	305	10721	1072102	30
2017	1	1	23	1	2017-01-23	19805	AA	305	10721	1072102	30
2017	1	1	24	2	2017-01-24	19805	AA	305	10721	1072102	30
2017	1	1	25	3	2017-01-25	19805	AA	305	10721	1072102	30
2017	1	1	26	4	2017-01-26	19805	AA	305	10721	1072102	30
2017	1	1	27	5	2017-01-27	19805	AA	305	10721	1072102	30
2017	1	1	29	7	2017-01-29	19805	AA	305	10721	1072102	30
2017	1	1	30	1	2017-01-30	19805	AA	305	10721	1072102	30
2017	1	1	31	2	2017-01-31	19805	AA	305	10721	1072102	30
2017	1	1	9	1	2017-01-09	19805	AA	306	11298	1129804	30
2017	1	1	10	2	2017-01-10	19805	AA	306	11298	1129804	30
2017	1	1	11	3	2017-01-11	19805	AA	306	11298	1129804	30
2017	1	1	12	4	2017-01-12	19805	AA	306	11298	1129804	30
2017	1	1	13	5	2017-01-13	19805	AA	306	11298	1129804	30

First steps with `ff` package

1. Create the directory where the data will be stored

```
> system("mkdir ffdf")  ## If you're a MAC user
```

First steps with `ff` package

2. Path to the newly created directory

```
> options(fftempdir = "/Users/dhafermalouche/Documents/Teaching/BigDataWithR/ffdf")
```

First steps with `ff` package

3. Importing the data now

```
> ptm<-proc.time()
> flights.ff<- read.csv.ffdf(file = "flight_ff.csv",VERBOSE = TRUE,first.rows = -1,header=T)
read.table.ffdf 1..65499 (65499)  csv-read=0.709sec ffdf-write=0.061sec
 csv-read=0.709sec  ffdf-write=0.061sec  TOTAL=0.77sec
> proc.time()-ptm
   user  system elapsed
  0.725   0.035   0.774
```

Comparing with the classical procedure

```
> library(readr)
> ptm<-proc.time()
> flights.classic <- read_csv("flight_ff.csv")
> proc.time()-ptm
   user  system elapsed 
  0.081   0.009   0.093 
```

Comparing the memory that is being used to store the data.

```
> format(object.size(flights.classic),"Mb")
[1] "4.3 Mb"
> format(object.size(flights.ff),"Mb")
[1] "0.1 Mb"
```

Created files with `ff` package

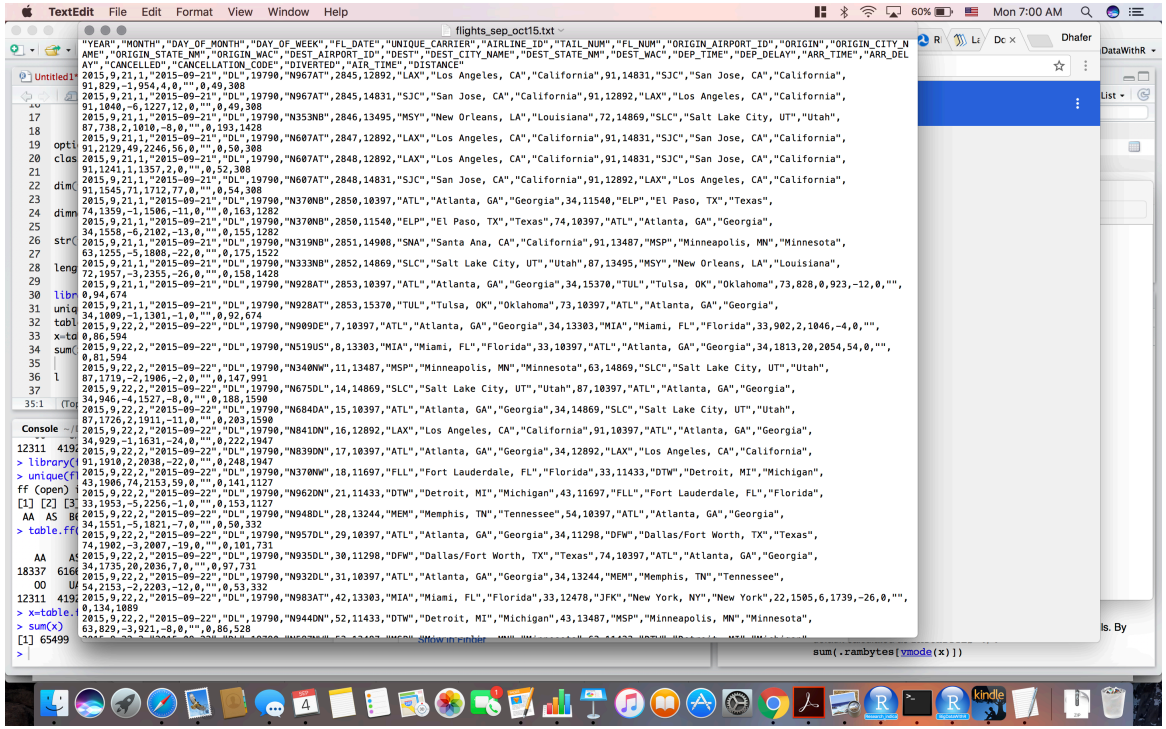
```
> list.files(path="/Users/dhafermalouche/Documents/Teaching/BigDataWithR/ffdf")
 [1] "ffdf345a19cd9166.ff" "ffdf345a228444fe.ff" "ffdf345a2d71c94f.ff"
 [4] "ffdf345a39d522ae.ff"
... #truncated output
```

Correspondance file <-> variable

```
> basename(filename(flights.ff$YEAR))
[1] "ffdf345a54b8c0ad.ff"
> dir.exists("/Users/dhafermalouche/Documents/Teaching/BigDataWithR/ffdf")
[1] TRUE
> file.exists("/Users/dhafermalouche/Documents/Teaching/BigDataWithR/ffdf/ffdf345a54b8c0ad.ff")
[1] TRUE
```

Second Example; larger dataset: ~150 MB

Month of September in txt format.



200,000 rows will be read in further chunk, with `ff`

```
> system("mkdir ffdf2")
> options(fftempdir = "/Users/dhafermalouche/Documents/Teaching/BigDataWithR/ffdf2")
> ptm<-proc.time()
> flights.ff2<-read.table.ffdf(file="flights_sep_oct15.txt",
+                              sep=",", VERBOSE=TRUE,header=TRUE,
+                              next.rows=200000,colClasses=NA)
read.table.ffdf 1..200000 (200000)  csv-read=4.14sec ffdf-write=0.728sec
read.table.ffdf 200001..400000 (200000)  csv-read=4.139sec ffdf-write=0.201sec
read.table.ffdf 400001..600000 (200000)  csv-read=4.231sec ffdf-write=0.191sec
read.table.ffdf 600001..800000 (200000)  csv-read=4.088sec ffdf-write=0.212sec
read.table.ffdf 800001..951111 (151111)  csv-read=3.122sec ffdf-write=0.191sec
 csv-read=19.72sec  ffdf-write=1.523sec  TOTAL=21.243sec
> proc.time()-ptm
   user  system elapsed
 20.191   0.892  21.250
```

Comparing with the classical procedure

```
> ptm<-proc.time()
> flights.classic2 <- read.csv("~/Documents/Teaching/BigDataWithR/flights_sep_oct15.txt")
> proc.time()-ptm
   user  system elapsed 
 19.379   0.346  19.814 
> format(object.size(flights.classic2),"Mb")
[1] "101.9 Mb"
> format(object.size(flights.ff2),"Mb")
[1] "0.4 Mb"
```

Saving `ff` object

- It will be saved in 29 flat files, each variable in one file. All will be stored in `saved_flights2` directory (in your workdirectory).

```
> save.ffdf(flights.ff2,dir = "saved_flights2")
```

Loading an `ff` object

- Remove it and load it again

```
> rm("flights.ff2")
> ls()
[1] "ptm"
> load.ffdf("/Users/dhafermalouche/Documents/Teaching/BigDataWithR/saved_flights2")
> ls()
[1] "flights.ff2" "ptm"
```

Exporting it in csv format

```
> file.size("flight_from_ff.csv")/1024/1024
[1] 149.0488
> file.size("flights_sep_oct15.txt")/1024/1024
[1] 149.0488
```

Massive data within R: bigmemory and data.table packages

- We consider a data file with ~ 1.5Gb
- We import airline_20MM.csv

```
> file.size("airline_20MM.csv")/1024/1024/1024
[1] 1.50516
```

`bigmemory`, `data.table` package

It can be used for

- massive data: solutions for managing and exploring massive data
- data analysis: using also `foreach` package.

Importing the data using `bigmemory`

```
> library(bigmemory)
> ptm<-proc.time()
> flights.bm <- read.big.matrix("airline_20MM.csv", header =TRUE ,
+                               backingfile ="airline.bin",
+                               descriptorfile ="airline.desc ",
+                               type ="integer")
> proc.time()-ptm
   user  system elapsed 
125.920   4.670 139.686 
> object.size(flights.bm)
664 bytes
> dim(flights.bm)
[1] 2.0e+07 2.6e+01
```

With `data.table` package

```
> library(data.table)
> ptm<-proc.time()
> flights.dt <- fread("airline_20MM.csv", stringsAsFactors = TRUE)
Read 20000000 rows and 26 (of 26) columns from 1.505 GB file in 00:00:20
> proc.time()-ptm
   user  system elapsed 
 16.027   3.426  32.394 
> format(object.size(flights.dt),"Mb")
[1] "1983.6 Mb"
```

Example: Netflix data

Description and Source of the data

- Netflix held the Netflix Prize open competition for the best algorithm to predict user ratings for films. The grand prize was $1,000,000 and was won by BellKor's Pragmatic Chaos team. This is the dataset that was used in that competition.
- It can be downloaded here
- Several files among them 4 txt files ~ 470MB each and our aim is to import these files into `R`, compose only one dataset and make some statistical operations.

