NoSQL Roadshow Munich 2013

Training - Big Data Analyses with R

Dr. rer. nat. Markus Schmidberger

@cloudHPC
markus.schmidberger@comsysto.com

comSysto GmbH

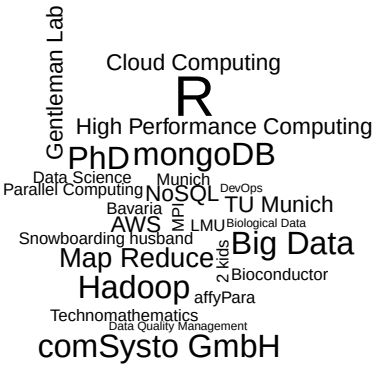
April 16th, 2013

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

Outline

Dr. rer. nat. Markus Schmidberger



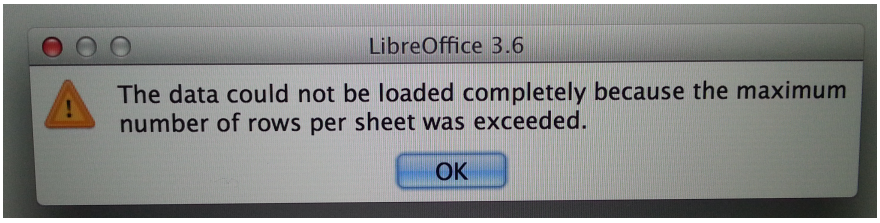
comSysto GmbH

- 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/Hadoop-User-Group-Munich/`
  - `http://www.meetup.com/Muenchen-MongoDB-User-Group/`
  - `http://www.meetup.com/munich-useR-group/`
- `http://www.comsysto.com`

comSysto sense and respond

Big Data

- a big hype topic
- everything is big data
- everyone want 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



- access to many different data sources (Internet)
- storage is cheep - store everything
- today, CIOs get interested in the power of all their data
- $\Rightarrow$ a lot of different and complex data have to be stored in one data warehouse
- $\Rightarrow$ NoSQL

NoSQL - MongoDB

- databases using looser consistency models to store data
- MongoDB:
  - open source document-oriented database system
  - supported by 10gen
  - stores structured data as JSON-like documents

`http://www.mongodb.org`

Big Data

my view on processing

- 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- backward-looking analysis is outdated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
- $\Rightarrow$ more complex methods, more processing time
- $\Rightarrow$ Hadoop, Super Computer and statistical tools

Hadoop

- a big data technology
- open-source software framework designed to support large scale data processing
- supports data-intensive distributed applications
- inspired by Google's MapReduce based computational infastructure
- MapReduce: 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 through its Streaming API

`http://hadoop.apache.org`

Big Data

my view on resources

- computing resources are available to everyone and cheep
- 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: = 'Fachkräftemangel'
- $\Rightarrow$ welcome to the 'Training Big Data Analyses with R' to solve this problem

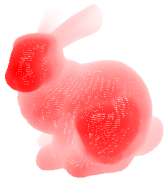
Outline

www.r-project.org

- open-source: **R** is a free software environment for statistical computing and graphics
- offers tools to manage and analyze data
- standard statistical methods are implemented
- huge online-libraries with **R**-packages:
  - CRAN: `http://cran.r-project.org/`
  - BioConductor (genomic data): `http://www.bioconductor.org/`
  - Omegahat: `http://www.omegahat.org/`
  - **R**-Forge: `http://r-forge.r-project.org/`
- all statistical analyzes tools and latest developments
- possibility to write personalized code and to contribute new packages

- source code is open and can be modified
- compiles and runs under different systems: UNIX, Windows, MacOS
- 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`
- support via several manuals and books:
  - `http://www.r-project.org/doc/bib/R-books.html`
  - `http://cran.r-project.org/manuals.html`
- no standardized graphical user interface (GUI)
- really famous since January 6, 2009: The New York Times, "Data Analysts Captivated by **R**'s Power"

- **R** is an S interpreter
  - S is a full-featured programming language
- a lot of string processing and graphical capabilities are built into **R**
- **R** can be used for a lot of things that have nothing to do with statistics, e.g.:
  - create a web gallery of your vacation photos
  - write serial emails
  - write presentation slides with LaTex
  - check **R** code against documentation
  - create a 3D shape files



```
> library(onion)
> data(bunny)
> p3d(bunny,theta=3,phi=104,box=FALSE)
```

R vs. SAS vs. Julia

R is open source, SAS is a commercial product and 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

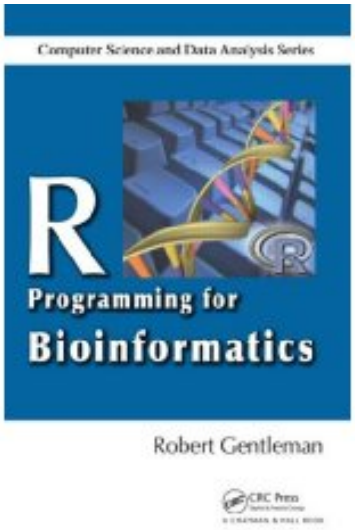
All three work very well and there are other powerful analysis tools

 ⇔  ⇔ Julia

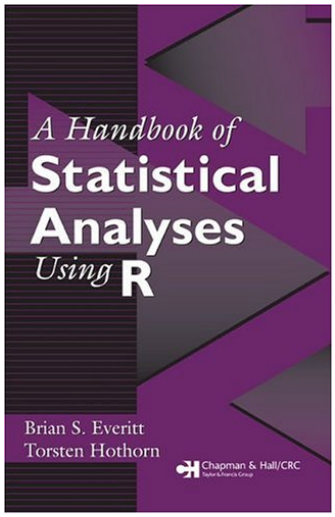
R books


Statistik und ihre Anwendungen
Uwe Ligges
Programmieren mit R
2. Auflage
Springer

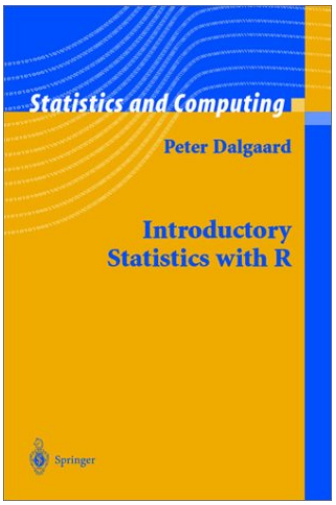
(a)


