Tutorial - Big Data Analyses with R
O'Reilly Strata Conference London

Dr. rer. nat. Markus Schmidberger

@cloudHPC
markus.schmidberger@comsysto.com

November 13th, 2013
Dr. rer. nat. Markus Schmidberger

High Performance Computing
mongoDB
Parallel Computing
AWS
NoSQL
Data Quality Management
R
Hadoop
MPI
Big Data
comSysto GmbH
Snowboarding
PhD
TU Munich
Map Reduce
Cloud Computing

husband
Big Data
Hadoop
Bavaria
comSysto GmbH
R
Data Science
Biological Data
Snowboarding
NoSQL
DevOps
Munich
Parallel Computing
Biocomputing
Biological Data
Devops
Cloud Computing
Biocomputing
Biological Data
Devops
Cloud Computing

M. Schmidberger
Tutorial - Big Data Analyses with R
November 13th, 2013

2 / 119
comSysto GmbH

- based in Munich, Germany
- Lean Company. Great Chances!
- **software company** specialized in lean business, technology development and Big Data
- focuses on **open source frameworks**
- **Meetup organizer** for Munich
  - http://www.meetup.com/munich-useR-group/
- http://www.comsysto.com
Project: Telefónica Dynamic Insights 'SmartSteps'

"Big decisions made Better"

http://dynamicinsights.telefonica.com/488/smart-steps
Motivation and Goals

- Today, there exists a lot of data and a huge pool of analyses methods
- **R** is a great tool for your Big Data analyses

- Provide an **overview** of Big Data technologies in **R**
- **Hands-on code** and exercises to get started
Outline

1. Big Data
2. R Intro and R Big Data Packages
3. R and Databases
4. R and Hadoop
Outline

1 Big Data

2 R Intro and R Big Data Packages

3 R and Databases

4 R and Hadoop
Big Data

- a big hype topic
- everything is big data
- everyone wants to work with big data
- Wikipedia: "... a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications ..."
Big Data
my view on data

- access to many different data sources (Internet)
- storage is cheap - store everything
- today, CIOs get interested in the power of all their data
- ⇒ a lot of different and complex data have to be stored
- ⇒ NoSQL
NoSQL - MongoDB

- NoSQL: databases with less constrained consistency models ⇒ schema-less
- **MongoDB**:
  - open source, cross-platform document-oriented database system
  - most popular NoSQL database system
  - supported MongoDB Inc.
  - stores structured data as JSON-like documents with dynamic schemas

**MongoDB as a German / European Service**

http://www.mongodb.org  http://www.mongosoup.de
more and more data have to be processed
backward-looking analysis is outdated
today, we are working on quasi real-time analysis
but, CIOs are interested in **forward-looking predictive analysis**
⇒ more **complex methods**, more processing time
⇒ **Hadoop**, Super Computer and statistical tools
Hadoop

- open-source software framework designed to support large scale data processing
- **Map Reduce**: a computational paradigm
  - application is divided into many small fragments of work
- **HDFS**: Hadoop Distributed File System
  - a distributed file system that stores data on the compute nodes
- the Ecosystem: Hive, Pig, Flume, Mahout, ...
- written in Java, opened up to alternatives by its Streaming API

![Hadoop Logo](http://hadoop.apache.org)
Big Data
my view on resources

- computing resources are available to everyone and cheap
- man-power is expensive and it is difficult to hire Big Data experts
  - Java Programmer: good programming - bad statistical background
  - Statistician: good methodology - bad programming and database knowledge
  - Big Data Analyst: ?? - skills shortage or 'Fachkräfte mangels'

⇒ welcome to the 'Tutorial - R and Big Data' to solve this problem
Outline

1. Big Data

2. **R** Intro and **R** Big Data Packages
   - www.r-project.org
   - **R** as Calculator
   - **R** Packages for Big Data

3. **R** and Databases

4. **R** and Hadoop
open-source: **R** is a free software environment for statistical computing and graphics

- offers tools to **manage and analyze data**
- standard and many more **statistical methods** are implemented
- support via the **R** mailing list by members of the core team
  - **R-announce**, **R-packages**, **R-help**, **R-devel**, ...
  - [http://www.r-project.org/mail.html](http://www.r-project.org/mail.html)
- support via several manuals and books:
  - [http://www.r-project.org/doc/bib/R-books.html](http://www.r-project.org/doc/bib/R-books.html)
  - [http://cran.r-project.org/manuals.html](http://cran.r-project.org/manuals.html)
• huge online-libraries with \texttt{R}-packages:
  ▶ CRAN: http://cran.r-project.org/
  ▶ BioConductor (genomic data): http://www.bioconductor.org/
  ▶ Omegahat: http://www.omegahat.org/
  ▶ \texttt{R}-Forge: http://r-forge.r-project.org/

• possibility to \textbf{write personalized code} and to contribute new packages

• really famous since January 6, 2009: The New York Times, ”Data Analysts Captivated by \texttt{R}’s Power”
R vs. SAS vs. Julia vs. Python vs. ...

R is open source, SAS is a commercial product, Julia a very new dynamic programming language, ...

- R is free and available to everyone
- R code is open source and can be modified by everyone
- R is a complete and enclosed programming language
- R has a **big and active community**
R books

(a)

(b)
RStudio IDE
http://www.rstudio.com
# Home Prices In Mid-West

```r
homes <- read.csv("homePriceData.csv")
View(homes)
names(homes)
summary(homes$price)
summary(homes$age)

states <- levels(homes$state)
avePrice <- round(mean(homes$price), 2)
aveAge <- round(mean(homes$age), 0)
```

## Subsetting Vectors, Matrices and Data Frames

### Description
Return subsets of vectors, matrices or data frames which meet conditions.

### Usage
```r
subset(x, ...)```

### Arguments
- `x`: the object from which subsets are wanted.
- `...`: conditions which must all be satisfied. (E.g., `subset(homes, states == "CA")`.)

### Example
```r
> subset(homes, states == "CA")
```
```r
# Black Scholes
# Option Pricing Model
# s = current stock price
# x = strike price
# r = risk free rate
# sigma = volatility
# t = time to expiration
# t.exp = expiration time