How is the data

```
1:
1488844,3,2005-09-06
822109,5,2005-05-13
885013,4,2005-10-19
30878,4,2005-12-26
823519,3,2004-05-03
893988,3,2005-11-17
124105,4,2004-08-05
```

Importing the data into R with ff (file 1)

```
> system("mkdir netflix1")
> options(fftempdir = "/Users/dhafermalouche/Documents/Teaching/BigDataWithR/netflix1")
> d1<-read.table.ffdf(file="combined_data_1.txt",
+                     VERBOSE=TRUE,header=F,
+                    next.rows=200000,colClasses=NA)
read.table.ffdf 1..200000 (200000)  csv-read=2.076sec ffdf-write=0.024sec
read.table.ffdf 200001..400000 (200000)  csv-read=2.087sec ffdf-write=0.224sec
read.table.ffdf 400001..600000 (200000)  csv-read=2.412sec ffdf-write=0.242sec
... #Truncated output
read.table.ffdf 24000001..24058263 (58263)  csv-read=0.646sec ffdf-write=4.699sec
 csv-read=311.57sec  ffdf-write=439.119sec  TOTAL=750.689sec
```

Importing the data into R with ff (file 2)

```
> system("mkdir netflix2")
> options(fftempdir = "/Users/dhafermalouche/Documents/Teaching/BigDataWithR/netflix2")
> d2<-read.table.ffdf(file="combined_data_2.txt",
+                     VERBOSE=TRUE,header=F,
+                     next.rows=200000,colClasses=NA)
read.table.ffdf 1..200000 (200000)  csv-read=3.047sec ffdf-write=0.043sec
read.table.ffdf 200001..400000 (200000)  csv-read=2.59sec ffdf-write=0.194sec
read.table.ffdf 400001..600000 (200000)  csv-read=2.626sec ffdf-write=0.269sec
....
read.table.ffdf 26800001..26982302 (182302)  csv-read=2.294sec ffdf-write=7.239sec
 csv-read=370.196sec  ffdf-write=729.266sec  TOTAL=1099.462sec
```

Importing the data into R with ff (file 3)

```
> system("mkdir netflix3")
> options(fftempdir = "/Users/dhafermalouche/Documents/Teaching/BigDataWithR/netflix3")
> d3<-read.table.ffdf(file="combined_data_2.txt",
+                     VERBOSE=TRUE,header=F,
+                     next.rows=200000,colClasses=NA)
read.table.ffdf 1..200000 (200000)  csv-read=6.331sec ffdf-write=0.075sec
read.table.ffdf 200001..400000 (200000)  csv-read=2.926sec ffdf-write=0.148sec
read.table.ffdf 400001..600000 (200000)  csv-read=2.826sec ffdf-write=0.268sec

....
read.table.ffdf 22600001..22605786 (5786)  csv-read=0.067sec ffdf-write=4.328sec
 csv-read=895.66sec  ffdf-write=503.358sec  TOTAL=1399.018sec
```

Importing the data into R with ff (file 4)

```
> system("mkdir netflix4")
> options(fftempdir = "/Users/dhafermalouche/Documents/Teaching/BigDataWithR/netflix4")
> d4<-read.table.ffdf(file="combined_data_4.txt",
+                     VERBOSE=TRUE,header=F,
+                     next.rows=200000,colClasses=NA)
read.table.ffdf 1..200000 (200000)  csv-read=2.467sec ffdf-write=0.027sec
read.table.ffdf 200001..400000 (200000)  csv-read=5.037sec ffdf-write=0.189sec
...
read.table.ffdf 26800001..26851926 (51926)  csv-read=0.552sec ffdf-write=8.004sec
 csv-read=340.948sec  ffdf-write=667.89sec  TOTAL=1008.838sec
```

Importing the data with `data.table` (file 1)

```
> ptm<-proc.time()
> d1.dt <- fread("combined_data_1.txt", header = F,sep = "\t")
Read 24058263 rows and 1 (of 1) columns from 0.461 GB file in 00:00:14
> proc.time()-ptm
   user  system elapsed 
 13.074   0.185  13.300 
> format(object.size(d1.dt),"Mb")
[1] "1019 Mb"
```

Importing the data with `data.table` (file 2)

```
> ptm<-proc.time()
> d2.dt <- fread("combined_data_2.txt", header = F,sep = "\t")
Read 26982302 rows and 1 (of 1) columns from 0.517 GB file in 00:00:39
> proc.time()-ptm
   user  system elapsed 
 37.590   1.000  40.916 
> format(object.size(d2.dt),"Mb")
[1] "1071.5 Mb"
```

Importing the data with `data.table` (file 3)

```
> ptm<-proc.time()
> d3.dt <- fread("combined_data_3.txt", header = F,sep = "\t")
Read 22605786 rows and 1 (of 1) columns from 0.433 GB file in 00:00:37
> proc.time()-ptm
   user  system elapsed 
 34.973   1.229  38.645 
> format(object.size(d3.dt),"Mb")
[1] "954 Mb"
```

Importing the data with `data.table` (file 4)

```
> ptm<-proc.time()
> d4.dt <- fread("combined_data_4.txt", header = F,sep = "\t")
Read 26851926 rows and 1 (of 1) columns from 0.515 GB file in 00:00:38
> proc.time()-ptm
   user  system elapsed 
 35.991   1.117  39.677 
> format(object.size(d4.dt),"Mb")
[1] "1076.8 Mb"
```

SQLite,

Installation (on mac)

Installing `SQLite` on your mac

1. Press Command+Space and type *Terminal* and press *enter/return* key
2. Run in Terminal app

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
< /dev/null 2> /dev/null
```

3. Run

```
brew install sqlite
```

Checking Installation

```
$ sqlite3
SQLite version 3.16.0 2016-11-04 19:09:39
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

First steps on SQLite

1. Navigate to the directory with your data,

```
$ cd /Users/dhafermalouche/Documents/Teaching/BigDataWithR
```

First steps on SQLite

2. Start SQLite by creating a new database, we call it flightsBD (set of tables)

```
$ sqlite3 flightsDB
```

First steps on SQLite

3. Type `.databases` to show all currently available databases:

```
sqlite> .databases
seq  name             file
---  ---------------  ----------------------------------------------------------
0    main             /Users/dhafermalouche/Documents/Teaching/BigDataWithR/fligthsDB
```

Importing an csv file (`flight.csv` ~156 MB)

1. Type of the separator in the `csv` file

```
sqlite> .separator ","
```

Importing a csv file

2. Importing the `csv` file as a new table called `flightsTB`

```
sqlite> .import flight.csv flightsTB
```

Importing a csv file

3. Check the available tables by running the `.tables`

```
sqlite> .tables
flightsTB
```

Checking the data in SQLite

Let's inspect the structure of the table using a PRAGMA statement:

```
sqlite> PRAGMA table_info("flightsTB");
0,YEAR,TEXT,0,,0
1,MONTH,TEXT,0,,0
2,DAY_OF_MONTH,TEXT,0,,0
3,DAY_OF_WEEK,TEXT,0,,0
4,FL_DATE,TEXT,0,,0
5,UNIQUE_CARRIER,TEXT,0,,0
6,AIRLINE_ID,TEXT,0,,0
7,TAIL_NUM,TEXT,0,,0
8,FL_NUM,TEXT,0,,0
9,ORIGIN_AIRPORT_ID,TEXT,0,,0
10,ORIGIN,TEXT,0,,0
11,ORIGIN_CITY_NAME,TEXT,0,
...
```

Checking the data in SQLite

- We print the scheme of the table using .scheme (names of the variables, their classes)

```
sqlite> .schema flightsTB
CREATE TABLE flightsTB(
  "YEAR" TEXT,
  "MONTH" TEXT,
  "DAY_OF_MONTH" TEXT,
  "DAY_OF_WEEK" TEXT,
  "FL_DATE" TEXT,
  "UNIQUE_CARRIER" TEXT,
  "AIRLINE_ID" TEXT,
  "TAIL_NUM" TEXT,
  "FL_NUM" TEXT,
...   output truncated
```

Connecting SQLite from `RStudio`

1. Make sure that the file `flightsDB` newly created (in `RStudio`)

```
> file.exists("flightsDB")
[1] TRUE
```

Connecting SQLite from RStudio

2. Installing the R packages for SQLite

```
> install.packages("devtools")
> devtools::install_github("RcppCore/Rcpp")
> devtools::install_github("rstats-db/DBI")
> install.packages("RSQLite")
```

Connecting SQLite from `RStudio`

3. Load the packages

```
> library(DBI)
> library(RSQLite)
```

Connecting SQLite from `RStudio`

4. We create a connection with the `flightsDB` SQLite database:

```
> con<-dbConnect(RSQLite::SQLite(), "flightsDB")
> con
<SQLiteConnection>
  Path: /Users/dhafermalouche/Documents/Teaching/BigDataWithR/flightsDB
  Extensions: TRUE
```

Managing SQL databases from `RStudio`

- `dbListTables()` provides information on the available tables in the connected database and columns within a specified table,

```
> dbListTables(con)
[1] "flightsTB"  "query_2_TB"
```

- You can also remove tables from your database.

```
> dbRemoveTable(con, "query_2_TB")
> dbListTables(con)
[1] "flightsTB"
```