Computer Science and Data Analysis Series
R
Programming for
Bioinformatics
Robert Gentleman
CRC Press

(b)


A Handbook of
Statistical
Analyses
Using R
Brian S. Everitt
Torsten Hothorn
Chapman & Hall/CRC

(c)


Statistics and Computing
Peter Dalgaard
Introductory
Statistics with R
Springer

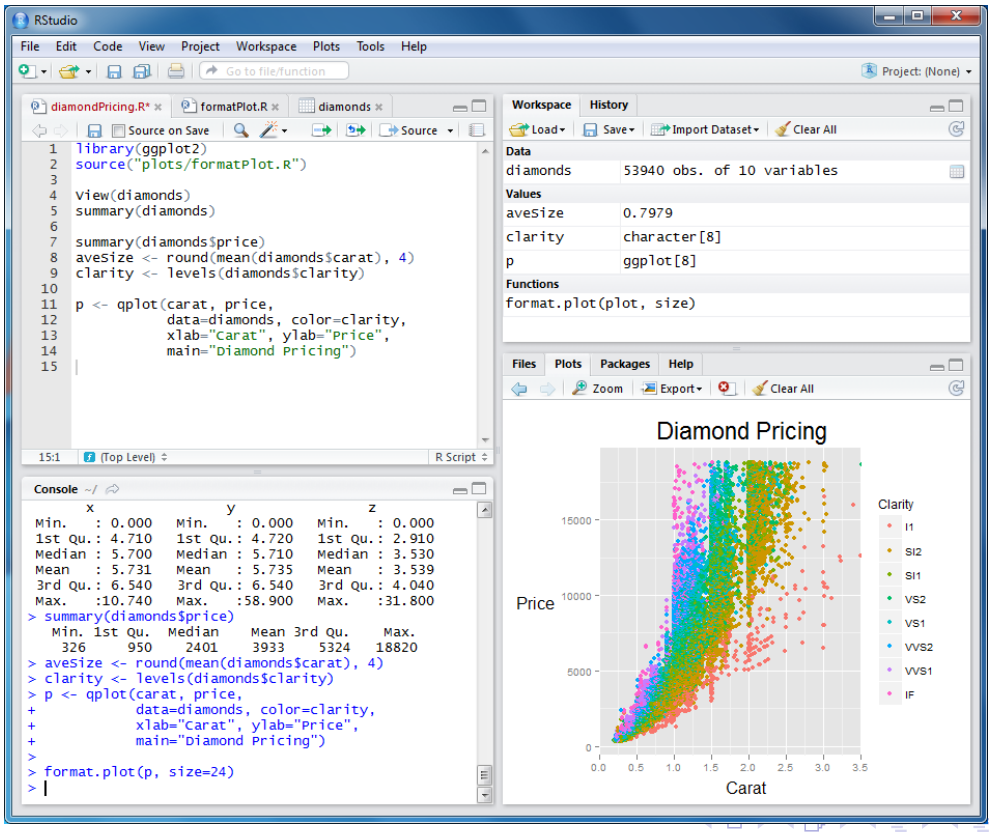
(d)