# price of call option
callprice.bs <- function (s, x, r, sigma, t.exp, t) {
  d.pos <- log(s/x) + (r + 0.5 * sigma^2) * (t.exp - t)
  d.neg <- d.pos + sigma * (t.exp - t)/0.5
  d.pos <- d.pos + sigma * (t.exp - t)/0.5
  s * pnorm(d.pos) - x * exp(- r * (t.exp - t)) * pnorm(d.neg)
}

# price of put option
p <- callprice.bs(s, x, r, sigma, t.exp, t)
```
R as Calculator

> (5+5) - 1 * 3  
[1]  7

> abs(-5)  
[1]  5

> x <- 3  
> x  
[1]  3

> x^2 + 4  
[1]  13
> help("mean")
> ?mean

> x <- 1:10
> x

[1]  1  2  3  4  5  6  7  8  9 10

> x < 5

[1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE

> x[3:7]

[1]  3  4  5  6  7

> x[-2]

[1]  1  3  4  5  6  7  8  9 10
load packages and data

```r
> # install.packages("onion")
> library(onion)
> data(bunny)
> head(bunny, n=3)

      x       y       z
[1,] -0.0378297 0.127940 0.00447467
[2,] -0.0447794 0.128887 0.00190497
[3,] -0.0680095 0.151244 0.03719530

> p3d(bunny, theta=3, phi=104, box=FALSE)
```
functions

> myfun <- function(x,y){
+   res <- x+y^2
+   return(res)
+ }
> myfun(2,3)
[1] 11

> myfun

function(x,y){
  res <- x+y^2
  return(res)
}
many statistical functions, e.g. k-means clustering

```r
> x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
+            matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))
> head(x, n=3)
   [,1]       [,2]
[1,] -0.15097039 -0.3043468
[2,]  0.01462622 -0.2168538
[3,]  0.14398056  0.3953751
> dim(x)
[1] 100  2
> cl <- kmeans(x, 4)
> cl
K-means clustering with 4 clusters of sizes 16, 49, 15, 20

Cluster means:
   [,1]       [,2]
1  0.54676699  0.98002268
2 -0.01015649 -0.03316055
3  1.16303315  0.71761137
4  1.06528514  1.24213166
```

M. Schmidberger
Tutorial - Big Data Analyses with R
November 13th, 2013 27 / 119
> plot(x, col = cl$cluster)
> points(cl$centers, col = 1:4, pch = 8, cex = 2)
data types

**vector**: all elements same data type

```R
> vector <- c(1,2,5.3,6,-2,4)
> vector
[1] 1.0 2.0 5.3 6.0 -2.0 4.0
```

**matrix**: all elements same data type

```R
> matrix <- matrix(LETTERS[c(19,20,18,1,20,1)],
+                  nrow=2, ncol=3)
> matrix
      [,1] [,2] [,3]
[1,]   "S"  "R"  "T"
[2,]   "T"  "A"  "A"
```
Data frame: different columns can have different data types

```r
> vector2 <- c("red", "white", "red", NA, "blue", "orange")
> vector3 <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)
> dataframe <- data.frame(vector, vector2, vector3)
> dataframe

   vector vector2 vector3
1  1.0    red    TRUE
2  2.0  white    TRUE
3  5.3    red    TRUE
4  6.0     <NA>   FALSE
5 -2.0    blue    TRUE
6  4.0  orange   FALSE

> typeof(dataframe)

[1] "list"
```
**list**: ordered collection of elements

```r
> mylist <- list(name="Fred", numbers=vector,
+ matrix=matrix, age=5.3)
> mylist

$name
[1] "Fred"

$numbers
[1] 1.0 2.0 5.3 6.0 -2.0 4.0

$matrix
  [,1] [,2] [,3]
[1,] "S"  "R"  "T"
[2,] "T"  "A"  "A"

$age
[1] 5.3
```
Package: **data.table**

- **extension of data.frame** for fast indexing, fast ordered joins, fast assignment, fast grouping and list columns

```r
> library(data.table)
> datatable <- data.table(vector, vector2, vector3)
> datatable

       vector vector2 vector3
1:     1.0     red    TRUE
2:     2.0   white    TRUE
3:     5.3     red    TRUE
4:     6.0      NA   FALSE
5:    -2.0    blue    TRUE
6:     4.0   orange   FALSE
```
> datatable[2]
  vector vector2 vector3
1: 2 white TRUE

> datatable[,vector]
[1] 1.0 2.0 5.3 6.0 -2.0 4.0

> datatable[,sum(vector),by=vector3]
  vector3 V1
1: TRUE 6.3
2: FALSE 10.0

> setkey(datatable,vector2)
> datatable["orange"]
  vector2 vector vector3
1: orange 4 FALSE
Package: plyr

- **tools** for splitting, applying and combining data
- functions are named according to what sort of data structure used (a, l, d, m)
- provides a set of **helper** functions for common **data analysis**

```r
> library(plyr)
> data(iris)
> count(iris, vars="Species")

       Species freq
1     setosa    50
2 versicolor    50
3  virginica    50
```
```r
> head(iris, n=3)

<table>
<thead>
<tr>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
</tbody>
</table>

> is.data.frame(iris)

[1] TRUE

> dim(iris)

[1] 150 5

> summary(iris)

<table>
<thead>
<tr>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. :4.300</td>
<td>Min. :2.000</td>
<td>Min. :1.000</td>
<td>Min. :0.100</td>
<td></td>
</tr>
<tr>
<td>1st Qu.:5.100</td>
<td>1st Qu.:2.800</td>
<td>1st Qu.:1.600</td>
<td>1st Qu.:0.300</td>
<td></td>
</tr>
<tr>
<td>Median :5.800</td>
<td>Median :3.000</td>
<td>Median :4.350</td>
<td>Median :1.300</td>
<td></td>
</tr>
<tr>
<td>Mean :5.843</td>
<td>Mean :3.057</td>
<td>Mean :3.758</td>
<td>Mean :1.199</td>
<td></td>
</tr>
<tr>
<td>3rd Qu.:6.400</td>
<td>3rd Qu.:3.300</td>
<td>3rd Qu.:5.100</td>
<td>3rd Qu.:1.800</td>
<td></td>
</tr>
</tbody>
</table>
```

M. Schmidberger
Tutorial - Big Data Analyses with R
November 13th, 2013 35 / 119
```r
> summarise(iris, mean_petal_length=mean(Petal.Length),
+           max_petal_length=max(Sepal.Length))

  mean_petal_length max_petal_length
1          3.758         7.9

> ddply(iris, .(Species), summarise,
+           mean_petal_length=mean(Petal.Length),
+           max_petal_length=max(Sepal.Length))