Managing SQL databases from RStudio

- `dbListFields()` shows the names of the variables in the table

```
> dbListFields(con,"flightsTB")
 [1] "YEAR"              "MONTH"             "DAY_OF_MONTH"      "DAY_OF_WEEK"
 [5] "FL_DATE"           "UNIQUE_CARRIER"    "AIRLINE_ID"        "TAIL_NUM"
 [9] "FL_NUM"            "ORIGIN_AIRPORT_ID" "ORIGIN"            "ORIGIN_CITY_NAME"
[13] "ORIGIN_STATE_NM"
... # truncated output
```

Creating a new table from the other table

```
> query.1 <- dbSendQuery(con, "SELECT * FROM flightsTB WHERE MONTH = 9")
> dbGetStatement(query.1)
[1] "SELECT * FROM flightsTB WHERE MONTH = 9"
```

Manipulating data with SQL statements

```
> query.1.res <- fetch(query.1, n=20)
> class(query.1.res)
[1] "data.frame"
> head(query.1.res[,1:4])
  YEAR MONTH DAY_OF_MONTH DAY_OF_WEEK
1 2015     9           22           2
2 2015     9           22           2
3 2015     9           22           2
4 2015     9           22           2
5 2015     9           22           2
6 2015     9           22           2
> dim(query.1.res)
[1] 20 29
```

Getting informations on the new data

- We can obtain additional information, for example its full SQL statement, the structure of the results set, and how many rows it returned:

```
> info <- dbGetInfo(query.1)
> str(info)
List of 4
 $ statement    : chr "SELECT * FROM flightsTB WHERE MONTH = 9"
 $ row.count    : int 40
 $ rows.affected: int 0
 $ has.completed: logi FALSE
```

Manipulating data with SQL statements

- We can clear the obtained results

```
> dbClearResult(query.1)
```

Manipulating data with SQL statements

Retrieving all records from the month 9.

```
> query.1 <- dbSendQuery(con,
+                        "SELECT * FROM flightsTB WHERE MONTH = 9")
> ptm=proc.time()
> query.0.res <- fetch(query.1, n=-1)
> proc.time()-ptm
   user  system elapsed
  8.252   0.256   8.804
> dim(query.0.res)
[1] 929892     29
> format(object.size(query.0.res),"Mb")
[1] "206.6 Mb"
> class(query.0.res)
[1] "data.frame"
```

Operations on the data with SQL statements

Computing average of delays according to Departure and Arrival cities.

```
> ptm=proc.time()
> query.2 <- dbSendQuery(con, "SELECT ORIGIN_CITY_NAME, DEST_CITY_NAME,
+ AVG(DEP_DELAY) AS 'AVR_DEP_DELAY',AVG(ARR_DELAY) AS 'AVR_DEP_DELAY',
+                       COUNT(DEP_DELAY) AS 'NUMB'
+                       FROM flightsTB  GROUP BY ORIGIN_CITY_NAME, DEST_CITY_NAME"
+ )
Warning message:
Closing open result set, pending rows
> proc.time()-ptm
   user  system elapsed
  1.308   0.544   1.865
>
> ptm=proc.time()
> query.2.res <- fetch(query.2, n=-1)
> proc.time()-ptm
   user  system elapsed
  1.129   0.091   1.240
>
> dim(query.2.res)
[1] 3746    5
```

Getting informations on the data

```
> info2 <- dbGetInfo(query.2)
> info2
$statement
[1] "SELECT ORIGIN_CITY_NAME, DEST_CITY_NAME,\nAVG(DEP_DELAY) AS 'AVR_DEP_DELAY',AVG(ARR_DELAY) AS 'AVR_DEP_DELAY', \n                    COUNT(DEP_DELAY) AS 'NUMB

$row.count
[1] 3746

$rows.affected
[1] 0

$has.completed
[1] TRUE
```

Exporting the data into a table in the SQL database

```
> ptm=proc.time()
> dbWriteTable(conn = con,name =  "query_2_TB",
+              value =  query.2.res[,1:5],
+              overwrite = TRUE, row.names = FALSE)
> proc.time()-ptm
   user  system elapsed 
  0.113   0.007   0.126 
> dbListTables(con)
[1] "flightsTB"  "query_2_TB"
```

Spark

What's Spark?

- Apache Spark is a powerful open source processing engine built around speed, ease of use, and sophisticated analytics. It was originally developed at UC Berkeley in 2009.
- Spark can be 100x faster than Hadoop for large scale data processing by exploiting in memory computing and other optimizations.

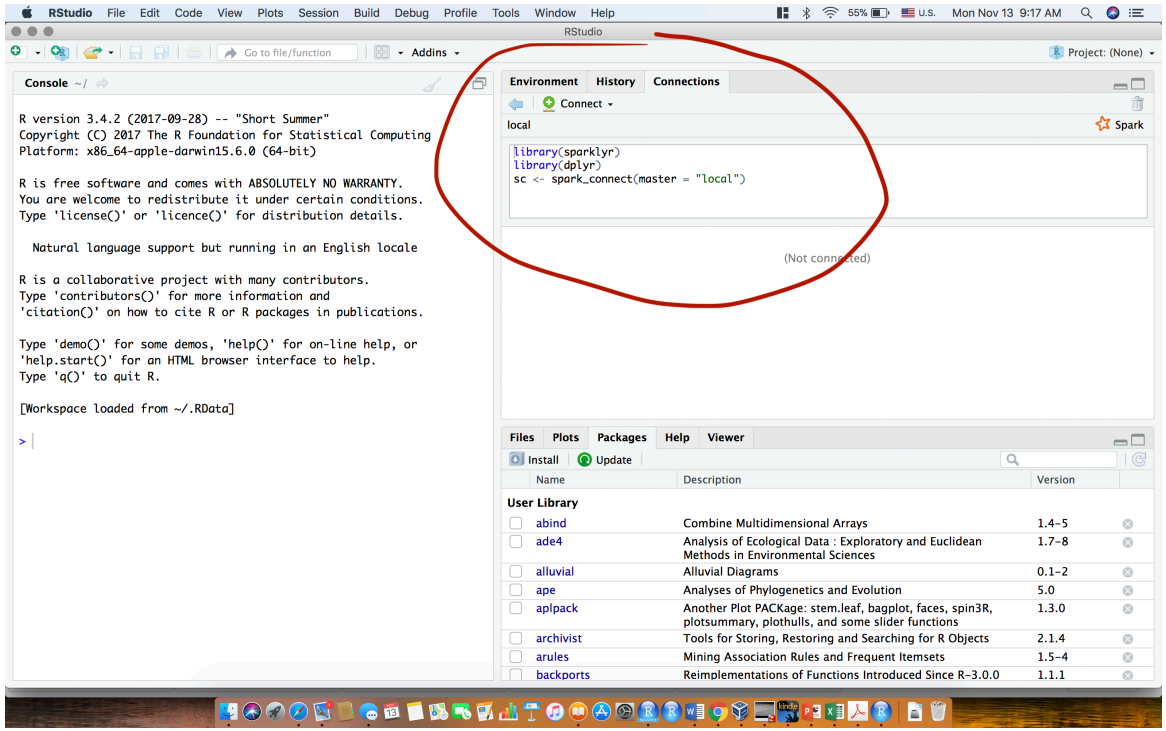
Spark From RStudio

The latest RStudio Preview Release of the RStudio IDE includes integrated support for Spark and the sparklyr package, including tools for:

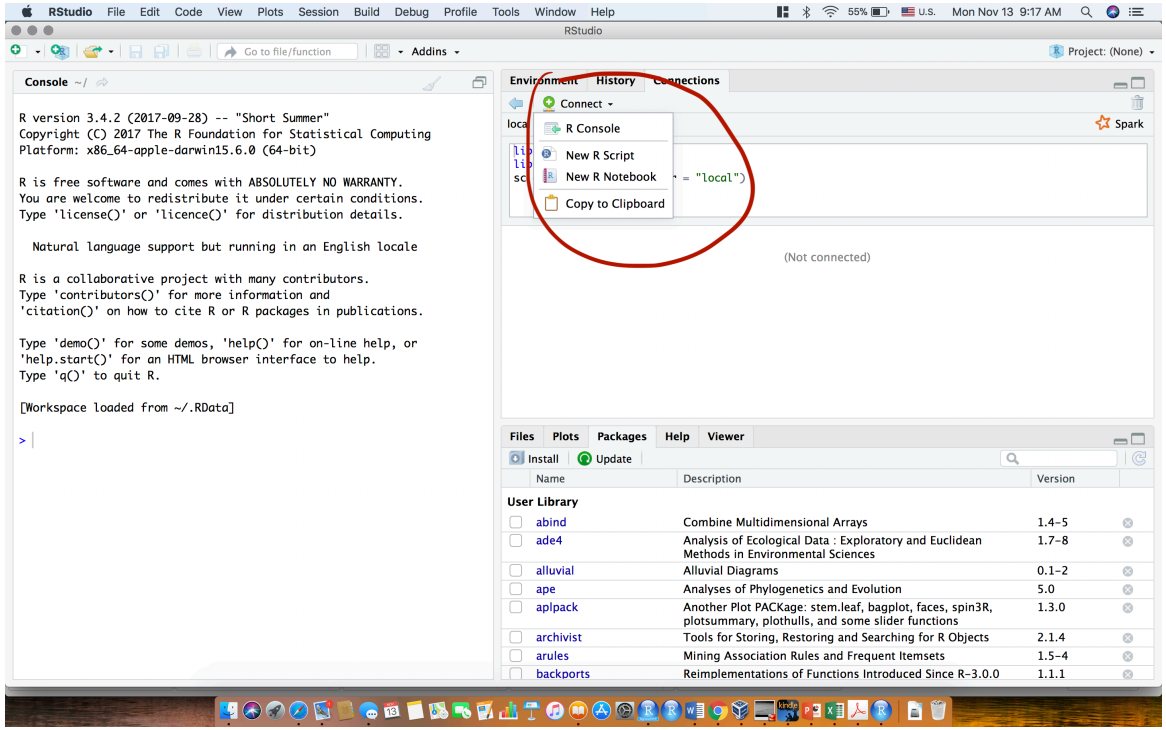
- Creating and managing Spark connections
- Browsing the tables and columns of Spark DataFrames
- Previewing the first 1,000 rows of Spark DataFrames