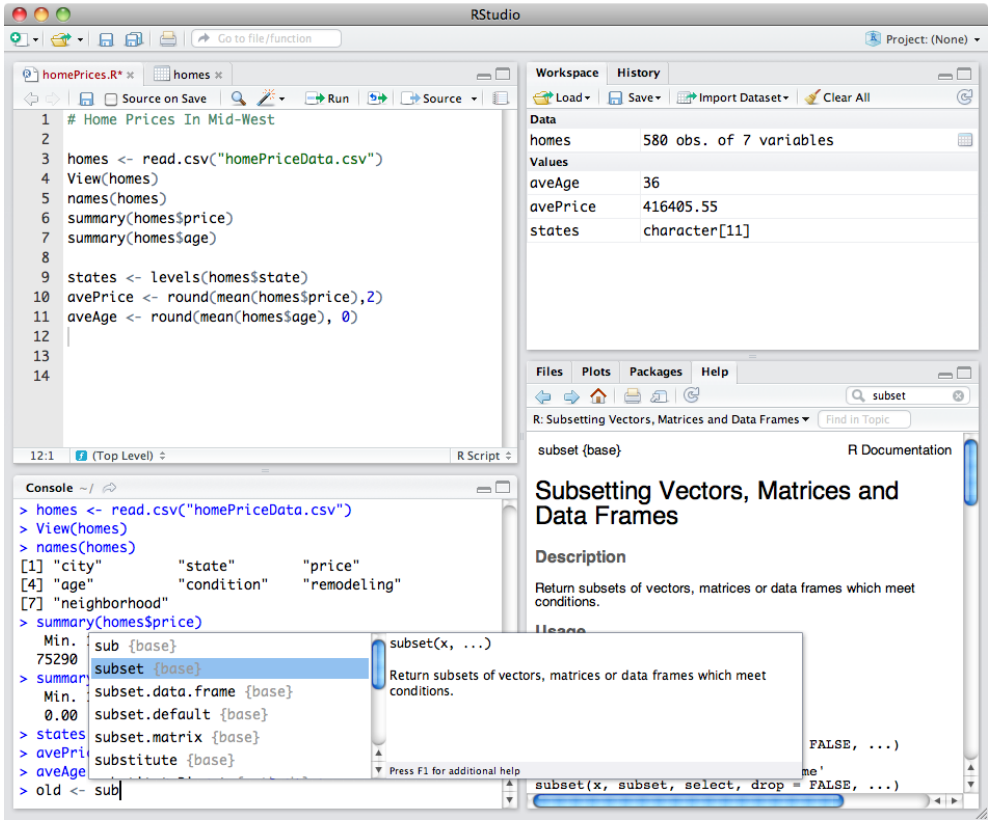
Installation

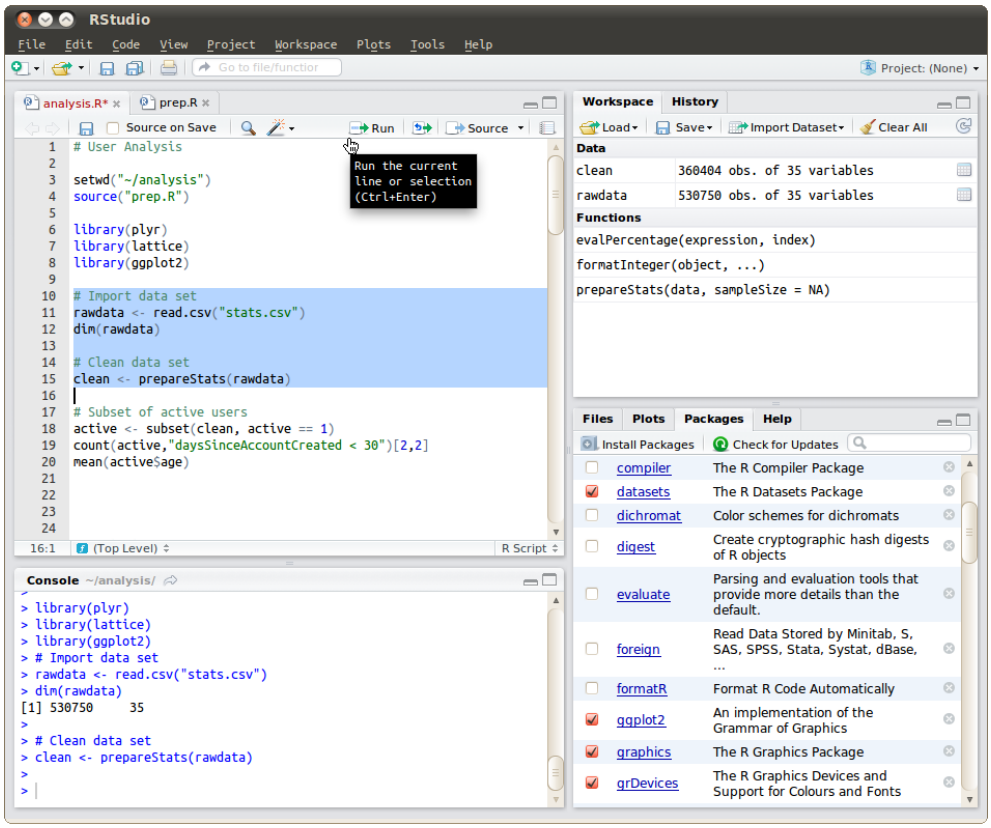
- Installation and Download:
  `http://cran.r-project.org/sources.html`
- Linux, MacOS X, and Windows support
- starting **R**: R (at your command line)
- Editors: emacs, Tinn-**R**, Eclipse, **R**Studio, ...
- quitting **R**:
  `> q()`