     Species mean_petal_length max_petal_length
1     setosa       1.462          5.8
2 versicolor       4.260          7.0
3  virginica       5.552          7.9

> daply(iris[,c(1,2,5)], .(Species), colwise(mean))

     Species Sepal.Length Sepal.Width
setosa      5.006       3.428
versicolor  5.936       2.77
virginica   6.588       2.974
```
Package: **RJSONIO**

- Serialize **R** objects **to and from JSON**

```r
> library(RJSONIO)
> json <- toJSON(list(a=c(1,2,3), name="Markus"))
> cat(json)
{
  "a": [ 1, 2, 3 ],
  "name": "Markus"
}
> robj <- fromJSON(json)
> robj

$a
[1] 1 2 3

$name
[1] "Markus"
```
Package: bigmemory and big...

- http://www.bigmemory.org/
- manage massive matrices with **shared memory and memory-mapped files**

```r
> library(bigmemory)
> x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
+            matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))
> dim(x)

[1] 100   2

> bigmatrix <- as.big.matrix(x)
> bigmatrix[1:3,]

 [,1]        [,2]
[1,] -0.10500752  0.1020906
[2,]  0.44259378  0.1069441
[3,]  0.01187035 -0.1730431
```
> library(biganalytics)
> require(foreach)
> res <- bigkmeans(bigmatrix, 3)
> res

K-means clustering with 3 clusters of sizes 35, 22, 43

Cluster means:

```
[,1]       [,2]
[1,] 1.22053252 1.062299708
[2,] 0.59426347 0.707238489
[3,] -0.03341732 0.008253787
```

Clustering vector:

```
[1] 3 3 3 3 3 2 3 3 3 3 3 3 3 3 2 2 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3
[38] 3 3 3 3 3 3 3 2 3 3 3 3 3 1 2 2 1 1 1 1 2 1 1 1 2 1 1 2 2 1 1 2
[75] 1 2 2 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 2 1 2 1
```

Within cluster sum of squares by cluster:

```
[1] 3.954684 2.812699 7.952937
```
Package: parallel

- first version was released with R 2.14.0
- contains functionality derived from and pretty much equivalent to the \texttt{multicore} and \texttt{snow} packages

```r
> x <- list(a = 1:10, b = exp(-3:3))
> lapply(x, mean)

$a
[1] 5.5

$b
[1] 4.535125
```
> library(parallel)
> cl <- makeCluster(2)
> parLapply(cl, x, mean)

$a
[1] 5.5

$b
[1] 4.535125

> stopCluster(cl)

> mclapply(x, mean)

$a
[1] 5.5

$b
[1] 4.535125
Package: **Rcpp**

- provides a clean, approachable API that lets you *write high-performance code*
- can help with loops, recursive functions and functions with advanced data structures
- lead to a **2-3** order of magnitude speed up

```r
> library(Rcpp)
> cppFunction('+
+   int add(int x, int y, int z) {
+     int sum = x + y + z;
+     return sum;
+   }'
+
+ )
> add
function (x, y, z)
.Primitive(".Call")(<pointer: 0x1012bed80>, x, y, z)
> add(1, 2, 3)
[1] 6
```
Package: ggplot2

- useful for producing complex graphics relatively simply
- an implementation of the Grammar of Graphics book by Liland Wilkinson
  - the basic notion is that there is a grammar to the composition of graphical components in statistical graphics
  - by directly controlling that grammar, you can generate a large set of carefully constructed graphics from a relatively small set of operations
  - "A good grammar will allow us to gain insight into the composition of complicated graphics, and reveal unexpected connections between seemingly different graphics."
> library(ggplot2)
> qplot(Sepal.Length, Petal.Length, data = iris, 
+     color = Species)
> res <- qplot(Sepal.Length, Petal.Length, data = iris,
+             color = Species, size = Petal.Width, alpha = I(0.5))
> res
```r
> res + geom_line(size=1)
```
> res + geom_boxplot(size=0.2, alpha=I(0.3))
Shiny - easy web application

- developed by RStudio
- turn analyses into interactive web applications that anyone can use
- let your users choose input parameters using friendly controls like sliders, drop-downs, and text fields
- easily incorporate any number of outputs like plots, tables, and summaries
- no HTML or JavaScript knowledge is necessary, only R

http://www.rstudio.com/shiny/
Shiny - Server

- node.js based application to **host** Shiny Apps on web server
- developed by **RStudio**
- hosted beta service by RStudio: https://rstudio.wufoo.com/forms/shiny-server-beta-program/

- **RApache** package provides similar functionality to host and execute R code
- **RApache** more difficult to use, but more flexibility
Hello World Shiny

- a simple application that generates a random distribution with a configurable number of observations and then plots it
  ```r
  > library(shiny)
  > runExample("01_hello")
  ```
- Shiny applications have **two components**:
  - a user-interface definition: ui.R
  - a server script: server.R
- For more documentation check the tutorial: http://rstudio.github.io/shiny/tutorial
Package: bigvis

- tools for exploratory data analysis of large data sets

Fig. 1. Average delay (colour, in minutes) as a function of distance (x axis, in miles) and speed (y axis, in mph) for 76 million flights. The initial view (left) needs refinement to be useful: first we focus on the middle 99.5% of the data (centre) then transform average delay to shrink the impact of unusually high delays and focus on typical values (right). Flights with higher than average speeds (top-right) have shorter delays (red); more interestingly, a subset of shorter, slower flights (bottom-left) have average delays very close to 0 (white).
Outline

1. Big Data

2. R Intro and R Big Data Packages

3. R and Databases
   - Starting Relational
   - R and MongoDB
   - Break and Time for Exercises

4. R and Hadoop
R and Databases

Starting relational

- SQL provides a **standard language** to filter, aggregate, group, sort data
- SQL-like query languages showing up in new places (Hadoop Hive, Impala, ...)
- ODBC provides SQL interface to non-database data (Excel, CSV, text files)
- R stores relational data in **data frames**
Package `sqldf`

- `sqldf` is an R package for running SQL statements on R data frames
- SQL statements in R using "data frame names" in place of "table names"
- a database with appropriate table layouts/schema is automatically created, the data frames are automatically loaded into the database
- the result is read back into R
- `sqldf` supports the SQLite back-end database (by default), the H2 java database, the PostgreSQL database and MySQL
```r
> library(sqldf)
> sqldf("select * from iris limit 2")