Connecting Spark to RStudio

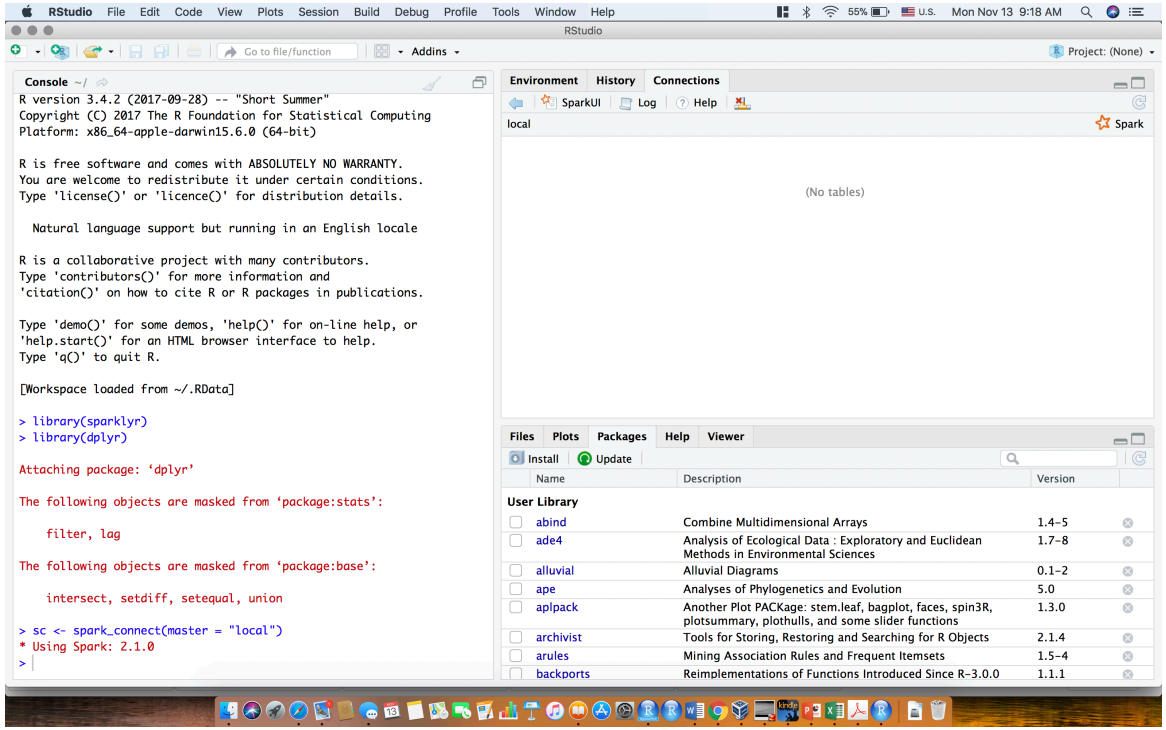
First steps: connecting Spark to RStudio



First steps: connecting Spark to RStudio



First steps: connecting Spark to RStudio



Importing a csv file into R with Spark: airline_20MM.csv

```
> ptm=proc.time()
> airline_20MM_sp <- spark_read_csv(sc, "airline_20MM",
+                                   "airline_20MM.csv")
> proc.time()-ptm
   user  system elapsed 
  0.234   0.211 129.666 
> object.size("airl20mm_sp")
104 bytes
> class(airline_20MM_sp)
[1] "tbl_spark" "tbl_sql"   "tbl_lazy"  "tbl"      
> colnames(airline_20MM_sp)
 [1] "Year"              "Month"       ...# truncated output
> dim(airline_20MM_sp)
[1] NA
```

Spark with Livy

- Livy, "An Open Source REST Service for Apache Spark (Apache License)",
- It's available in `sparklyr`
- It enables connections from the `RStudio` desktop to `Apache Spark` when Livy is available and correctly configured in the remote cluster.

Installation of Livy on your RStudio

- Assume that `SparkR`, `sparklyr` and `dplyr` packages are already installed
- First time running Livy

```
> livy_install()
* Using Spark: 2.0.1
* Using Spark home: /Users/dhafermalouche/spark/spark-2.0.1-bin-hadoop2.7
* Installing livy 0.3.0
* Downloading from:
- 'http://archive.cloudera.com/beta/livy/livy-server-0.3.0.zip'
* Installing to:
- '~/Library/Caches/livy/livy-0.3.0'
trying URL 'http://archive.cloudera.com/beta/livy/livy-server-0.3.0.zip'
Content type 'application/zip' length 95253743 bytes (90.8 MB)
==================================================
```

Starting Livy

```
> library(sparklyr)
> library(dplyr)
> livy_service_start()
17/11/19 12:38:00 INFO StateStore$: Using BlackholeStateStore for recovery.
17/11/19 12:38:00 INFO BatchSessionManager: Recovered 0 batch sessions. Next session id: 0
...# truncated output
```

Connecting `Spark`

```
> sc <- spark_connect(master = "http://localhost:8998", method = "livy")
17/11/19 12:42:23 INFO InteractiveSession$: Creating LivyClient for sessionId: 0
17/11/19 12:42:23 WARN RSCConf: Your hostname, 192.168.1.3, resolves to a loopback address, but we couldn t find any external IP address!
17/11/19 12:42:23 WARN RSCConf: Set livy.rsc.rpc.server.address if you need to bind to another address.
17/11/19 12:42:23 INFO InteractiveSessionManager: Registering new session 0
1
...# truncated output
```

Impoting a csv file

airline_20MM.csv

```
> ptm=proc.time()
> airline_20MM_sp <- spark_read_csv(sc, "airline_20MM",
+                                   "airline_20MM.csv")
> proc.time()-ptm
   user  system elapsed
  4.955   2.000 836.226
17/11/19 13:06:05 INFO ContextLauncher: 17/11/19 13:06:05 INFO SparkSqlParser: Parsing command: SHOW TABLES
17/11/19 13:06:07 INFO ContextLauncher: 17/11/19 13:06:07 INFO CodeGenerator: Code generated in 94.82887 ms
... # truncated output
```

Example: back to Netflix data

Objectif: Creating one dataset using `ff` objects

- Separate between ID movies and create one column with it
- Creat three columns Netflix user ID, Rate and Date

Creating one function with arguments `ff` object

```
library(plyr)
TransforData<-function(d){
da=as.character(d[,1])
i=grep(":",da) ## Checking the rows where's the ID of the movies
dList=vector("list",length(i))
il=c(i,length(da))
names(dList)=da[i]
for(j in 1:(length(il)-1)){
  Sys.sleep(.1) # A counter for the loop
    cat("\r", j, "of", (length(il)-1))
    flush.console()
  x=sapply(da[(il[j]+1):(il[j+1]-1)],
           function(x)strsplit(x = x,split = ","))
  if(j==(length(il)-1)) x=sapply(da[(il[j]+1):il[j+1]],
                                 function(x)strsplit(x = x,split = ","))
  x=unname(x)
  x=ldply(x)
  dList[[j]]=x
}
dData=ldply(dList)
return(dData)
}
```

Transforming the importing datasets and rbinding them

```
> ptm=proc.time()
> d1Data=TransforData(d1)
 4499 of 4499
> proc.time()-ptm
    user   system  elapsed
 657.262   25.802 1149.297
> d2Data=TransforData(d3)
 4157 of 4157
> d3Data=TransforData(d3)
 4157 of 4157
> d4Data=TransforData(d4)
 4403 of 4403
> dList=list(d1Data,d2Data,d3Data,d4Data)
> data_net=do.call(rbind,dList) ## One dataset
> dim(data_net)
[1] 100480507         4
> format(object.size(data_net),"Gb")
[1] "2.9 Gb"
> write.csv(data_net,"data_all_netflix.csv")
```

Exporting the data to Spark

- From csv to Spark

```
> data_net_tbl <- copy_to(sc, data_net,name = "data_net")
> colnames(data_net_tbl)
[1] "Movie_ID" "User_ID"  "Rate"     "Date"
```

- From Spark to csv

```
> spark_write_csv(data_net_tbl, "netflix_all_data.csv")
> file.exists("netflix_all_data.csv")
[1] TRUE
```

Manipulating data: dplyr package

How many views for each movie ?

```
> freq_movies<-data_net_tbl %>%
+   group_by(Movie_ID) %>%
+   summarize(count = n())
> freq_movies
# Source: lazy query [?? x 2]
# Database: spark_connection
   Movie_ID count
   <chr>    <dbl>
 1 12611:     319
 2 12629:     356
 3 12640:     709
 4 12642:     329
 5 12643:     323
 6 12648:    1996
 7 12663:    1926
 8 12667:   10819
 9 12674:     767
10 12677:    5617
# ... with more rows
> object.size(freq_movies)
11672 bytes
> object.size(data_net_tbl)
8032 bytes
```

Comparing sizes and execution time.