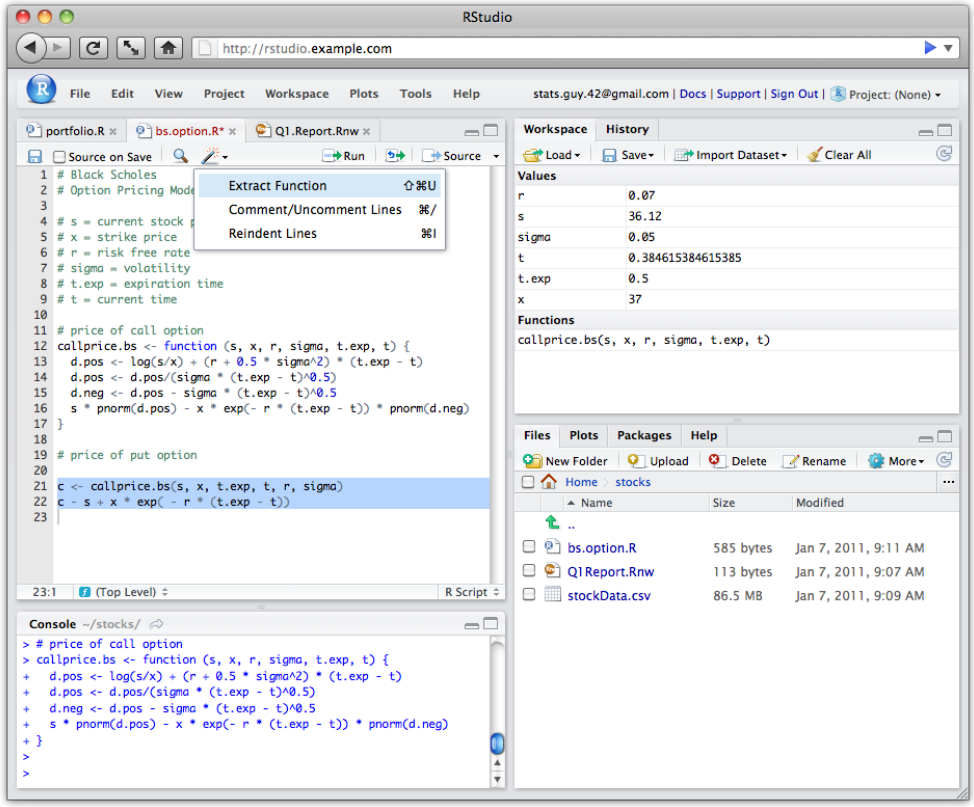
RStudio IDE

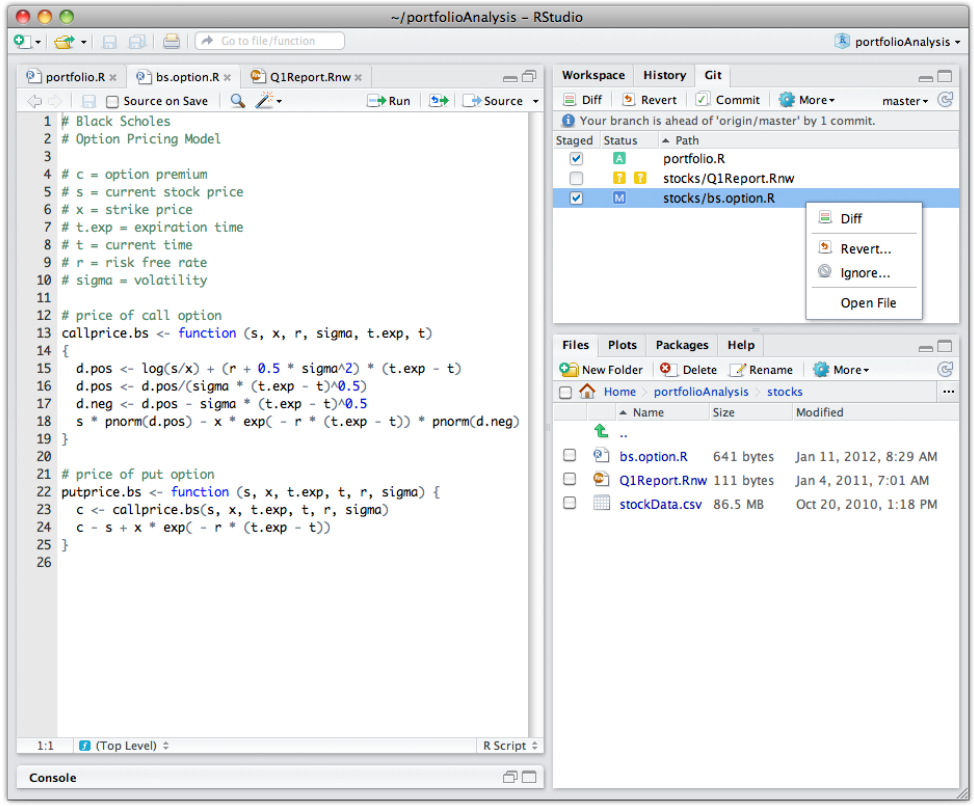
`http://www.rstudio.com`




RStudio
homePrices.R*
homes
Source on Save
Run
Source
1 # Home Prices In Mid-West
2
3 homes <- read.csv("homePriceData.csv")
4 View(homes)
5 names(homes)
6 summary(homes$price)
7 summary(homes$age)
8
9 states <- levels(homes$state)
10 avePrice <- round(mean(homes$price),2)
11 aveAge <- round(mean(homes$age), 0)
12
13
14
12:1
(Top Level)
R Script
Console ~/
> homes <- read.csv("homePriceData.csv")
> View(homes)
> names(homes)
[1] "city" "state" "price"
[4] "age" "condition" "remodeling"
[7] "neighborhood"
> summary(homes$price)
Min.
75290
> summary
Min.
0.00
> states
> avePri
> aveAge
> old <- sub
sub {base}
subset {base}
subset.data.frame {base}
subset.default {base}
subset.matrix {base}
substitute {base}
subset(x, ...)
Return subsets of vectors, matrices or data frames which meet conditions.
Press F1 for additional help
Workspace
History
Load
Save
Import Dataset
Clear All
Data
homes 580 obs. of 7 variables
Values
aveAge 36
avePrice 416405.55
states character[11]
Files
Plots
Packages
Help
subset
R: Subsetting Vectors, Matrices and Data Frames
Find in Topic
subset {base}
R Documentation
Subsetting Vectors, Matrices and Data Frames
Description
Return subsets of vectors, matrices or data frames which meet conditions.
FALSE, ...)
subset(x, subset, select, drop = FALSE, ...)


RStudio
File Edit Code View Project Workspace Plots Tools Help
analysis.R*
prep.R
Source on Save
Run
Source
1 # User Analysis
2
3 setwd("~/analysis")
4 source("prep.R")
5
6 library(plyr)
7 library(lattice)
8 library(ggplot2)
9
10 # Import data set
11 rawdata <- read.csv("stats.csv")
12 dim(rawdata)
13
14 # Clean data set
15 clean <- prepareStats(rawdata)
16
17 # Subset of active users
18 active <- subset(clean, active == 1)
19 count(active,"daysSinceAccountCreated < 30")[2,2]
20 mean(active$age)
21
22
23
24
Run the current line or selection (Ctrl+Enter)
16:1
(Top Level)
R Script
Console ~/analysis/
> library(plyr)
> library(lattice)
> library(ggplot2)
> # Import data set
> rawdata <- read.csv("stats.csv")
> dim(rawdata)
[1] 530750 35
>
> # Clean data set
> clean <- prepareStats(rawdata)
>
>
Workspace
History
Load
Save
Import Dataset
Clear All
Data
clean 360404 obs. of 35 variables
rawdata 530750 obs. of 35 variables
Functions
evalPercentage(expression, index)
formatInteger(object, ...)
prepareStats(data, sampleSize = NA)
Files
Plots
Packages
Help
Install Packages
Check for Updates
compiler The R Compiler Package
datasets The R Datasets Package
dichromat Color schemes for dichromats
digest Create cryptographic hash digests of R objects
evaluate Parsing and evaluation tools that provide more details than the default.
foreign Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, dBase, ...
formatR Format R Code Automatically
ggplot2 An implementation of the Grammar of Graphics
graphics The R Graphics Package
grDevices The R Graphics Devices and Support for Colours and Fonts


RStudio
http://rstudio.example.com
File Edit View Project Workspace Plots Tools Help
stats.guy.42@gmail.com | Docs | Support | Sign Out | Project: (None)
portfolio.R
bs.option.R*
Q1.Report.Rnw
Source on Save
Run
Source
Extract Function ⇧⌘U
Comment/Uncomment Lines ⌘/
Reindent Lines ⌘I
# Black Scholes
# Option Pricing Mode
# s = current stock
# x = strike price
# r = risk free rate
# sigma = volatility
# t.exp = expiration time
# t = current 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.pos <- d.pos/(sigma * (t.exp - t)^0.5)
d.neg <- 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
c <- callprice.bs(s, x, t.exp, t, r, sigma)
c - s + x * exp(- r * (t.exp - t))
23:1 (Top Level) R Script
Console ~/stocks/
> # 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.pos <- d.pos/(sigma * (t.exp - t)^0.5)
+ d.neg <- d.pos - sigma * (t.exp - t)^0.5
+ s * pnorm(d.pos) - x * exp(- r * (t.exp - t)) * pnorm(d.neg)
+ }
Workspace History
Load Save Import Dataset Clear All
Values
r 0.07
s 36.12
sigma 0.05
t 0.384615384615385
t.exp 0.5
x 37
Functions
callprice.bs(s, x, r, sigma, t.exp, t)
Files Plots Packages Help
New Folder Upload Delete Rename More
Home stocks
Name Size Modified
bs.option.R 585 bytes Jan 7, 2011, 9:11 AM
Q1Report.Rnw 113 bytes Jan 7, 2011, 9:07 AM
stockData.csv 86.5 MB Jan 7, 2011, 9:09 AM