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Petal_Length</th>
<th>Petal_Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
</tbody>
</table>

> sqldf("select count(*) from iris")

<table>
<thead>
<tr>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
</tr>
</tbody>
</table>

> sqldf("select Species, count(*) from iris group by Species")

<table>
<thead>
<tr>
<th>Species</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>50</td>
</tr>
<tr>
<td>versicolor</td>
<td>50</td>
</tr>
<tr>
<td>virginica</td>
<td>50</td>
</tr>
</tbody>
</table>
```
```r
> sqldf("select Species, avg(Sepal_Length) 'mean Sepal_Length'
+        variance(Sepal_Width) 'var Sepal_Width'
+        from iris group by Species")

Species mean Sepal_Length var Sepal_Width
1  setosa     5.006        0.14368980
2 versicolor  5.936        0.09846939
3 virginica   6.588        0.10400408
```
Other relational package

- **RMySQL** package provides an interface to MySQL.
- **RPostgreSQL** package provides an interface to PostgreSQL.
- **ROracle** package provides an interface for Oracle.
- **RJDBC** package provides access to databases through a JDBC interface.
- **RSQLite** package provides access to SQLite. The source for the SQLite engine is included.

**One big problem:**
- all packages **read the full result in R memory**
- with Big Data this will fail
on CRAN there are two packages to connect R with MongoDB

- `rmongodb` supported by MongoDB Inc.
  - powerful for big data
  - difficult to use due to BSON objects

- `RMongo`
  - is very similar to all the relational packages
  - easy to use
  - limited functionality
  - reads full results in R memory
  - difficult to install on MAC OS X
uses a **R** to Java bridge and the Java MongoDB driver

```r
> library(RMongo)
> Sys.setenv(NOAWT=1) # for MAC OS X
> mongo <- mongoDbConnect("cc_JwQcDLJSYQJb",
+ "dbs001.mongosoup.de", 27017)
> dbAuthenticate(mongo, username="JwQcDLJSYQJb",
+ password="RSXPkUkXXXXX")
> dbShowCollections(mongo)

[1] "zips"   "ccp"   "system.users"   "system.indexes"
[5] "test"   "test_data"
```
> dbGetQuery(mongo, "zips", "{"state":"AL"}",
+     skip=0, limit=3)

<table>
<thead>
<tr>
<th>X_id</th>
<th>state</th>
<th>loc</th>
<th>pop</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AL</td>
<td>[ -86.51557, 33.584132 ]</td>
<td>6055</td>
<td>ACMAR</td>
</tr>
<tr>
<td>2</td>
<td>AL</td>
<td>[ -86.959727, 33.588437 ]</td>
<td>10616</td>
<td>ADAMSVILLE</td>
</tr>
<tr>
<td>3</td>
<td>AL</td>
<td>[ -87.167455, 33.434277 ]</td>
<td>3205</td>
<td>ADGER</td>
</tr>
</tbody>
</table>

> dbInsertDocument(mongo, "test_data",
+     '{"foo": "bar", "size": 5 }')

[1] "ok"

> dbDisconnect(mongo)
supports the **aggregation framework**

```r
> output <- dbAggregate(mongo, "zips",
+   c('{
+    "$group" : { "_id" : "$state", totalPop : 
+                 { $sum : "$pop" } } } ',
+   ' { "$match" : {totalPop : { $gte : 10000 } } } } ') 
```

in SQL: `SELECT state, SUM(pop) AS pop FROM zips GROUP BY state HAVING pop > (10000)`
Package: rmongodb

- developed on top of the MongoDB supported C driver
- runs almost entirely in native code, so you can expect high performance
- MongoDB and rmongodb use BSON documents to represent objects in the database and for network messages
- BSON is an efficient binary representation of JSON-like objects (http://www.mongodb.org/display/DOCS/BSON)
- there are numerous functions in rmongodb for serializing/deserializing data to/from BSON
rmongodb NEWS:

- new maintainer: markus@mongosoup.de
- new repository: https://github.com/mongosoup/rmongodb
- resubmitted to CRAN and back since end of October: version 1.1.3
- new milestones and features are coming
- please provide feedback
> library(rmongodb)
> mongo <- mongo.create(host="dbs001.mongosoup.de",
>                        db="cc_JwQcDLJSYQJb",
>                        username="JwQcDLJSYQJb",
>                        password="RSXPkUxxxxxxx")

> mongo
[1] 0
attr(,"mongo")
<pointer: 0x1090455c0>
attr(,"class")
[1] "mongo"
attr(,"host")
[1] "dbs001.mongosoup.de"
attr(,"name")
[1] ""
attr(,"username")
[1] "JwQcDLJSYQJb"
attr(,"password")
[1] "RSXPkUkxRd0X"
> mongo.get.database.collections(mongo, "cc_JwQcDLJSYQJb")

[1] "cc_JwQcDLJSYQJb.zips" "cc_JwQcDLJSYQJb.ccp"
[3] "cc_JwQcDLJSYQJb.test" "cc_JwQcDLJSYQJb.test_data"

> # creating BSON buffers
> buf <- mongo.bson.buffer.create()
> err <- mongo.bson.buffer.append(buf, "state", "AL")
> mongo.bson.from.buffer(buf)

state : 2   AL

> # creating BSON buffers from JSON - NEW
> mongo.bson.from.JSON('{"state":"AL"}')

state : 2   AL
> res <- mongo.find.one(mongo, "cc_JwQcDLJSYQJb.zips",
+                     mongo.bson.from.JSON('{"state":"AL"}')
> res

city : 2 ACMAR
loc : 4
0 : 1  -86.515570
1 : 1  33.584132

pop : 16 6055
state : 2 AL
_id : 2 35004

> mongo.bson.value(res, "pop")
[1] 6055
```r
> cursor <- mongo.find(mongo, "cc_JwQcDLJSYQJb.zips", 
+ query=mongo.bson.from.JSON('{}"state":"AL"}')
> res <- NULL
> while (mongo.cursor.next(cursor)){
+ tmp <- mongo.bson.to.list(mongo.cursor.value(cursor))
+ res <- rbind(res, tmp)
+ }
> err <- mongo.cursor.destroy(cursor)
> head(res, n=4)