- If we transform the `tbl` to a `data.frame` object

```
> freq_movies_d=as.data.frame(freq_movies)
> object.size(freq_movies_d)
1138064 bytes
```

- Sending data.frame to Spark

```
> freq_movies_tbl<-copy_to(sc,freq_movies_d,"freq_movies")
> object.size(freq_movies_tbl)
7920 bytes
```

Comparing sizes and execution time.

`tbl` object

```
> ptm <- proc.time()
> arrange(freq_movies,desc(count))
# Source: lazy query [?? x 2]
# Database: spark_connection
# Ordered by: desc(count)
   Movie_ID  count
   <chr>     <dbl>
 1 5317:    232944
 2 15124:   216596
 3 14313:   200832
 4 15205:   196397
 5 1905:    193941
 6 6287:    193295
 7 11283:   181508
 8 16377:   181426
 9 16242:   178068
10 12470:   177556
# ... with more rows
> proc.time() - ptm
   user  system elapsed
  0.062   0.012   5.632
```

Comparing sizes and execution time.

`tbl` object with Spark

```
> ptm <- proc.time()
> arrange(freq_movies_tbl,desc(count))
# Source: table<freq_movies> [?? x 2]
# Database: spark_connection
# Ordered by: desc(count)
   Movie_ID  count
   <chr>     <dbl>
 1 5317:    232944
 2 15124:   216596
 3 14313:   200832
 4 15205:   196397
 5 1905:    193941
 6 6287:    193295
 7 11283:   181508
 8 16377:   181426
 9 16242:   178068
10 12470:   177556
# ... with more rows
> proc.time() - ptm
   user  system elapsed
  0.062   0.003   0.111
```

Daily views

- Compute the number of daily views

```
date_movies<-data_net_tbl %>%
  group_by(Date) %>%
  summarize(count = n())%>%collect
```

- *Notice:* `collect` makes `data_movies` as a `tribble` object

```
> date_movies
# A tibble: 2,182 x 2
   Date                count
   <dttm>              <dbl>
 1 2005-07-13 22:00:00 172930
 2 2004-04-22 23:00:00  71144
 3 2003-11-06 23:00:00  38261
 4 2005-07-24 22:00:00 209331
 5 2005-08-07 22:00:00 208867
 6 2005-08-14 22:00:00 216861
 7 2004-05-25 23:00:00  91295
 8 2004-08-10 23:00:00  92593
 9 2003-09-16 23:00:00  39707
10 2004-01-05 23:00:00  74197
# ... with 2,172 more rows
```

Daily views

- Constructing the time series

```
> library(dygraphs)
> library(xts)
> timeserie <-xts(date_movies, order.by = date_movies$Date)
```

- Drawing the time series *daily-views*

```
> dygraph(timeserie) %>%
+   dyRangeSelector() %>%
+   dyHighlight(highlightCircleSize = 4,
+               highlightSeriesBackgroundAlpha = 0.5,
+               hideOnMouseOut = TRUE)%>%
+   dyLegend(show = "follow")
```

Daily views

Daily views

H_2O, `rsparkling` and `dplyr`

- H_2O is an open source, in-memory, distributed, fast, and scalable machine learning and predictive analytics platform that allows you to build machine learning models on big data.
- `rsparkling` provides bindings to H2O's distributed machine learning algorithms via sparklyr.
- `dplyr` is an `R` package for working with structured data both in and outside of `R`. It makes data manipulation for R users easy, consistent, and performant.

Example: Starting H_2O from R

1. Installing spark, hadoop, sparkyingr package

```
> spark_install(version = "2.2.0", hadoop_version = "2.6")
> library(devtools)
> devtools::install_github("h2oai/rsparkling", ref = "master")
```

2. Checking if the you have installed the last version

```
> library(sparklyr)
> spark_available_versions()
```

Example: Starting H_2O from R

3. Installing `h2o` package

```
> install.packages("h2o", type = "source",
+                  repos =  "https://h2o-release.s3.amazonaws.com/h2o/rel-weierstrass/2/R")
```

Example: Starting H_2O from R

4. Connecting so Spark

```
> library(rsparkling)
> library(SparkR)
> library(dplyr)
> sc <- spark_connect(master = "local", version = "2.2.0")
```

Example: Starting H_2O from R

4. Loading and initilizing H_2O (on local machine)

```
> library(h2o)
> h2o.init()

H2O is not running yet, starting it now...
...
Starting H2O JVM and connecting: .. Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         3 seconds 766 milliseconds
    H2O cluster version:        3.14.0.2
    H2O cluster version age:    3 months and 1 day
    H2O cluster name:           H2O_started_from_R_dhafermalouche_vqu557
    H2O cluster total nodes:    1
    H2O cluster total memory:   3.56 GB
    H2O cluster total cores:    4
    H2O cluster allowed cores:  4
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
    R Version:                  R version 3.4.2 (2017-09-28)
```

Example: Airline data from `flight.csv` file (~146Mb)

1. Importing the data with `h2o`

```
> airlines.hex = h2o.importFile(path = "flight.csv",destination_frame = "airlines.hex")
  |======================================================================| 100%
```

2. Checking `h2o` objects and `h2o` environnement

```
>  h2o.ls()
                                        key
1                               airlines.hex
2 frame_rdd_26_810a7a3a9758692195f6b06a3c39474f
```

Example: Airline data from `flight.csv` file (~146Mb)

3. Getting information on the h2o cluster

```
> h2o.clusterInfo()
R is connected to the H2O cluster:
    H2O cluster uptime:         1 hours 15 minutes
    H2O cluster version:        3.14.0.7
    H2O cluster version age:    1 month and 2 days
    H2O cluster name:           sparkling-water-dhafermalouche_local-1511412255091
    H2O cluster total nodes:    1
    H2O cluster total memory:   0.65 GB
    H2O cluster total cores:    4
    H2O cluster allowed cores:  4
    H2O cluster healthy:        TRUE
    H2O Connection ip:          127.0.0.1
    H2O Connection port:        54323
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    H2O API Extensions:         Algos, XGBoost, AutoML, Core V3, Core V4
    R Version:                  R version 3.4.2 (2017-09-28)
```

Example: Airline data from `flight.csv` file (~146Mb), Data Manipulation

- Colnames of the `h2o` dataframe

```
> colnames(airlines.hex)
 [1] "YEAR"              "MONTH"             "DAY_OF_MONTH"
 [4] "DAY_OF_WEEK"       "FL_DATE"           "UNIQUE_CARRIER"
 [7] "AIRLINE_ID"        "TAIL_NUM"          "FL_NUM"
[10] "ORIGIN_AIRPORT_ID" "ORIGIN"            "ORIGIN_CITY_NAME"
...
```

Example: Airline data from `flight.csv` file (~146Mb), Data Manipulation

We compute by the origine city name

- number of flights
- average and sd of the arrival delays

```
> originFlights = h2o.group_by(data = airlines.hex,
+                              by ="ORIGIN_CITY_NAME",
+                              nrow("ORIGIN_CITY_NAME"),
+                              mean("ARR_DELAY"),
+                              sd("ARR_DELAY"),
+                              gb.control=list(na.methods="rm"))
> originFlights
  ORIGIN_CITY_NAME nrow mean_ARR_DELAY sdev_ARR_DELAY
1     Aberdeen, SD  131      0.1923077       23.73664
2      Abilene, TX  394     -5.1662338       43.77093
3  Adak Island, AK   17     23.0588235       24.34510
4    Aguadilla, PR  191      2.5026178       29.96944
5        Akron, OH  955     -2.8164557       34.38834
6       Albany, GA  165      8.2865854       42.59839
[305 rows x 4 columns]
> originFlightsR=as.data.frame(originFlights)
```

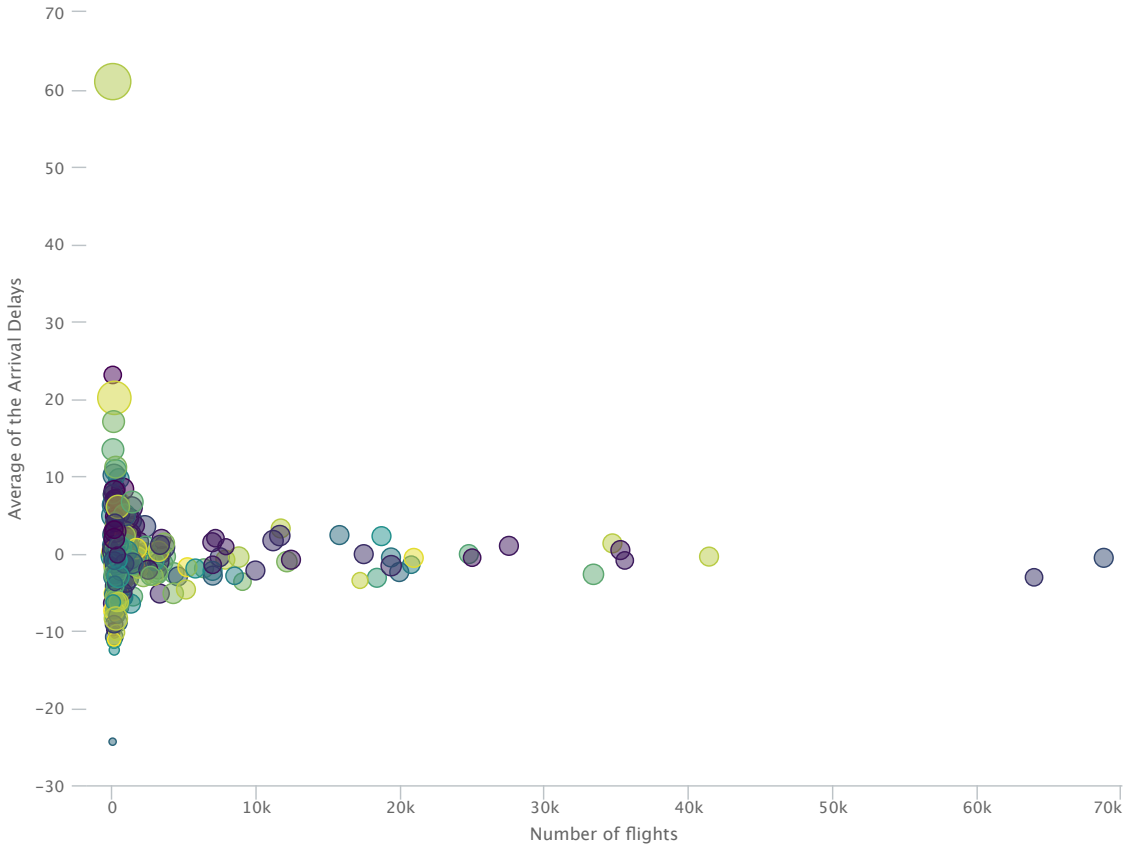
Example: Airline data from `flight.csv` file (~146Mb), Data Manipulation