~/portfolioAnalysis - RStudio
portfolioAnalysis
portfolio.R
bs.option.R
Q1Report.Rnw
Source on Save
Run
Source
# Black Scholes
# Option Pricing Model
# c = option premium
# s = current stock price
# x = strike price
# t.exp = expiration time
# t = current time
# r = risk free rate
# sigma = volatility
# 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.pos <- d.pos/(sigma * (t.exp - t)^0.5)
d.neg <- 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
putprice.bs <- function (s, x, t.exp, t, r, sigma) {
c <- callprice.bs(s, x, t.exp, t, r, sigma)
c - s + x * exp(- r * (t.exp - t))
}
1:1 (Top Level) R Script
Console
Workspace History Git
Diff Revert Commit More master
Your branch is ahead of 'origin/master' by 1 commit.
Staged Status Path
portfolio.R
stocks/Q1Report.Rnw
stocks/bs.option.R
Diff
Revert...
Ignore...
Open File
Files Plots Packages Help
New Folder Delete Rename More
Home portfolioAnalysis stocks
Name Size Modified
bs.option.R 641 bytes Jan 11, 2012, 8:29 AM
Q1Report.Rnw 111 bytes Jan 4, 2011, 7:01 AM
stockData.csv 86.5 MB Oct 20, 2010, 1:18 PM

R as calculator

```
>  (5+5) - 1 * 3
[1] 7
>  abs(-5)
[1] 5
>  x <- 3
>  x
[1] 3
>  x^2 + 4
[1] 13
> sum(1,2,3,4)
[1] 10
```


```
> help("mean")
> ?mean
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> y <- c(1,2,3)
> y
[1] 1 2 3
> 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

```
> # 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)
```

Package: **ggplot2**

- Author: Prof. Hardley Wickham (since 2012 RStudio member)
- 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)
> head(mtcars, n=3)
```

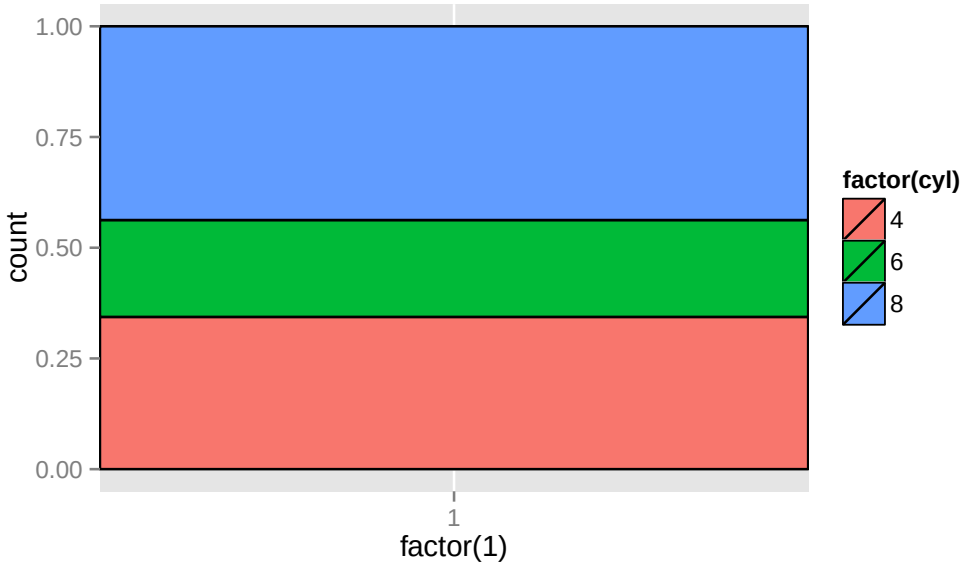
```
               mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

```
> summary(mtcars)
```

```
      mpg             cyl             disp             hp       
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
      drat             wt             qsec             vs        
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
       am              gear            carb      
 Min.   :0.0000   Min.   :3.000   Min.   :1.000  
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
 Median :0.0000   Median :4.000   Median :2.000  
 Mean   :0.4062   Mean   :3.688   Mean   :2.812  
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
 Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

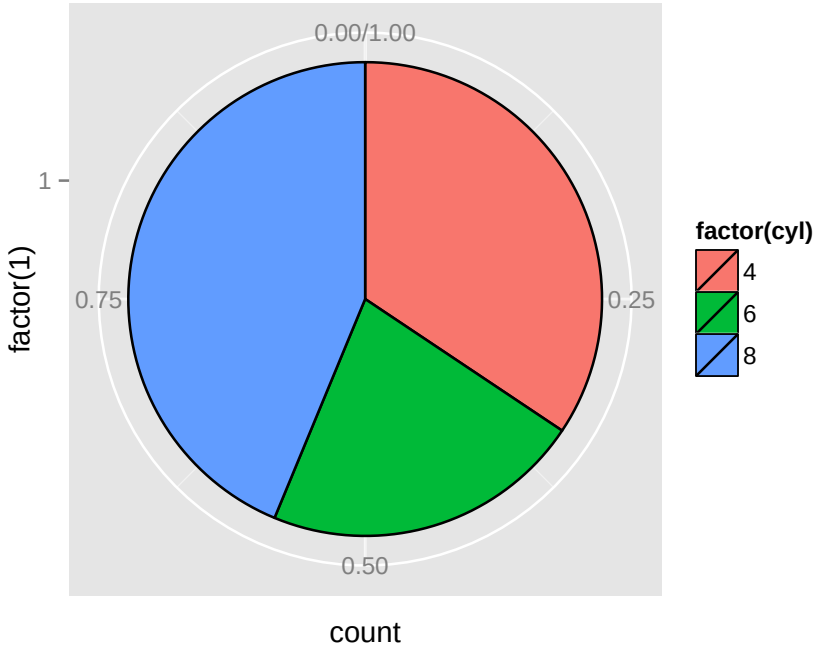


```
> pie <- ggplot(mtcars, aes(x = factor(1),
+                           fill = factor(cyl))) +
+       geom_bar(width = 1,
+                position = "fill", color = "black")
> pie
```