     city loc        pop state _id
    tmp "ACMAR" Numeric,2  6055 "AL" "35004"
    tmp "ADAMSVILLE" Numeric,2 10616 "AL" "35005"
    tmp "ADGER" Numeric,2  3205 "AL" "35006"
    tmp "KEYSTONE" Numeric,2 14218 "AL" "35007"
```
> # NEW: find all in one batch
> res <- mongo.find.batch(mongo, "cc_JwQcDLJSYQJb.zips",
+ query=mongo.bson.from.JSON('{"state":"AL"}')
> head(res, n=4)

<table>
<thead>
<tr>
<th>city</th>
<th>loc</th>
<th>pop</th>
<th>state</th>
<th>_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMAR</td>
<td>Numeric</td>
<td>6055</td>
<td>&quot;AL&quot;</td>
<td>&quot;35004&quot;</td>
</tr>
<tr>
<td>ADAMSVILLE</td>
<td>Numeric</td>
<td>10616</td>
<td>&quot;AL&quot;</td>
<td>&quot;35005&quot;</td>
</tr>
<tr>
<td>ADGER</td>
<td>Numeric</td>
<td>3205</td>
<td>&quot;AL&quot;</td>
<td>&quot;35006&quot;</td>
</tr>
<tr>
<td>KEYSTONE</td>
<td>Numeric</td>
<td>14218</td>
<td>&quot;AL&quot;</td>
<td>&quot;35007&quot;</td>
</tr>
</tbody>
</table>
> # OLD
> buf <- mongo.bson.buffer.create()
> err <- mongo.bson.buffer.start.object(buf, "pop")
> err <- mongo.bson.buffer.append(buf, "$gt", 100000)
> err <- mongo.bson.buffer.finish.object(buf)
> query <- mongo.bson.from.buffer(buf)
> #
> # NEW
> query <- mongo.bson.from.JSON({'"pop":{"$gt":100000}'})
> #
> mongo.count(mongo, "cc_JwQcDLJSYQJb.zips",
>        query )

[1] 4
```r
> mongo.drop(mongo, "cc_JwQcDLJSYQJb.test_data")
[1] TRUE

> mongo.insert(mongo, "cc_JwQcDLJSYQJb.test_data",
+ list(foo="bar", size=5L))
[1] TRUE

> mongo.insert(mongo, "cc_JwQcDLJSYQJb.test_data",
+ mongo.bson.from.JSON('{"nl":10}')
[1] TRUE

> mongo.count(mongo, "cc_JwQcDLJSYQJb.test_data",
+ mongo.bson.empty())
[1] 2
```
Create BSON objects

- It WAS about creating BSON query or field objects:
  
  ```r
  > ?mongo.bson
  > ?mongo.bson.buffer.append
  > ?mongo.bson.buffer.start.array
  > ?mongo.bson.buffer.start.object
  ```
check version 1.2.X on github

```r
> mongo.get.keys(mongo, "cc_JwQcDLJSYQJb.ccp")
> # implemented
>
> mongo.apply(mongo, "cc_JwQcDLJSYQJb.people",
+           1, keys="age", mean)
> mongo.summary(mongo, "cc_JwQcDLJSYQJb.people")
> # implemented
>
> mongo.aggregate( ... )
> mongo.table( ... )
```
Break and Time of Exercises

- Connect to the RStudio Server:
  - http://rtraining.comsysto.com:8787
  - Get your user and password at the front desk

- Use R as calculator:
  - In your exercise/home directory you can find a file "exercise_Rbasics.R". Run all the commands.
  - R experts will find small challenges in the file.

- Check the Shiny App running on:
  - http://rtraining.comsysto.com:3838/user01/01_hello
    - In your home directory you can find a folder "ShinyApp". This folder holds all the code for several example ShinyApps.
    - There are 11 different ShinyApps. Go to the URL of one or two other ShinyApps from your user, e.g. :
      - http://rtraining.comsysto.com:3838/userXX/05_sliders
    - Feel free to make changes and check the results in your browser.
Use **Rockmongo** to view content of mongodb:
  - http://rtraining.comsysto.com/rockmongo
  - User/pw : admin/admin

or use **ShinyMongo** to view content of mongodb:

Use **R** to connect and query MongoDB
  - In your home directory you can find a file "exercise_Rmongodb.R". Run all the commands.
  - **R** experts will find small challenges in the file.
BREAK

Slides available at

The Hard Life of a NoSQL Coder

- There is one thing you should know before anything happens tonight.

- I hate relations.
NoSQL - all in one

```
{
  "ID": 1,
  "FIRST": "Frank",
  "LAST": "Weigel",
  "ZIP": "94040",
  "CITY": "MV",
  "STATE": "CA"
}
```

<table>
<thead>
<tr>
<th>KEY</th>
<th>First</th>
<th>Last</th>
<th>ZIP_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Frank</td>
<td>Weigel</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Ali</td>
<td>Dodson</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Mark</td>
<td>Azad</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Steve</td>
<td>Yen</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ZIP_id</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEN</td>
<td>CO</td>
<td>30303</td>
</tr>
<tr>
<td>2</td>
<td>MV</td>
<td>CA</td>
<td>94040</td>
</tr>
<tr>
<td>3</td>
<td>CHI</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>4</td>
<td>NY</td>
<td>NY</td>
<td>10010</td>
</tr>
</tbody>
</table>

= +
NoSQL - implicit schema

- **relational schema**: defines columns, their names, and their datatypes
  - error if insert of data doesn’t fit in the schema
- schema-less database allows **any data**
  - structured with individual fields and structures
  - reduces ceremony and increases flexibility
- **implicit schema** in schema-less
  - code that manipulates the data needs to make some assumptions about its structure, such as the **name of fields**
  - data / commands that doesn’t fit: leading to errors
NoSQL - schema-less example