- create a state and city variable

```
> originFlightsR=as.data.frame(originFlights)
> x=strsplit(x = as.character(originFlightsR$ORIGIN_CITY_NAME),split=",")
> originFlightsR$Depart_city=unlist(lapply(x,function(x)x[1]))
> originFlightsR$Depart_state=unlist(lapply(x,function(x)x[2]))
> head(originFlightsR,3)
  ORIGIN_CITY_NAME nrow mean_ARR_DELAY sdev_ARR_DELAY Depart_city Depart_state
1     Aberdeen, SD  131      0.1923077       23.73664    Aberdeen           SD
2      Abilene, TX  394     -5.1662338       43.77093     Abilene           TX
3  Adak Island, AK   17     23.0588235       24.34510 Adak Island           AK
```

Example: Airline data from `flight.csv` file (~146Mb), Scatter plot Number of flights x Arrival delays.

Example: Airline data from `flight.csv` file (~146Mb), Scatter plot Number of flights x Arrival delays.



Construct test and train sets using sampling

```
> airlines.split = h2o.splitFrame(data = airlines.hex,
+                                 ratios = 0.85)
> airlines.train = airlines.split[[1]]
> dim(airlines.train)
[1] 808709     28
> airlines.test = airlines.split[[2]]
> dim(airlines.test)
[1] 142402     28
```

A glm model with h2o

1. Creatung a binary variable Detecting delays?

```
> x=(airlines.hex$DEP_DELAY>10)
> h2o.table(x)
  DEP_DELAY  Count
1         0 808040
2         1 143071
[2 rows x 2 columns]
> airlines.hex$IsDepDelayed=as.factor(x)
```

A glm model with h2o

2. Setting up the training and test data.

```
> airlines.split = h2o.splitFrame(data = airlines.hex,
+                                 ratios = 0.85)
> airlines.train = airlines.split[[1]]
> dim(airlines.train)
[1] 808466     29
> airlines.test = airlines.split[[2]]
> dim(airlines.test)
[1] 142645     29
```

A glm model with h2o

3. Implementing the model

```
> ptm=proc.time()
> airlines.glm <- h2o.glm(training_frame=airlines.train,
+                         x=X, y="IsDepDelayed", family = "binomial", alpha = 0.5)
  |======================================================================| 100%
> proc.time()-ptm
   user  system elapsed
  0.332   0.020   3.092
```

A glm model with h2o

4. Summary

```
> summary(airlines.glm)
Model Details:
==============

H2OBinomialModel: glm
Model Key:  GLM_model_R_1511412268657_8
GLM Model: summary
    family  link                               regularization number_of_predictors_total
1 binomial logit Elastic Net (alpha = 0.5, lambda = 2.728E-5 )                        929
  number_of_active_predictors number_of_iterations    training_frame
1                         266                    5 RTMP_sid_95de_268

H2OBinomialMetrics: glm
** Reported on training data. **

MSE:  0.1263806
RMSE:  0.3555004
LogLoss:  0.4180576
Mean Per-Class Error:  0.445333
AUC:  0.5796161
Gini:  0.1592322
R^2:  0.01014465
Residual Deviance:  675970.7
AIC:  676504.7

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
            0      1    Error            Rate
0      303804 383189 0.557777  =383189/686993
1       40437  81036 0.332889   =40437/121473
Totals 344241 464225 0.523987  =423626/808466

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold    value idx
1                        max f1  0.141048 0.276716 251
2                        max f2  0.104030 0.471475 346
3                  max f0point5  0.171942 0.216035 167
4                  max accuracy  0.331574 0.849746   0
5                 max precision  0.280362 0.303178  12
6                    max recall  0.061425 1.000000 399
7                max specificity  0.331574 0.999997   0
```

```
8             max absolute_mcc  0.145412 0.080456 239
9   max min_per_class_accuracy  0.150421 0.554866 225
10 max mean_per_class_accuracy  0.145412 0.556243 239

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`



Scoring History:
            timestamp   duration iterations negative_log_likelihood objective
1 2017-11-23 20:17:33  0.000 sec          0             342098.02286   0.42314
2 2017-11-23 20:17:34  0.224 sec          1             338105.95616   0.41859
3 2017-11-23 20:17:34  0.331 sec          2             338034.34112   0.41852
4 2017-11-23 20:17:34  0.439 sec          3             338034.56574   0.41852
5 2017-11-23 20:17:34  0.705 sec          4             337985.14489   0.41847
6 2017-11-23 20:17:34  0.814 sec          5             337985.33733   0.41847

Variable Importances: (Extract with `h2o.varimp`)
================================================

Standardized Coefficient Magnitudes: standardized coefficient magnitudes
       names coefficients sign
1 ORIGIN.BWI     0.623784  POS
2 ORIGIN.DAL     0.461662  POS
3 ORIGIN.HPN     0.452875  POS
4 ORIGIN.FLL     0.413892  POS
5 ORIGIN.HOU     0.399226  POS

---
                          names coefficients sign
924  DEST_CITY_NAME.Williston, ND     0.000000  POS
925 DEST_CITY_NAME.Wilmington, NC     0.000000  POS
926  DEST_CITY_NAME.Worcester, MA     0.000000  POS
927   DEST_CITY_NAME.Wrangell, AK     0.000000  POS
928    DEST_CITY_NAME.Yakutat, AK     0.000000  POS
929       DEST_CITY_NAME.Yuma, AZ     0.000000  POS
```

A glm model with h2o

5. Prediction

```
> ptm=proc.time()
> pred = h2o.predict(object = airlines.glm, newdata =
+                      airlines.test)
  |=============================================================| 100%
> proc.time()-ptm
   user  system elapsed
  0.064   0.008   1.129
```

A glm model with h2o

6. Displaying the result of the prediction

```
> perf<-h2o.performance(airlines.glm,airlines.test$IsDepDelayed)
> perf
H2OBinomialMetrics: glm

MSE:  0.1290872
RMSE:  0.3592871
LogLoss:  0.4277275
Mean Per-Class Error:  0.5
AUC:  0.5
Gini:  0
R^2:  -0.004682275
Residual Deviance:  122026.4
AIC:  122560.4

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
       0      1    Error            Rate
0      0 121047 1.000000  =121047/121047
1      0  21598 0.000000        =0/21598
Totals 0 142645 0.848589  =121047/142645

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold    value idx
1                       max f1  0.126883 0.263001   0
2                       max f2  0.126883 0.471496   0
3                 max f0point5  0.126883 0.182361   0
4                 max accuracy  0.126883 0.151411   0
5                max precision  0.126883 0.151411   0
6                   max recall  0.126883 1.000000   0
7              max specificity  0.126883 0.000000   0
8             max absolute_mcc  0.126883 0.000000   0
9   max min_per_class_accuracy  0.126883 0.000000   0
10 max mean_per_class_accuracy  0.126883 0.500000   0
```

Back to Netflix data on H_2O.

- Intilializing H_2O again (we set 5Gb as needed memory)

```
> library(rsparkling)
> sc <- spark_connect(master = "local", version = "2.2.0")
Spark version 2.2 detected. Will call latest Sparkling Water version 2.2.2
> library(h2o)
> h2o.init(max_mem_size = "5g")
H2O is not running yet, starting it now...

Note:  In case of errors look at the following log files:
...
R is connected to the H2O cluster:
    H2O cluster uptime:         4 seconds 75 milliseconds
    H2O cluster version:        3.14.0.2
    H2O cluster version age:    3 months and 3 days
    H2O cluster name:           H2O_started_from_R_dhafermalouche_mwl200
    H2O cluster total nodes:    1
    H2O cluster total memory:   4.44 GB
    H2O cluster total cores:    4
    H2O cluster allowed cores:  4
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
    R Version:                  R version 3.4.2 (2017-09-28)
```

Back to Netflix data on H_2O.

- Importing the data into H_2O: we use the file already created with Spark (see this slide)

```
> netflix.hex = h2o.importFile(path = "netflix_all_data.csv",
+                              destination_frame = "netflix.hex")
  |======================================================================| 100%
> h2o.ls()
         key
1 netflix.hex
> h2o.clusterInfo()
R is connected to the H2O cluster:
    H2O cluster uptime:         7 minutes 23 seconds
    H2O cluster version:        3.14.0.2
    H2O cluster version age:    3 months and 3 days
    H2O cluster name:           H2O_started_from_R_dhafermalouche_mwl200
    H2O cluster total nodes:    1
    H2O cluster total memory:   2.96 GB
    H2O cluster total cores:    4
    H2O cluster allowed cores:  4
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
    R Version:                  R version 3.4.2 (2017-09-28)
```

Question: Which movie is the best rated?