[cyl: number of cylinders]

```
> pie + coord_polar(theta = "y")
```



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 webserver
- Developed by RStudio
- **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

```
> 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`

Exercise 1

- Connect to the **R**Studio Server:
  - `http://bigdata.comsysto.com:8787`
  - user01 - user30 (pw: comsysto)
- Use **R** as calculator:
  - In your home directory you can find a file "exercise1.R". Run all the commands.
  - R experts will find a small challenge at the end of the file.
- Check the Shiny App running on:
  `http://bigdata.comsysto.com:3838/users/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://bigdata.comsysto.com:3838/users/userXX/05_sliders`
  - Feel free to make changes and check the results in your browser.

Outline

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)
- ODBC provides SQL interface to non-database data (Excel, CSV, text files)
- **R** stores relational data in data.frames

```
> data(iris)
> head(iris, n=3)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
> class(iris)
[1] "data.frame"
```

Package **sqldf**

- sqldf is an **R** package for runing 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 backend database (by default), the H2 java database, the PostgreSQL database and MySQL

```
> library(sqldf)
> sqldf("select * from iris limit 2")
  Sepal_Length Sepal_Width Petal_Length Petal_Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
> sqldf("select count(*) from iris")
  count(*)
1      150
> sqldf("select Species, count(*) from iris group by Species")
     Species count(*)
1     setosa       50
2 versicolor       50
3  virginica       50
> sqldf("select avg(Sepal_Length) mean, variance(Sepal_Length) var
      mean       var
1 5.843333 0.6856935
```

Package **RODBC**

- provides access to databases (including Microsoft Access and Microsoft SQL Server) through an ODBC interface.

```
> library(RODBC)
> myconn <-odbcConnect("mydsn", uid="Rob", pwd="aardvark")
> crimedat <- sqlFetch(myconn, Crime)
> pundat <- sqlQuery(myconn, "select * from Punishment")
> close(myconn)
```

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 full result in **R** memory
- for Big Data this can be a problem

R and MongoDB

- on CRAN there are two packages to connect **R** with MongoDB
  - **rmongodb** supported bei 10gen
    - ★ powerfull 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
    - ★ does not work on MAC OSX

Package **Rmongo**

- uses a **R** to Java bridge and the Java MongoDB driver

```
> library(Rmongo)
> mongo <- mongoDbConnect("test", "localhost", 27017)
> dbShowCollections(mongo)
> dbGetQuery(mongo, "zips","{'state':'AL'}")
> dbGetQuery(mongo, "zips","{'pop':{'$gt': 100000}}}")
> dbInsertDocument(mongo, "test_data",
+                  '{"foo": "bar", "size": 5 }')
> dbInsertDocument(mongo, "test_data",
+                  '{"foo": "nl", "size": 10 }')
> dbGetQuery(mongo, "test_data","{}")
```

- supports the aggregation framework

```
> 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


```
> mongo <- mongo.create()
> mongo.insert(mongo, namespace, record)
> result <- mongo.find.one(mongo, namespace, query)
> mongo.update(mongo, namespace, criteria, objNew)
> mongo.remove(mongo, namespace, criteria)
> mongo.destroy(mongo)
```

```
> library(rmongodb)
rmongodb package (mongo-r-driver) loaded
Use 'help("mongo")' to get started.
> mongo <- mongo.create(host="localhost")
> mongo.get.database.collections(mongo, "test")
[1] "test.zips"      "test.test"      "test.test2"     "test.test_
> buf <- mongo.bson.buffer.create()
> mongo.bson.buffer.append(buf, "state", "AL")
[1] TRUE
> query <- mongo.bson.from.buffer(buf)
> cursor <- mongo.find(mongo, "test.zips", query)
> cursor
[1] 0
attr(,"mongo.cursor")
<pointer: 0x1006dce00>
attr(,"class")
[1] "mongo.cursor"
```


```
> res <- NULL
> while (mongo.cursor.next(cursor)){
+     tmp <- mongo.bson.to.list(mongo.cursor.value(cursor))
+     res <- rbind(res, tmp)
+ }
> mongo.cursor.destroy(cursor)
[1] FALSE
> mongo.destroy(mongo)
NULL
> 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"
```

```
> mongo <- mongo.create()
> buf <- mongo.bson.buffer.create()
> mongo.bson.buffer.start.object(buf, "pop")
[1] TRUE
> mongo.bson.buffer.append(buf, "$gt", 100000L)
[1] TRUE
> mongo.bson.buffer.finish.object(buf)
[1] TRUE
> query <- mongo.bson.from.buffer(buf)
> mongo.count(mongo, "test.zips", query)
[1] 4
```


```
> mongo.insert(mongo, "test.test_data",
+              list(foo="bar", size=5L))
[1] TRUE
> mongo.insert(mongo, "test.test_data",
+              list(foo="nl", size=10L))
[1] TRUE
> buf <- mongo.bson.buffer.create()
> query <- mongo.bson.from.buffer(buf)
> mongo.count(mongo, "test.test_data", query)
[1] 6
```

```
> cursor <- mongo.find(mongo, "test.test_data", query)
> res <- NULL
> while (mongo.cursor.next(cursor)){
+     tmp <- mongo.bson.to.list(mongo.cursor.value(cursor))
+     res <- rbind(res, tmp)
+ }
> mongo.cursor.destroy(cursor)
[1] FALSE
> mongo.destroy(mongo)
NULL
> head(res, n=3)
    _id      foo   size
tmp 90162792 "bar" 5
tmp 81870720 "nl"  10
tmp 84941544 "bar" 5
```