```json
{ 'first': 'martin', 'last': 'fowler',
  'zip_id': { 'city': 'munich', 'zip': 80883 },
  'card_id': 2334 }
{ 'first': 'peter', 'last': 'sadalage',
  'zip_id': 'Munich, Lindwurmstr. 97',
  'card_id': 1333 }
{ 'surname': 'humble', 'first': 'tom',
  'zip_id': { 'zip': 94104, 'city': 'Munich' },
  'card_id': [2334, 6534] }
```
```r
> library(RJSONIO)
> json1 <- fromJSON("{"first":"martin", "last":"fowler",
+ "zip_id": {"city":"munich", "zip":80883},
+ "card_id":2334}"")
> json2 <- fromJSON("{"first":"peter", "last":"sadalage",
+ "zip_id":"Munich, 80892, Lindwurmstr. 97",
+ "card_id":1333}"")
> json3 <- fromJSON("{"surname":"humble", "first":"tom",
+ "zip_id": {"zip":94104, "city":"Munich"},
+ "card_id":[2334, 6534]}"")
> data <- rbind(json1, json2, json3)
> dim(data)

[1] 3 4
```
```r
> data

<table>
<thead>
<tr>
<th>first</th>
<th>last</th>
<th>zip_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>json1</td>
<td>martin</td>
<td>fowler</td>
</tr>
<tr>
<td>json2</td>
<td>peter</td>
<td>sadalage</td>
</tr>
<tr>
<td>json3</td>
<td>humble</td>
<td>tom</td>
</tr>
</tbody>
</table>

> data[3, "zip_id"]

```

```r
data$zip
```

```r
[[1]]
[[1]]$zip
```

```r
[1] 94104
```

```
[[1]]$city
```

```r
[1] "Munich"
```

```r
> data[3, "card_id"]
```

```r
[[1]]
```

```r
[1] 2334 6534
```
> summary(data)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-none-</td>
<td>character</td>
<td>1</td>
<td>-none-</td>
<td>character</td>
</tr>
<tr>
<td>1</td>
<td>-none-</td>
<td>character</td>
<td>1</td>
<td>-none-</td>
<td>character</td>
</tr>
<tr>
<td>1</td>
<td>-none-</td>
<td>character</td>
<td>1</td>
<td>-none-</td>
<td>character</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>zip_id.Length</th>
<th>zip_id.Class</th>
<th>zip_id.Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-none-</td>
<td>list</td>
</tr>
<tr>
<td>1</td>
<td>-none-</td>
<td>character</td>
</tr>
<tr>
<td>2</td>
<td>-none-</td>
<td>list</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>card_id.Length</th>
<th>card_id.Class</th>
<th>card_id.Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-none-</td>
<td>numeric</td>
</tr>
<tr>
<td>1</td>
<td>-none-</td>
<td>numeric</td>
</tr>
<tr>
<td>2</td>
<td>-none-</td>
<td>numeric</td>
</tr>
</tbody>
</table>

R and NoSQL makes your analyses **difficult**
Outline

1. Big Data

2. R Intro and R Big Data Packages

3. R and Databases

4. R and Hadoop
   - Simple Hadoop and Map Reduce Intro
   - Packages `rmr2`, `rhdfs`, `plyrmlr`
   - RHadoop Advanced
   - Exercise for the Discussion Round
RHadoop

- an open source project sponsored by Revolution Analytics
- package overview:
  - **rmr2** hosts all Map Reduce related functions
    * uses Hadoop Streaming API
  - **rhdfs** for interaction with HDFS file system
  - **plyrmr** convenient processing on a Hadoop cluster of large data sets
  - **rhbase** connect with Hadoop’s NoSQL database HBase
- **installation** is the biggest challenge
  - check web for installation guidelines
  - works with MapR, Cloudera, Hortonworks and Apache Hadoop distribution
  - so far there is no official AWS EMR support

https://github.com/RevolutionAnalytics/RHadoop
HDFS and Hadoop cluster

- HDFS is a **block-structured file system**
  - blocks are stored across a cluster of one or more machines with data storage capacity: DataNode
  - data is accessed in a write once and read many model
- HDFS does come with its own utilities for file management
- HDFS file system stores its **metadata** reliably: NameNode
Hadoop Distribution

- the training system runs on MapR
  - a enterprise-grade platform for Hadoop
  - complete Hadoop distribution
  - comprehensive management suite
  - industry-standard interfaces
  - combines open source packages with Enterprise-grade dependability
  - higher performance
  - mount Hadoop with Direct Access NFS

http://www.mapR.com
Parallel Computing basics

- Serial and parallel tasks:

- Problem is broken into a **discrete series** of instructions and they are processed one after another.

- Problem is broken into discrete parts, that can be solved concurrently.
Simple Parallel Computing with R

```r
> x <- 1:5
> x
[1] 1 2 3 4 5
> lapply(x, function(y) y^2)
[1] 1 4 9 16 25
> library(parallel)
> mclapply(x, function(y) y^2)
[1] 1 4 9 16 25
```
My first Map Reduce Job

```r
> library(rmr2)
> rmr.options(backend=c("local"))
> NULL

> small.ints <- to.dfs(keyval(1, 1:100))
> out <- mapreduce(
+   input = small.ints,
+   map = function(k, v) cbind(v, v^2))
> df <- from.dfs(out)
> head(df$val, n=5)

 v
[1,] 1 1
[2,] 2 4
[3,] 3 9
[4,] 4 16
[5,] 5 25
```
> library(rmr2)
> #small.ints <- to.dfs(1:100)
> small.ints <- to.dfs(keyval(1, 1:100))
> out <- mapreduce(
+    input = small.ints,
+    map = function(k, v) cbind(v, v^2))
> df <- from.dfs(out)

- **to.dfs** put the data into HDFS
  - not possible to write out big data, not in a scalable way
  - nonetheless very useful for a variety of uses like writing test cases, learning and debugging
  - can put the data in a file of your own choosing
  - if you don’t specify one it will create temp files and clean them up when done
  - return value is something we call a "big data object"
  - it is a stub, that is the data is not in memory, only some information
library(rmr2)
small.ints <- to.dfs(keyval(1, 1:100))
out <- mapreduce(
  input = small.ints,
  map = function(k, v) cbind(v, v^2))
df <- from.dfs(out)

- **mapreduce** replaces lapply
- input is the variable small.ints which contains the output of to.dfs
- function to apply, which is called a ”map function” is a regular R function with a few constraint
  ▶ a function of two arguments, a collection of keys and one of values
  ▶ returns key value pairs using the function keyval, which can have vectors, lists, matrices or data.frames as arguments
  ▶ avoid calling keyval explicitly but the return value x will be converted with a call to keyval(NULL,x)
- (a reduce function, which we are not using here)
- we are not using the keys at all, only the values, but we still need both to support the general map reduce case
> library(rmr2)
> small.ints <- to.dfs(keyval(1, 1:100))
> out <- mapreduce(
+   input = small.ints,
+   map = function(k, v) cbind(v, v^2))
> df <- from.dfs(out)

- return value is big data object
- you can pass it as input to other jobs
- read it into memory with from.dfs
  ▶ it will fail for big data!
  ▶ from.dfs is complementary to to.dfs and returns a key-value pair collection
Dealing with Input Formats