- We compute frenquency Movie x Rates

```
> library(rsparkling)
> library(SparkR)
> library(dplyr)
> ptm<-proc.time()
> NetflixRateMovie= h2o.group_by(data = netflix.hex,
+                                by= c("Movie_ID","Rate"), nrow(c("Movie_ID","Rate")),
+                                gb.control=list(na.methods="rm"))
Warning message:
In if (is.na(col.idx)) stop("No column named ", col.name, " in ",  :
  the condition has length > 1 and only the first element will be used
> proc.time()-ptm
   user  system elapsed
  0.003   0.000   0.003
> NetflixRateMovie
  Movie_ID Rate nrow
1   10000:    1   21
2   10000:    2   54
3   10000:    3   86
4   10000:    4   33
5   10000:    5   21
6   10001:    1   18
[88806 rows x 3 columns]
```

Question: Which movie is the best rated?

- we construct then a contingency table Movie x Rate

```
> Netflix.dt=as.data.frame(NetflixRateMovie)
> Netflix.dt2=xtabs(nrow~Movie_ID+Rate,data=Netflix.dt)
> head(Netflix.dt2)
        Rate
Movie_ID    1    2    3    4    5
  10000:   21   54   86   33   21
  10001:   18    5   23   56   56
  10002:  349  206  703  672  694
  10003:   32   21   35   10    9
  10004:   83  150  552 1295 1941
  10005:   18   31  112   83   25
```

Question: Which movie is the best rated?

- We perform then a Correspondance Analysis and scale the coordinates of the rows to one

```
> library(FactoMineR)
> ca<-CA(X = Netflix.dt2a,graph = F)
> dt=cbind(ca$row$coord[,1:2])
> colnames(dt)=c("Axis1","Axis2")
> dt=as.data.frame(dt)
> dt$Axis1=dt$Axis1/sqrt(ca$eig[1,1])
> dt$Axis2=dt$Axis2/sqrt(ca$eig[2,1])
> dt$Movie_ID=rownames(dt)
```

- Same thing for the variables (Rates)

```
dtVar=ca$col$coord[,1:2]
colnames(dtVar)=c("Axis1","Axis2")
dtVar=as.data.frame(dtVar)
dtVar$Axis1=dtVar$Axis1/sqrt(ca$eig[1,1])
dtVar$Axis2=dtVar$Axis2/sqrt(ca$eig[2,1])
dtVar$Rate=rownames(dtVar)
```

Question: Which movie is the best rated?

- Classification of the movies: A Movie is in the class of Rate $i \in \{1, \dots, 5\}$ if it's close to the point of rate i:

$$i = \text{argmin}_{j=1,\dots,5} d(m, j)$$

```
> OneDist=function(iM,jR){
+   sum((dt[iM,1:2]-dtVar[jR,1:2])^2)
+ }
> WhichClass=function(iM){
+   z=unlist(sapply(1:5,function(x)OneDist(iM,x)))
+   which.min(z)
+ }
> z=sapply(1:nrow(dt),WhichClass)
> z[1:3]
[1] 2 5 2
> dt$Rate=z
```

Question: Which movie is the best rated?

- Biplot of the CA.

```
> TotViews=rowSums(Netflix.dt2a)
> dt=cbind(ca$row$coord[,1:2],TotViews,Netflix.dt2a)
> colnames(dt)=c("Axis1","Axis2","TotalViews",colnames(Netflix.dt2a))
```

Question: Which movie is the best rated?

- We add movie names (and deleting utf 8 characters)

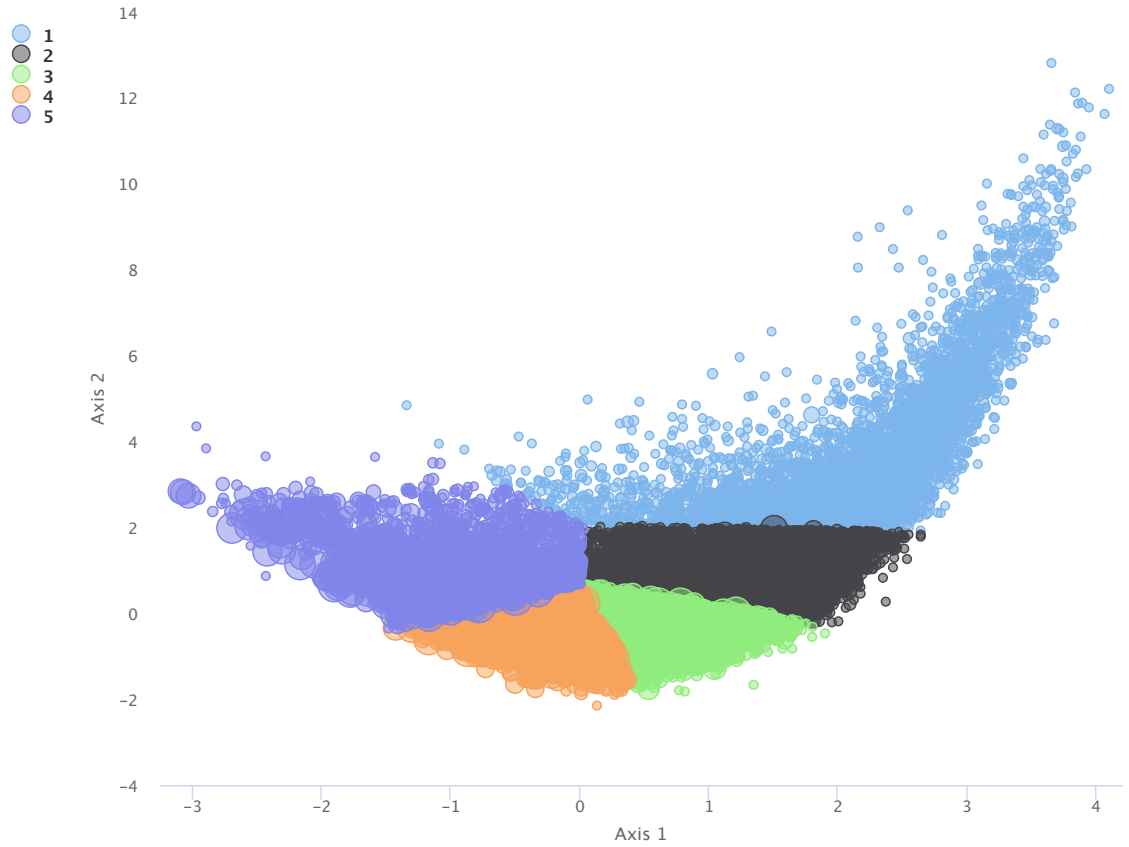
```
> library(readr)
> movie_titles <- read_csv("movie_titles.csv",
+     col_names = FALSE)
> x=gsub(pattern = ":",replacement = "",dt$Movie_ID)
> x[1:3]
[1]  10000 10001 10002
> dt$Movie_ID=x
> i=match(dt$Movie_ID,movie_titles$X1)
> dt=cbind.data.frame(dt,movie_titles[i,])
> x=dt$X3
> y=iconv(x, from = "latin1", to = "UTF-8")
> dt$X3=y
```

Question: Which movie is the best rated?

- Biplot of the CA.

```
library(magrittr)
library(highcharter)
> x <- c("Movie ","# of Views","Rate","1", "2","3","4","5")
> y1 <- sprintf("{point.%s:.0f}", c("TotalViews","Rate",
+                                   "1","2","3","4","5"))
> y0<-paste0("{point.","X3","}")
> y<-c(y0,y1)
> tltip <- tooltip_table(x, y)
> p<-hchart(dt, "scatter",
+           hcaes(Axis1, Axis2,
+                 size = TotalViews,
+                 group=Rate),
+           debug = TRUE)
|=========================================================|100% ~0 s remaining
> p <- p %>%  hc_tooltip(useHTML = TRUE, headerFormat = "",
+                        pointFormat = tltip) %>%
+   hc_size(height = 600)
> p<- p%>%  hc_chart(backgroundColor = "white") %>%
+   hc_xAxis(title=list(text="Axis 1"), gridLineWidth = 0) %>%
+   hc_yAxis(title=list(text="Axis 2"), gridLineWidth = 0)
> p <- p %>%hc_legend(align = "left", verticalAlign = "top",
+                     layout = "vertical", x = 0, y = 2.5)
> p
```

Question: Which movie is the best rated?