Create BSON objects

- It is all about creating BSON query or field objects:

```
> b <- mongo.bson.from.list(
+   list(name="Fred", age=29, city="Boston"))
> b
name : 2   Fred
age : 1   29.000000
city : 2   Boston
> l <- mongo.bson.to.list(b)
> l
$name
[1] "Fred"

$age
[1] 29

$city
[1] "Boston"
```

```
> ?mongo.bson.buffer.append
> buf <- mongo.bson.buffer.create()
> mongo.bson.buffer.append(buf, "name", "Fred")
[1] TRUE
> mongo.bson.buffer.append(buf, "city", "Dayton")
[1] TRUE
> mongo.bson.buffer.append.int(buf, "year", 1968)
[1] TRUE
> b <- mongo.bson.from.buffer(buf)
> b
name : 2   Fred
city : 2   Dayton
year : 16   1968
```


```
> buf <- mongo.bson.buffer.create()
> mongo.bson.buffer.start.array(buf, "Fibonacci")
[1] TRUE
> mongo.bson.buffer.append.double(buf, "0", c(1,2,3))
[1] TRUE
> mongo.bson.buffer.finish.object(buf)
[1] TRUE
> mongo.bson.from.buffer(buf)
Fibonacci : 4
0 : 4
0 : 1   1.000000
1 : 1   2.000000
2 : 1   3.000000
```

```
> buf <- mongo.bson.buffer.create()
> mongo.bson.buffer.start.object(buf, "name")
[1] TRUE
> mongo.bson.buffer.append(buf, "first", "Jeff")
[1] TRUE
> mongo.bson.buffer.append(buf, "last", "Davis")
[1] TRUE
> mongo.bson.buffer.finish.object(buf)
[1] TRUE
> mongo.bson.buffer.append(buf, "city", "Toronto")
[1] TRUE
> mongo.bson.from.buffer(buf)
name : 3
first : 2   Jeff
last : 2   Davis

city : 2   Toronto
```

Exercise 2

- Connect to the **R**Studio Server:
  - `http://bigdata.comsysto.com:8787`
- Use **Rockmongo** to view content of mongodb:
  - `http://bigdata.comsysto.com/rockmongo`
  - User/pw : admin/admin
- Use **R** to connect and query MongoDB
  - In your home directory you can find a file "exercise2.R". Run all the commands.
  - **R** experts will find a small challenge at the end of the file.

Outline

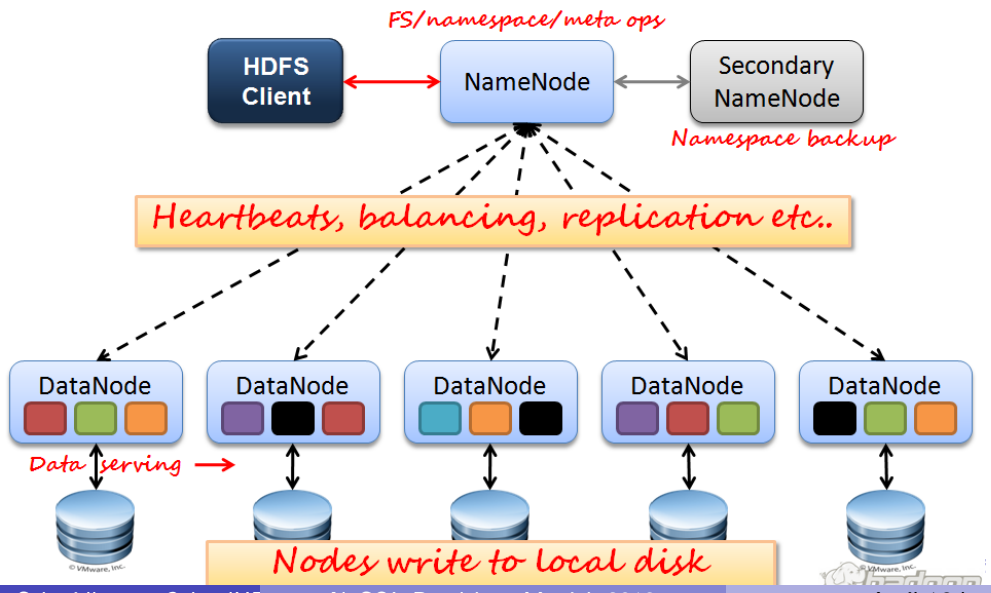
RHadoop

- an open source project sponsored by Revolution Analytics
- package overview:
  - **rmr2** hosts all MapReduce related functions
    - ★ uses Hadoop Streaming API
  - **rhdfs** for interaction with HDFS file system
  - **rhbase** connect with Hadoop's NoSQL database HBase
- installation is the biggest challenge
  - check web for installation guidelines
  - works with MapR and Cloudera Hadoop distribution
  - so far there is no AWS EMR support

`https://github.com/RevolutionAnalytics/RHadoop`

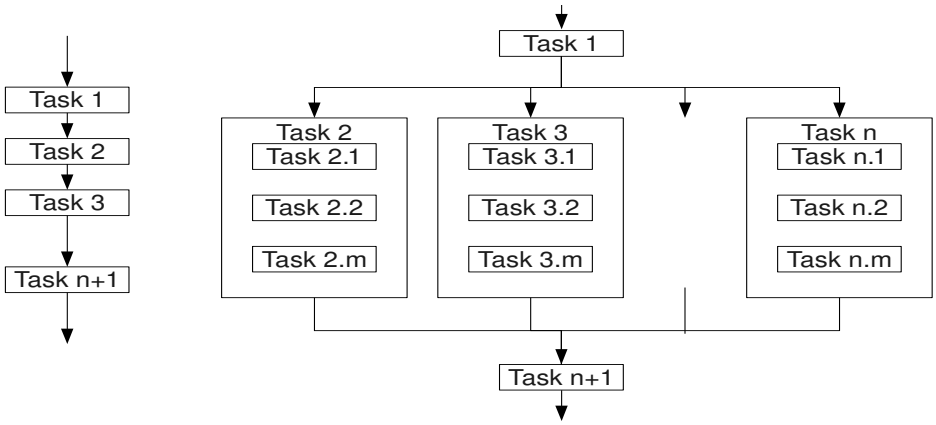
HDFS and Hadoop cluster

- 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



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**

```
> x <- c(1,2,3,4,5)
> x
[1] 1 2 3 4 5
> unlist( lapply(x, function(y) y^2) )
[1]  1  4  9 16 25
> library(parallel)
> unlist( mclapply(x, function(y) y^2) )
[1]  1  4  9 16 25
```

