Introduction to R

Mandy Vogel

University Leipzig

September 28, 2015

Overview

- The big plan
- R Intro
- Packages
- First Session
- Citation/License
- Objects
- Combining Data Frames
- Reading Data
- The Apply Family
  - `apply()`
  - `tapply()`
- Putting it all together

Table of Contents I

today and next week

- Today
  - preliminaries
  - objects/data structures
  - combining data frames
  - indexing
  - reading data
  - apply family
  - reading the data files
- next time
  - data manipulation with dplyr
  - dates and times with lubridate
  - stringr/tidyr
  - graphics with ggplot2
  - classical tests
  - functions - simulations understanding classical tests

Table of Contents I

What's R?

- R is a high-level language and an environment for data analysis and graphics
- influenced by S (Becker, Chamber, Wilks) and Scheme (Sussman)
- and created by Ross Ihaka and Robert Gentleman at the university of Auckland
- R is free.
- R is open source.
- R is a dialect of S system.

What R can do...

R provides a wide variety of statistical and graphical techniques including

- linear and nonlinear modelling
- classical statistical tests
- time-series analysis
- classification
- clustering and many more

What R can do...

R provides a wide variety of statistical and graphical techniques including

- linear and nonlinear modelling
- classical statistical tests
- time-series analysis
- classification
- clustering and many more

R is easily extensible, can produce publication-quality graphs including mathematical symbols; dynamic and interactive graphics are available through additional packages.

Pros

- R is free and R is open source
- there is a lot of material and books available
- there is a lot of help on the web, including developers who are active in mailing lists
- most of your problems are already solved and with a high probability the solution is available from one of the repositories (as package)
- there are a lot of intuitive GUIs
- the language is easy to learn and also intuitive
- the graphics capabilities are impressive

R is Different

- R is a little different from other packages for statistical analysis
- these differences make R very powerful, but for new users they can sometimes be confusing - that's normal!