Other commands with H_2O

- Initializing h2o

```
> library(sparklyr)
> sc <- spark_connect(master = "local", version = "2.2.0")
> library(h2o)
> h2o.init(max_mem_size = "5g")

H2O is not running yet, starting it now...
> library(rsparkling)
```

Other commands with H_2O

- Importing the data into h2o

```
> netflix.hex = h2o.importFile(path = "netflix_all_data.csv",
+                              col.types=c("factor","factor","numeric","factor"),
+                              destination_frame = "netflix.hex")
  |======================================================================| 100%
> netflix.hex[1,4]
[1] 2003-06-18
2182 Levels: 1999-11-11 1999-12-06 1999-12-08 1999-12-09 1999-12-10 1999-12-11 1999-12-12 ... 2005-12-31
```

Other commands with H_2O

- Creating year, month and day variables

```
> x=h2o.strsplit(netflix.hex[4],split = "-")
> x1=h2o.cbind(netflix.hex,x)
> head(x1)
  Movie_ID User_ID Rate       Date   C1 C2 C3
1   15751:  949610    4 2003-06-18 2003 06 18
2   15751:  593525    4 2005-07-20 2005 07 20
3   15751: 2225344    3 2004-06-15 2004 06 15
4   15751:   10645    4 2004-03-24 2004 03 24
5   15751: 2031797    5 2005-11-04 2005 11 04
6   15751: 2133079    4 2005-11-13 2005 11 13
> netflix.hex=x1
> colnames(netflix.hex)[5:7]
[1] "C1" "C2" "C3"
> colnames(netflix.hex)[5:7]=c("year","month","day")
```

Other commands with H_2O

- Exporting the file into a csv file

```
> h2o.downloadCSV(data = netflix.hex,filename = "netflix_data_2611.csv")
cmd: curl
args: -o netflix_data_2611.csv http://localhost:54321/3/DownloadDataset?frame_id=RTMP_sid_beee_26
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 70.6M    0 14.9M    0     0  36.4M      0 --:--:-- --:--:-- --:--:-- 3100 4789M    0 4789M    0     0  53.8M      0 --:--:--  0:01:29 --:--:-- 53.2M
> file.exists("netflix_data_2611.csv")
[1] TRUE
> file.size("netflix_data_2611.csv")/1024/1024/1024
[1] 4.67717
```

Example Netflix: Which days of the week we watch the most Netflix?

- Daily views data

```
> ptm=proc.time()
> daily_views=h2o.group_by(netflix.hex,by = "Date",
+                          nrow("Date"))
> proc.time()-ptm
   user  system elapsed 
  0.003   0.001   0.003 
> daily_views
        Date nrow
1 1999-11-11   28
2 1999-12-06   12
3 1999-12-08   29
4 1999-12-09   80
5 1999-12-10   21
6 1999-12-11   28
```

Example Netflix: Which days of the week we watch the most Netflix?

- Creating week days variable

```
> daily_view.df=as.data.frame(daily_views)
> x=as.Date(daily_view.df$Date)
> x[1:10]
 [1] "1999-11-11" "1999-12-06" "1999-12-08" "1999-12-09" "1999-12-10" "1999-12-11" "1999-12-12"
 [8] "1999-12-14" "1999-12-15" "1999-12-16"
> y=weekdays(x)
> y[1:10]
 [1] "Thursday"  "Monday"    "Wednesday" "Thursday"  "Friday"    "Saturday"  "Sunday"    "Tuesday"
 [9] "Wednesday" "Thursday"
> daily_view.df$days=y
> head(daily_view.df)
        Date nrow      days
1 1999-11-11   28  Thursday
2 1999-12-06   12    Monday
3 1999-12-08   29 Wednesday
4 1999-12-09   80  Thursday
5 1999-12-10   21    Friday
6 1999-12-11   28  Saturday
```

Example Netflix: Which days of the week we watch the most Netflix?

- Using now `data.table` package

```
> daily_views2=as.data.table(daily_view.df)
> daily_views2
           Date  nrow      days
   1: 1999-11-11    28  Thursday
   2: 1999-12-06    12    Monday
   3: 1999-12-08    29 Wednesday
   4: 1999-12-09    80  Thursday
   5: 1999-12-10    21    Friday
  ---
2178: 2005-12-27 87804   Tuesday
2179: 2005-12-28 98671 Wednesday
2180: 2005-12-29 97320  Thursday
2181: 2005-12-30 77599    Friday
2182: 2005-12-31 45670  Saturday
```

Example Netflix: Which days of the week we watch the most Netflix?

- Using now `data.table` package

```
> d1=aggregate(nrow~ days,sum,data = daily_views2)
> d1
       days     nrow
1    Friday 13404388
2    Monday 17318845
3  Saturday 10027687
4    Sunday 10730350
5  Thursday 14476074
6   Tuesday 17784852
7 Wednesday 16738311
```

Example Netflix: Which days of the week we watch the most Netflix?

- Correcting the order of the days

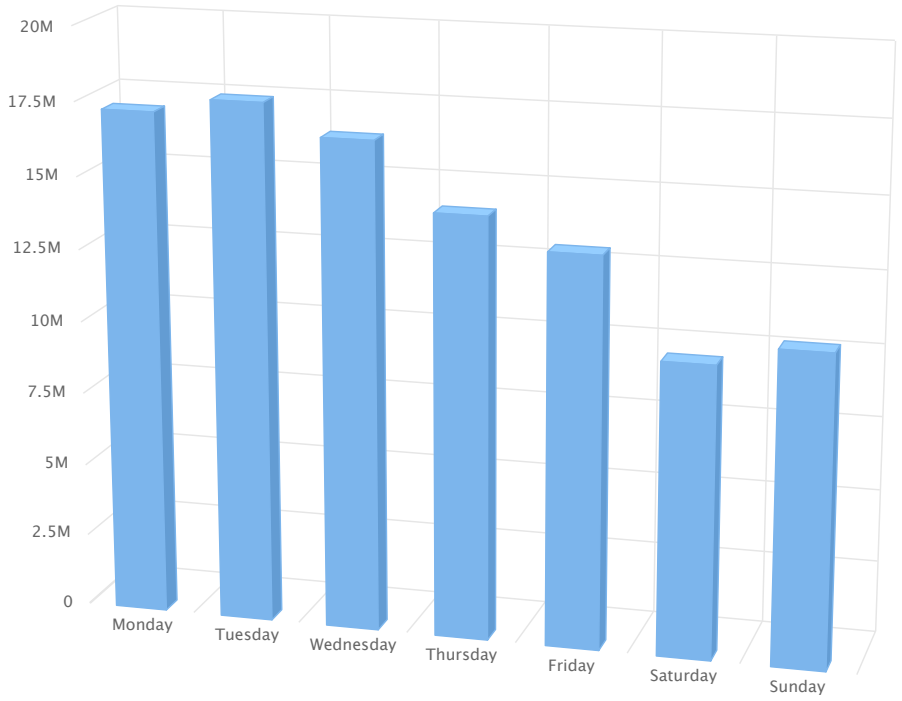
```
> wd=weekdays(seq.Date(as.Date("2017/11/27"),as.Date("2017/12/3"),"days"))
[1] "Monday"    "Tuesday"   "Wednesday" "Thursday"  "Friday"    "Saturday"  "Sunday"
> i=match(wd,d1$days)
> i
[1] 2 6 7 5 1 3 4
> d1=d1[i,]
> d1
       days     nrow
2    Monday 17318845
6   Tuesday 17784852
7 Wednesday 16738311
5  Thursday 14476074
1    Friday 13404388
3  Saturday 10027687
4    Sunday 10730350
```

Example Netflix: Which days of the week we watch the most Netflix?

- The figure

```
> library(highcharter)
> library(magrittr)
> hc <- highchart() %>%
+   hc_xAxis(categories = d1$days) %>%
+   hc_add_series(name = "nrow", data = d1$nrow) %>%
+   hc_chart(type = "column",
+            options3d = list(enabled = TRUE, beta = 15, alpha = 15))
> hc%>%hc_legend()
```

Example Netflix: Which days of the week we watch the most Netflix?



Comparing Machine Learning tools H_2O and `dplyr`

Importing `airline_20MM.csv` file

- with `h2o`

```
> airline.h2o= h2o.importFile(path = "airline_20MM.csv",
+                         destination_frame = "airline.h2o")
  |======================================================================| 100%
```

- with `dplyr`

```
> airline.tbl<-spark_read_csv(sc, "airline","airline_20MM.csv")
```

Comparing Machine Learning tools H_2O and `dplyr`

Setting training and test partitions

- with `h2o`

```
> ptm<-proc.time()
> airlines.split.h2o = h2o.splitFrame(data = airline.hex,
+                                     ratios = 0.66)
> proc.time()-ptm
   user  system elapsed
  0.003   0.002   0.006
> airlines.h2o.train = airlines.split.h2o[[1]]
> airlines.h2o.test = airlines.split.h2o[[2]]
> airline.h2o=airline.hex
> ptm<-proc.time()
> nrow(airlines.h2o.train)
[1] 13200128
> proc.time()-ptm
   user  system elapsed
  0.001   0.000   0.002
```

Comparing Machine Learning tools H_2O and `dplyr`

Setting training and test partitions

- with `dplyr`

```
> ptm<-proc.time()
> partitions.tbl <- airline.tbl %>%
+   sdf_partition(training = 0.665, test = 0.34, seed = 1099)
> proc.time()-ptm
   user  system elapsed 
  0.022   0.004   0.238 
> ptm<-proc.time()
> partitions.tbl$training%>%summarise(count=n())
# Source: lazy query [?? x 1]
# Database: spark_connection
     count
     <dbl>
1 13235735
> proc.time()-ptm
   user  system elapsed 
  0.064   0.045  28.643 
```

Performing a Linear regression model

`ArrDelay ~ DepDelay + Distance`

- with `dplyr`

```
> fit <- partitions.tbl$training %>%
+   ml_linear_regression(response = "ArrDelay", features = c("Distance", "DepDelay"))
> proc.time()-ptm
   user  system elapsed
  0.805   0.556 268.195
```

Performing a Linear regression model

`ArrDelay ~ DepDelay + Distance`

- with `h2o`

```
> ptm<-proc.time()
> fit.h20 <- h2o.glm(x= c("Distance", "DepDelay"),
+                    y = "ArrDelay",
+                    training_frame = airlines.h2o.train,
+                    model_id = "lm.h2o",
+                    alpha=0.5,
+                    family="gaussian")
  |=================================================| 100%
> proc.time()-ptm
   user  system elapsed 
  0.284   0.016   3.658 
```