My first MapReduce Job

```
> library(rmr2)
> small.ints <- to.dfs(1:1000)
> out <- mapreduce(
+     input = small.ints,
+     map = function(k, v) cbind(v, v^2))
> df <- as.data.frame(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(1:1000)
> out <- mapreduce(
+     input = small.ints,
+     map = function(k, v) cbind(v, v^2))
> df <- as.data.frame(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 mapreduce case


```
> library(rmr2)
> small.ints <- to.dfs(1:1000)
> out <- mapreduce(
+     input = small.ints,
+     map = function(k, v) cbind(v, v^2))
> df <- as.data.frame(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

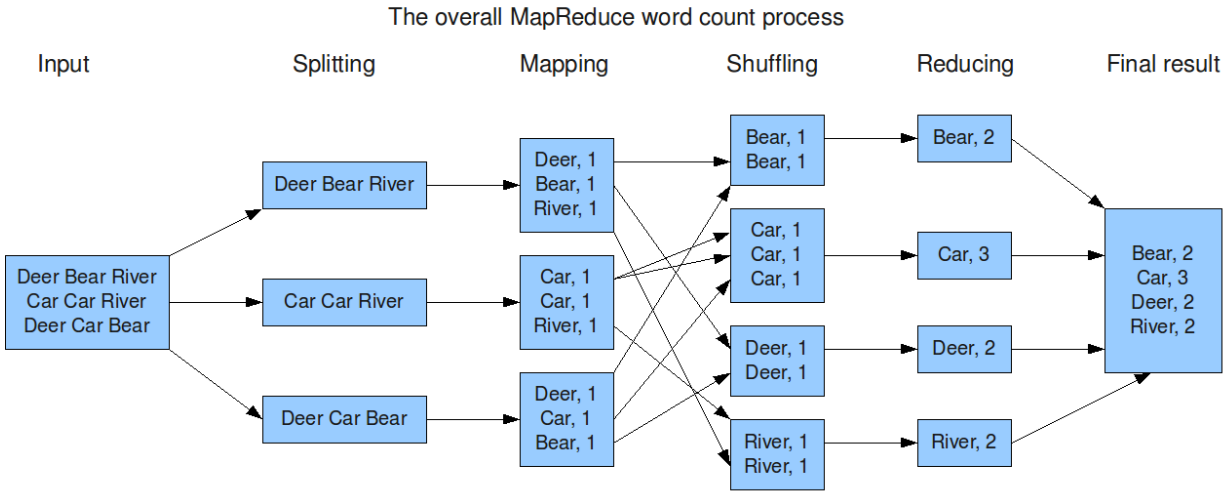
another MapReduce Job

```
> groups <- rbinom(32, n = 50, prob = 0.4)
> groups[1:10]
 [1]  9 16 11 13 12 15  9 14 14 14
> tapply(groups, groups, length)
 7  9 10 11 12 13 14 15 16 19
 1  8  3  4  5  7 12  3  5  2
```

- creates a sample from the binomial and counts how many times each outcome occurred

```
>   groups = to.dfs(groups)
>   from.dfs(
+     mapreduce(
+       input = groups,
+       map = function(., v) keyval(v, 1),
+       reduce =
+         function(k, vv)
+           keyval(k, length(vv))))
```

Hadoop Hello World - Wordcount



```
> wc.map =
+       function(., lines) {
+         keyval(
+           unlist(
+             strsplit(
+               x = lines,
+               split = pattern)),
+           1)}
> wc.reduce =
+     function(word, counts ) {
+         keyval(word, sum(counts))}
```


```
> sourceFile <- "/etc/passwd"
> tempFile <- "/tmp/wordcount-test"
> file.copy(sourceFile, tempFile)
> pattern <- " +"
> out <- mapreduce(
+       input = tempFile ,
+       input.format = "text",
+       map = wc.map,
+       reduce = wc.reduce,
+       combine = T)
> res <- from.dfs(out)
```

Hadoop environments variables

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

```
> # Sys.setenv(HADOOP_HOME="/usr/lib/hadoop")
> Sys.setenv(HADOOP_CMD="/usr/bin/hadoop")
> Sys.setenv(HADOOP_STREAMING="/usr/lib/hadoop-0.20-mapreduce/cont
> library(rmr2)
> small.ints <- to.dfs(1:1000)
> out <- mapreduce(
+     input = small.ints,
+     map = function(k, v) cbind(v, v^2))
> df <- as.data.frame(from.dfs(out))
```

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

```
> Sys.setenv(HADOOP_CMD="/usr/bin/hadoop")
> library(rhdfs)
> hdfs.init()
> hdfs.ls()
> data <- 1:1000
> filename <- "my_smart_unique_name"
> file <- hdfs.file(filename, "w")
> hdfs.write(data, file)
> hdfs.close(file)
```

RHadoop advanced

- **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 **hive**
- Want to learn more? Check Revolution Webinars

Exercise 3

- Connect to the **R**Studio Server:
  - `http://bigdata.comsysto.com:8787`
- Use **R** to run MapReduce jobs and explore HDFS
  - In your home directory you can find a file "exercise3.R". Run all the commands.
  - **R** experts will find a small challenge at the end of the file.

Summary

- **R** is a powerfull statistical tool to analyse many different kind of data: from small to big
- **R** can access databases
  - MongoDB and **rmongodb** support big data
- **R** can run Hadoop jobs
  - **rmr2** runs your MapReduce jobs
- **R** is open source and there is a lot of community driven development

`http://www.r-project.org/`

the tutorial did not cover ...

- parallel computing with **R** (Package **parallel**)
- big data in memory with **R** (Packages **bigmemory**, **bigmatrix**)
- visualizing big data with **R** (Packages **ggplot**, **lattice**, **bigvis**)
- the commercial **R** version: Revolution **R** Enterprise

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?
  - two day **R** beginners training (October 2013)
  - one day **R** Data Handling and Graphics (**plyr**, **ggplot**, **shiny**, ...) (November 2013)
  - one day **R** Big Data Analyses (November 2013)
  - registration: office@comsysto.com

Goodbye

- thanks a lot for your attention
- please feel free to get in contact concerning **R**
- meet you in one of our Meetup groups: `http://www.meetup.com/munich-useR-group/`

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