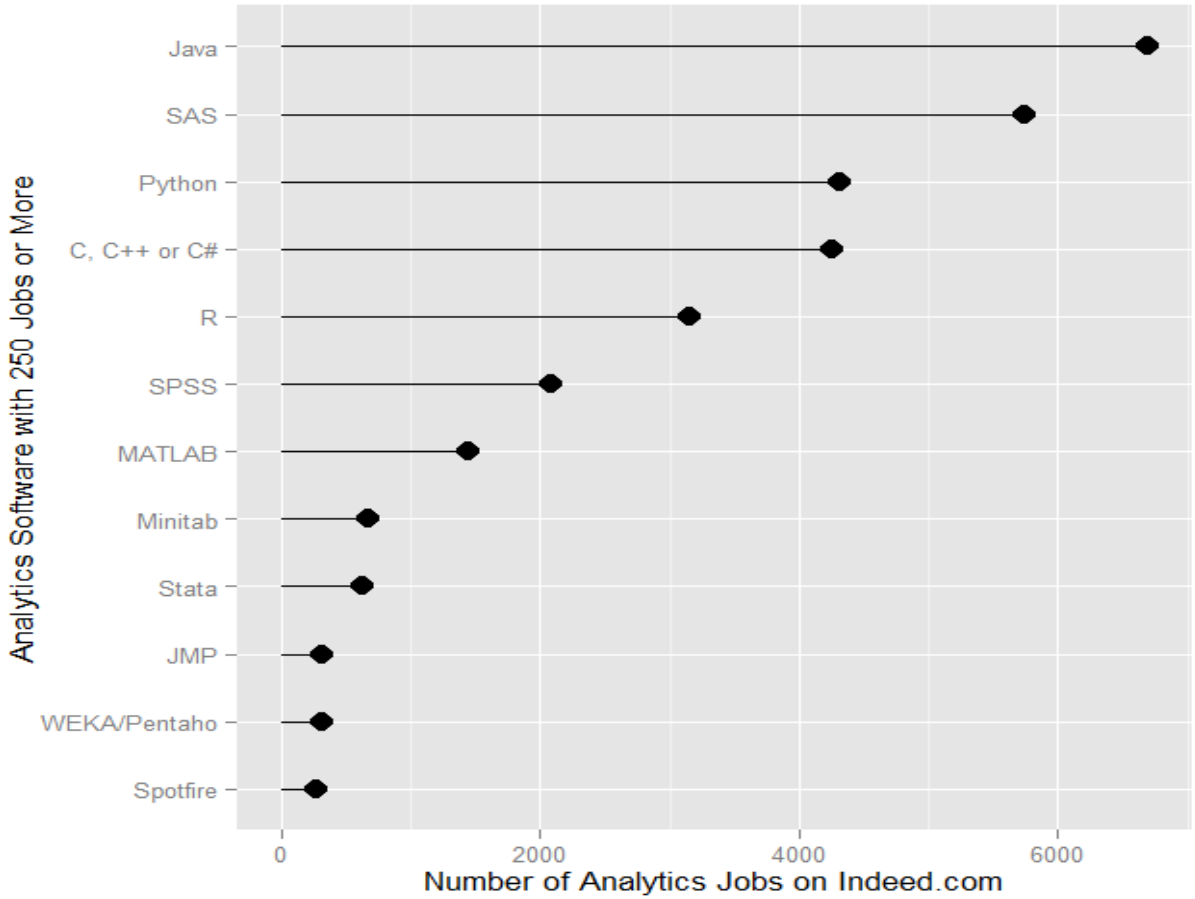
Nothing is lost or hidden

- statistical packages provide canned procedures to address common statistical problems
- canned procedures are useful for routine analysis, but they are also limiting - you can only do what the programmer lets you do
- in R, the result of statistical calculation are always accessible, so
  - you can use them for further calculations
  - you can always see how calculations were done