```r
> air.in =
+   make.input.format(
+       "csv",
+       sep = ",",
+       col.names=c("iata", "airport", "city",
+                     "state", "country", "lat", "long"),
+       stringsAsFactors = FALSE
+   )

> air2007 <- from.dfs("../exercises/data/airports.csv",
+                      format = air.in)

> air2007$key
NULL

> head(air2007$val, n=3)

         iata     airport      city     state  country         lat         long
1  00M  Thigpen Bay Springs    MS  USA 31.95376  -89.23450
2  00R Livingston Municipal Livingston TX  USA 30.68586  -95.01793
3  00V Meadow Lake Colorado Springs  CO  USA 38.94575 -104.56989
```
```r
> air.subs =
+   mapreduce(
+     ".../exercises/data/airports.csv",
+     input.format = air.in)
> air.subs

function ()
{
  fname
}
<environment: 0x1045d6630>

> air.mem = from.dfs(air.subs)
> names(air.mem)
[1] "key" "val"
> air.df = values(air.mem)
```
```r
> important.cols = c("airport", "city", "state")
> air.df = subset(air.df, select = important.cols)
> head(air.df, n=3)

<table>
<thead>
<tr>
<th>airport</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thigpen Bay Springs MS</td>
<td>Bay Springs</td>
<td>MS</td>
</tr>
<tr>
<td>Livingston Municipal</td>
<td>Livingston</td>
<td>TX</td>
</tr>
<tr>
<td>Meadow Lake Colorado</td>
<td>Colorado Springs</td>
<td>CO</td>
</tr>
</tbody>
</table>
```
\begin{verbatim}
> air.subs = 
+   mapreduce(
+     air.subs, 
+     map = 
+       function(k, v)
+         subset(v, select = important.cols))

> head( values( from.dfs(air.subs) ), n=3)

          airport             city            state
 1  Thigpen Bay Springs    MS
 2 Livingston Municipal   Livingston    TX
 3  Meadow Lake Colorado Springs CO
\end{verbatim}
> air.subs =
+    mapreduce(
+        ".../exercises/data/airports.csv",
+        input.format = air.in,
+        map =
+            function(k, v)
+            subset(v, city=="Perry")
+    )
> head(values(from.dfs(air.subs), n=3))

<table>
<thead>
<tr>
<th>iata</th>
<th>airport</th>
<th>city</th>
<th>state</th>
<th>country</th>
<th>lat</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Perry-Warsaw</td>
<td>Perry</td>
<td>NY</td>
<td>USA</td>
<td>42.74135</td>
<td>-78.05208</td>
</tr>
<tr>
<td>382</td>
<td>Perry-Foley</td>
<td>Perry</td>
<td>FL</td>
<td>USA</td>
<td>30.06928</td>
<td>-83.58058</td>
</tr>
<tr>
<td>1456</td>
<td>Perry Municipal</td>
<td>Perry</td>
<td>OK</td>
<td>USA</td>
<td>36.38560</td>
<td>-97.27721</td>
</tr>
</tbody>
</table>
> air.subs =
+    mapreduce(
+        ".../exercises/data/airports.csv",
+        input.format = air.in,
+        map =
+            function(k, v){
+                v = v[!is.na(as.character(v$state)), ]
+                keyval( as.character(v$state),
+                        as.integer(v$lat) ) },
+        reduce =
+            function(k, v)
+                cbind(state=k,
+                    lat_mean= mean(v, na.rm = TRUE) )
+    )
> head( values( from.dfs(air.subs) ), n=4)

    state  lat_mean
[1,] "MS"   "32.375"
[2,] "TX"   "30.9808612440191"
[3,] "CO"   "38.7959183673469"
[4,] "NY"   "41.9587628865979"
Hadoop Hello World - "Word - Count"

The overall MapReduce word count process

Input

Splitting

Mapping

Shuffling

Reducing

Final result

Deer Bear River

Car Car River

Deer Car Bear

Deer, 1 Bear, 1 River, 1

Car, 1 Car, 1 Car, 1

Deer, 1 Deer, 1

River, 1 River, 1

Bear, 2 Bear, 1

Car, 3 Car, 3 Car, 3

Deer, 2 Deer, 2 Deer, 2

River, 2 River, 2 River, 2
> wc.map =
+    function(., lines) {
+        key <- unlist(
+            strsplit(x = lines,
+                split = pattern))
+        keyval( key, 1 )
+    }
> wc.reduce =
+    function(word, counts) {
+        keyval( word, sum(counts) )
+    }
> rmr.options(backend=c("local"))

NULL

> pattern <- " +"
> out <- mapreduce(  
+   input = "../exercises/data/faust.txt" ,  
+   input.format = "text",  
+   map = wc.map,  
+   reduce = wc.reduce,  
+   combine = T,  
+   in.memory.combine = F)
> res <- from.dfs(out)
> id <- which(res$key=="Alle")
> res$val[id]

[1] 5
Hadoop Environments Variables

- all RHadoop packages have to access hadoop
- in Linux they need the correct environment variables
- in R you have to set them explicitly