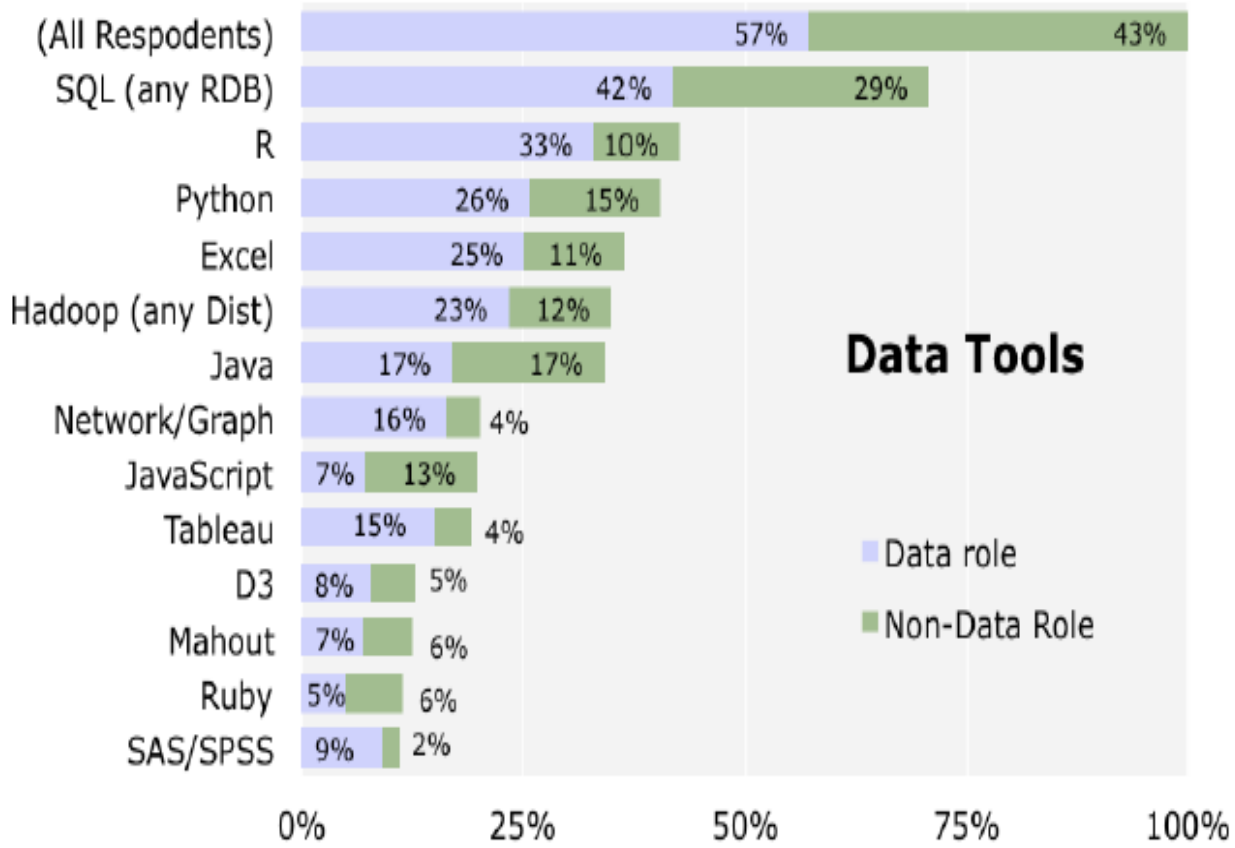
Cons

- there is a LOT of help on the web
- with a high probability there is more than one solution for your problem
- there are a lot of intuitive GUIs so you have to decide what you want (so first you have to know what you want)
- the real power of R (i.e. high flexibility) is not entirely available through GUIs
- and therefore the learning curve can be lengthy in the beginning (but soon accelerating ;)

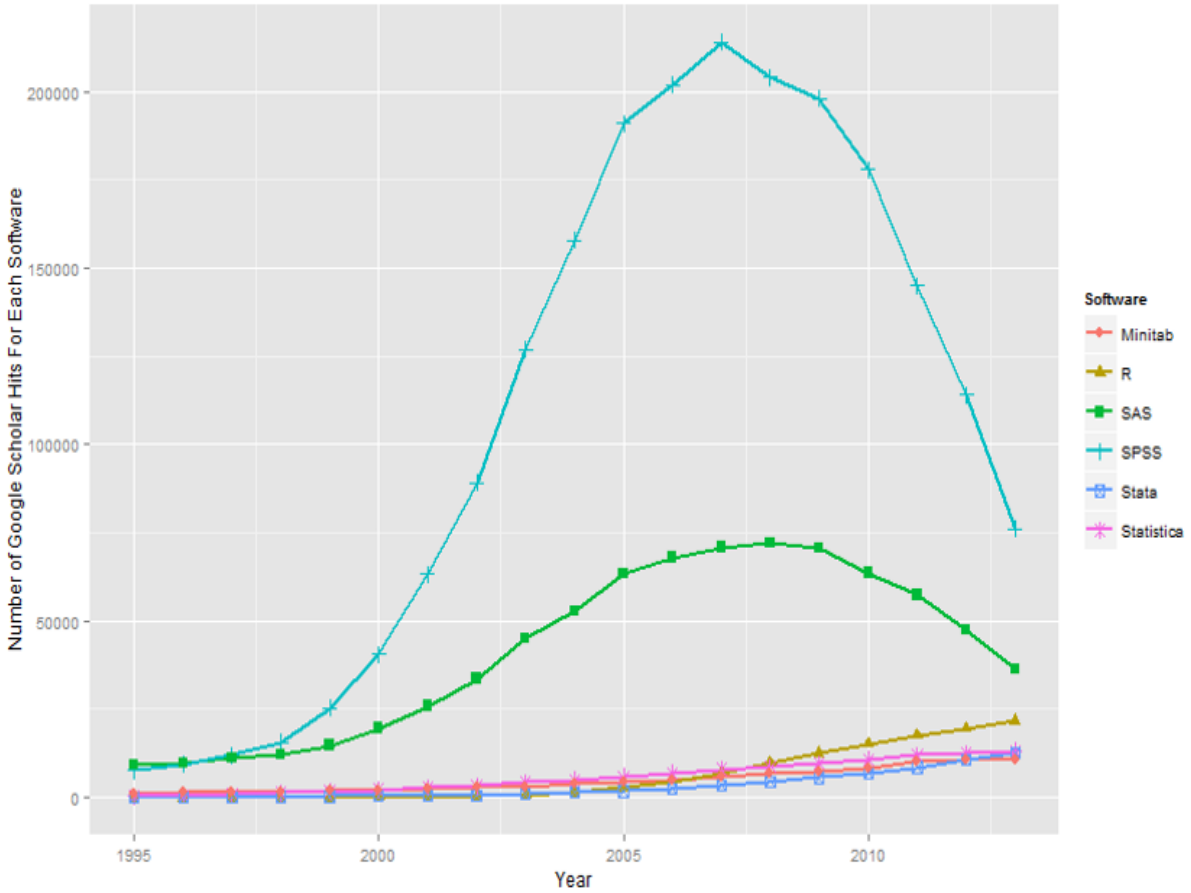
The number of analytics jobs for the more popular software (250 jobs or more, 2/2014).



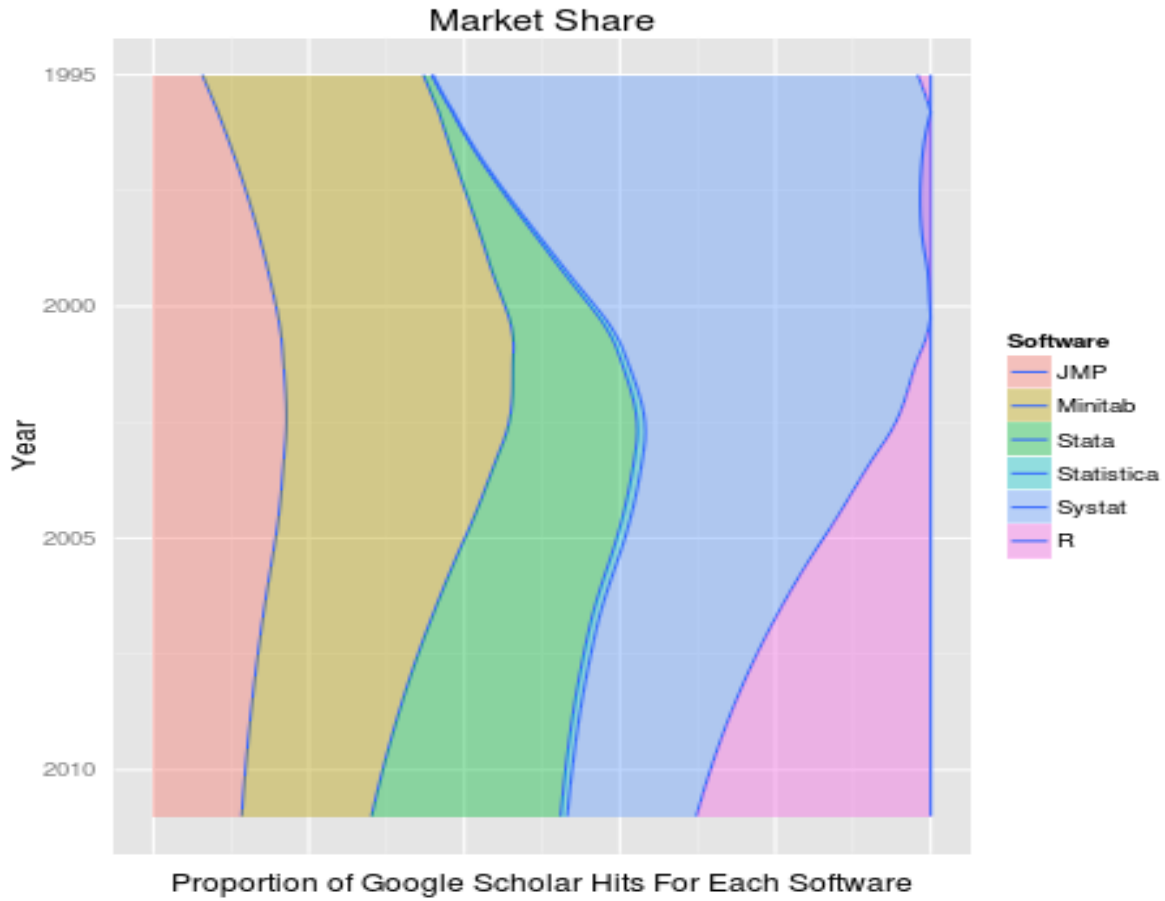
O'Reilly Data Science Survey results for 2012 and 2013 combined.



Impact of data analysis software on academic publications as measured by hits on Google Scholar



Impact of data analysis software on academic publications as measured by hits on Google Scholar



Use it!