```r
> Sys.setenv(HADOOP_CMD="/opt/mapr/hadoop/hadoop-0.20.2/bin/hadoop")
> Sys.setenv(HADOOP_STREAMING="/opt/mapr/hadoop/hadoop-0.20.2/contrib/streaming/hadoop-0.20.2-dev-streaming.jar")
> library(rmr2)
> # rmr.options(backend=c("hadoop"))
> small.ints <- to.dfs(keyval(1, 1:100))
> out <- mapreduce(
+     input = small.ints,
+     map = function(k, v) cbind(v, v^2))
```
Package *rhdfs*

- basic connectivity to the **Hadoop Distributed File System**
- can browse, read, write, and modify files stored in HDFS
- package has a dependency on **rJava**
- is dependent upon the HADOOP_CMD environment variable

```r
> Sys.setenv(HADOOP_CMD="/opt/mapr/hadoop/
  + hadoop-0.20.2/bin/hadoop")
> library(rhdfs)
> hdfs.init()
> hdfs.ls()
> data <- 1:1000
> file <- hdfs.file("my_test", "w")
> hdfs.write(data, file)
> hdfs.close(file)
> hdfs.ls()
```
with the MapR Hadoop distribution we basically **do not need** the package

- mount Hadoop with Direct Access **NFS**

```r
> list.files(path = "/mnt/mapr/data")
> list.dir(path = "/mnt/mapr/")
> file.info("/mnt/mapr/data/zips.json")
> ?file.create
```
Package **plyrmr**

- perform common **data manipulation** operations on very large data sets stored on Hadoop

```r
> air.df = transform(air.df,
+     aiport = airport == "Hilliard Airpark")
> head( air.df, n=5 )
```

<table>
<thead>
<tr>
<th>airport</th>
<th>city</th>
<th>state</th>
<th>aiport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thigpen Bay Springs</td>
<td>MS</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>Livingston Municipal</td>
<td>Livingston</td>
<td>TX</td>
<td>FALSE</td>
</tr>
<tr>
<td>Meadow Lake Colorado Springs</td>
<td>CO</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>Perry-Warsaw</td>
<td>Perry</td>
<td>NY</td>
<td>FALSE</td>
</tr>
<tr>
<td>Hilliard Airpark</td>
<td>Hilliard</td>
<td>FL</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
library(plyrmr)

air.subs = transform(
    input("../exercises/data/airports.csv",
        format = air.in),
    airport = airport == "Hilliard Airpark")

> air.subs

[1] "Got it! To generate results call the functions output or as.data.frame on this object. Computation has been delayed at least in part."

> air.df = as.data.frame(air.subs)

> head( air.df, n=5 )

    iata  airport          city          state country    lat    long
1    00M  Thigpen Bay Springs   MS     USA  31.95376 -89.23450
2    00R Livingston Municipal Livingston   TX     USA  30.68586 -95.01793
3    00V Meadow Lake Colorado Springs CO     USA  38.94575 -104.56989
4    01G Perry-Warsaw Perry       NY     USA  42.74135 -78.05208
5    01J Hilliard Airpark Hilliard    FL     USA  30.68801 -81.90594

airport
1 FALSE
2 FALSE
3 FALSE
4 FALSE
5 TRUE
>` air.df = select(air.df, airport, city)
>` head( air.df, n=3 )

        airport      city
   1   Thigpen Bay Springs
   2  Livingston Municipal Livingston
   3 Meadow Lake Colorado Springs
\begin{verbatim}
> air.subs = + select(air.subs, +     airport, city) > head( as.data.frame(air.subs), n=3 )

   airport          city
1   Thigpen Bay Springs
2 Livingston Municipal Livingston
3 Meadow Lake Colorado Springs

... and many more commands: gather, merge, sample, summarize, where, ...
\end{verbatim}
• **rmr2** the easiest, most productive, most elegant way to write map reduce jobs

• with **rmr2** one-two orders of magnitude less code than Java

• with **rmr2** readable, reusable, extensible map reduce

• with **rmr2** is a great prototyping, executable spec and research language

• **rmr2** is a way to work on big data sets in a way that is 'R-like'

• 'Simple things should be simple, complex things should be possible'
- **rmr2** is not Hadoop Streaming
  - it uses streaming
  - no support for every single option that streaming has
  - streaming is accessible from R with no additional packages because R can execute an external program and R scripts can read stdin and stdout

- map reduce programs written in rmr2 are not going to be the most efficient
- Hive, Impala you can access via RODBC
- Hive you can access with the R package RHive
- HBASE you can access with the R package rhbase
- Want to learn more? Check Revolution Analytics Webinars
RHadoop k-means clustering

- live demo
Exercise for the Discussion Round

- Connect to the RStudio Server:
  - http://rtraining.comsysto.com:8787
- Use R to run MapReduce jobs and explore HDFS
  - In your home/exercise directory you can find a file "exercise_Rhadoop.R". Run all the commands.
  - R experts: check the hadoop-*\.R files and start loving Hadoop.
  - You can access the MapR dashboard via https://rtraining.comsysto.com:8443 (user: 'user', password: 'comsysto')
Summary

- **R** is a powerful statistical tool to analyse many different kind of data: from small to big data

- **R** can access databases
  - MongoDB and **rmongodb** support big data

- **R** can run Hadoop jobs
  - **rmr2** runs your Map Reduce jobs
  - **plyrmr** makes big data management on Hadoop easy

- **R** is open source and there is a lot of community driven development

http://www.r-project.org/
the tutorial did not cover ...  
- the commercial R version: **Revolution R Enterprise**
  - RRE7 is coming
  - R functions to run directly on Hadoop
  - in-database analytics (Teradata)
  - [http://www.revolutionanalytics.com](http://www.revolutionanalytics.com)
- Oracle R Advanced Analytics for Hadoop package:  
- pbdR: programming with big data in R: [http://r-pbd.org/](http://r-pbd.org/)
http://datacommunitydc.org/blog/2013/05/stepping-up-to-big-data-with-r-and-python
How to go on

- start playing around with **R and Big Data**
- get part of the community
  - http://www.r-bloggers.com
  - http://hadoop.comsysto.com
- interested in more **R courses** hosted by comSysto GmbH in Munich, Germany?
  - two day **R** beginners training (05. - 06.12.2013)
  - one day **R** Data Handling and Graphics (12.12.2013)
  - one day **R** Big Data Analyses (13.12.2013)
  - http://comsysto.com/events
- **rmongodb** webinar in January 2014:
  https://www3.gotomeeting.com/register/287023934
Goodbye

Twitter: @cloudHPC
Email: markus.schmidberger@comsysto.com