The best way to learn R is to use it!

Where can I get it?

For the basic installation CRAN is a good place to start

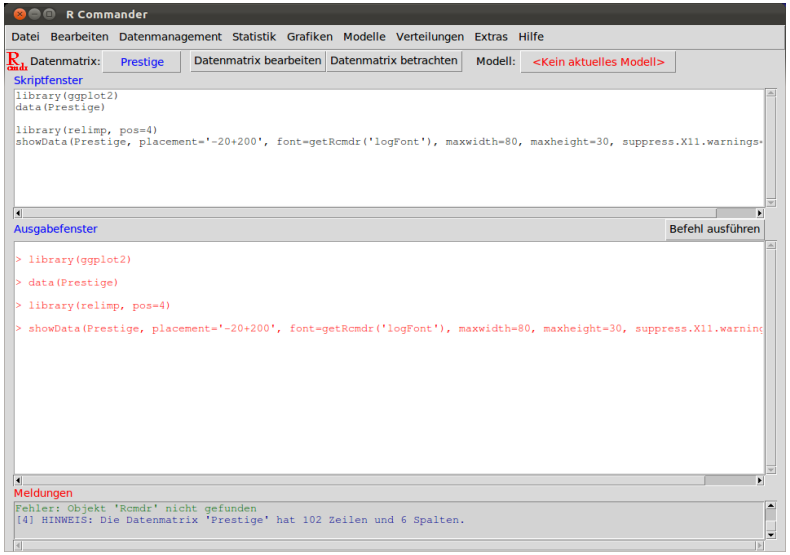
- CRAN stands for Comprehensive R Archive Network
- `http://cran.r-project.org`
  - Microsoft Windows: :: http://cran.r-project.org/bin/windows/base/
  - MacOS: :: http://cran.r-project.org/bin/macosx/
  - Linux: :: http://cran.r-project.org/bin/linux/
- for mac and pc users: just download and install the precompiled binaries
- for ubuntu users: add
  `deb http://ftp5.gwdg.de/pub/misc/cran/bin/linux/ubuntu vivid/`
  to
  `/etc/apt/sources.list`;
  detailed howto:
  `http://cran.r-project.org/bin/linux/ubuntu/`
- `https://cran.r-project.org/mirrors.html`
- `https://cran.r-project.org/mirmon_report.html`

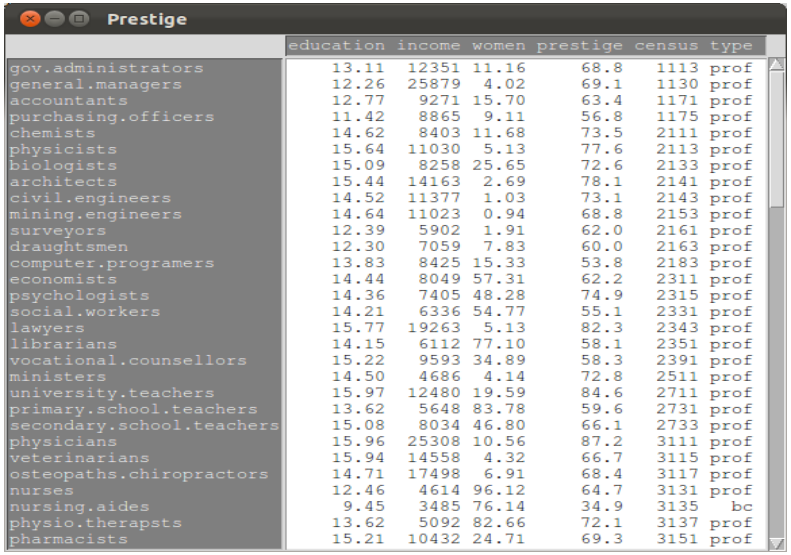
The R-Commander

The R commander, developed by John Fox is a complete GUI for R. It is implemented in the package `Rcmdr`:

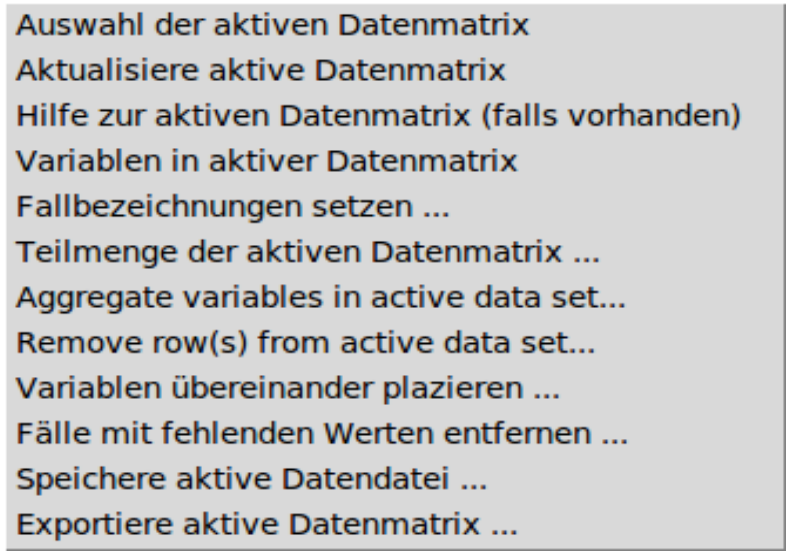
- `Rcmdr` has a comprehensive menu, which includes data reading, summaries, statistical analyses, etc.
- When the menu is activated, the `Rcmdr` will generate an R script. This script can be used as a log for documentation or for self learning.
- It has excellent graphical tools.

The R-Commander





	education	income	women	prestige	census	type
gov.administrators	13.11	12351	11.16	68.8	1113	prof
general.managers	12.26	25879	4.02	69.1	1130	prof
accountants	12.77	9271	15.70	63.4	1171	prof
purchasing.officers	11.42	8865	9.11	56.8	1175	prof
chemists	14.62	8403	11.68	73.5	2111	prof
physicists	15.64	11030	5.13	77.6	2113	prof
biologists	15.09	8258	25.65	72.6	2133	prof
architects	15.44	14163	2.69	78.1	2141	prof
civil.engineers	14.52	11377	1.03	73.1	2143	prof
mining.engineers	14.64	11023	0.94	68.8	2153	prof
surveyors	12.39	5902	1.91	62.0	2161	prof
draughtsmen	12.30	7059	7.83	60.0	2163	prof
computer.programers	13.83	8425	15.33	53.8	2183	prof
economists	14.44	8049	57.31	62.2	2311	prof
psychologists	14.36	7405	48.28	74.9	2315	prof
social.workers	14.21	6336	54.77	55.1	2331	prof
lawyers	15.77	19263	5.13	82.3	2343	prof
librarians	14.15	6112	77.10	58.1	2351	prof
vocational.counsellors	15.22	9593	34.89	58.3	2391	prof
ministers	14.50	4686	4.14	72.8	2511	prof
university.teachers	15.97	12480	19.59	84.6	2711	prof
primary.school.teachers	13.62	5648	83.78	59.6	2731	prof
secondary.school.teachers	15.08	8034	46.80	66.1	2733	prof
physicians	15.96	25308	10.56	87.2	3111	prof
veterinarians	15.94	14558	4.32	66.7	3115	prof
osteopaths.chiropractors	14.71	17498	6.91	68.4	3117	prof
nurses	12.46	4614	96.12	64.7	3131	prof
nursing.aides	9.45	3485	76.14	34.9	3135	bc
physio.therapsts	13.62	5092	82.66	72.1	3137	prof
pharmacists	15.21	10432	24.71	69.3	3151	prof



The R-Commander

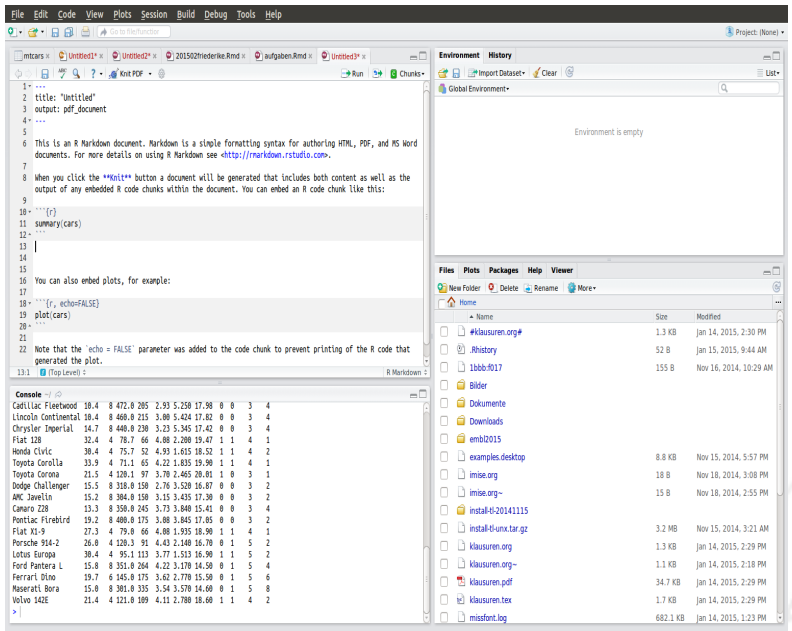
To install `Rcmdr` go to Packages $\rightarrow$ Install package(s) (or simply type `install.packages("Rcmdr")`), then choose a CRAN mirror close to you, than OK. A window with a list of packages will pop-up, on this list choose `Rcmdr` and OK. A bundle of packages will be automatically installed.
To run the "R Commander" GUI type at the prompt line:

```
> library(Rcmdr)
```

This will start a GUI similar to other statistical software. Therefore, any typical process, like read data, produce plots, make statistical analyses, etc. will be made by clicking the appropriate menu.

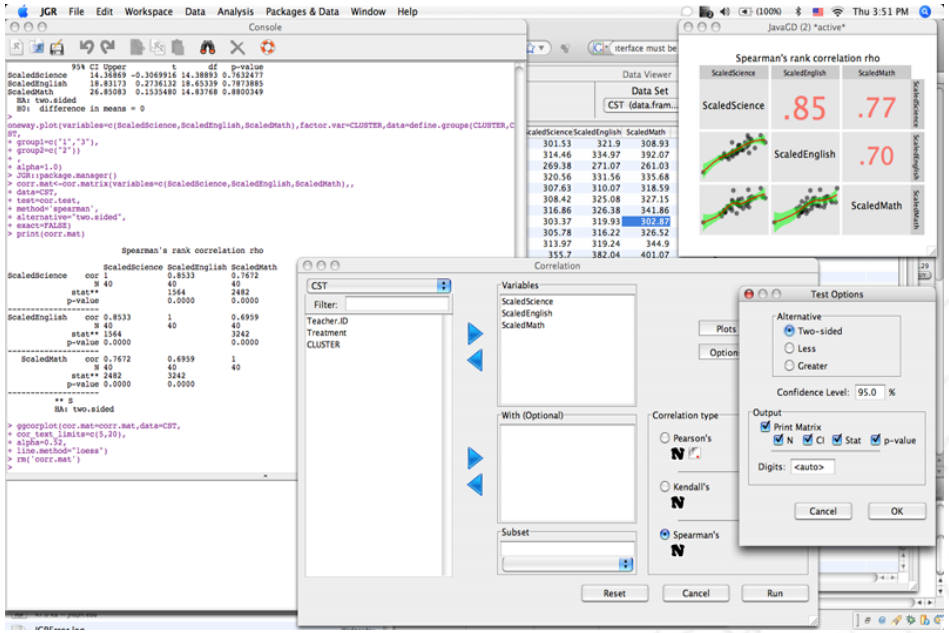
Getting R-Studio

RStudio is a free and open source integrated development environment (IDE) for R. You can run it on your desktop (Windows, Mac, or Linux) or even over the web using RStudio Server. Available at `http://rstudio.org/` (install R first)



Getting Deducer

Deducer is designed to be a free easy-to-use alternative to proprietary data analysis software such as SPSS and Minitab. It has a menu system to perform common data manipulation and analysis tasks, and an excel-like spreadsheet in which to view and edit data frames. Available at `http://www.deducer.org`; for Windows there is a all-in-one installer (incl. R)



Deducer Ubuntu

There were some problems with color setting in the plot builder when they are defined via the GUI. They are fixed in the recent version (0.6-3)

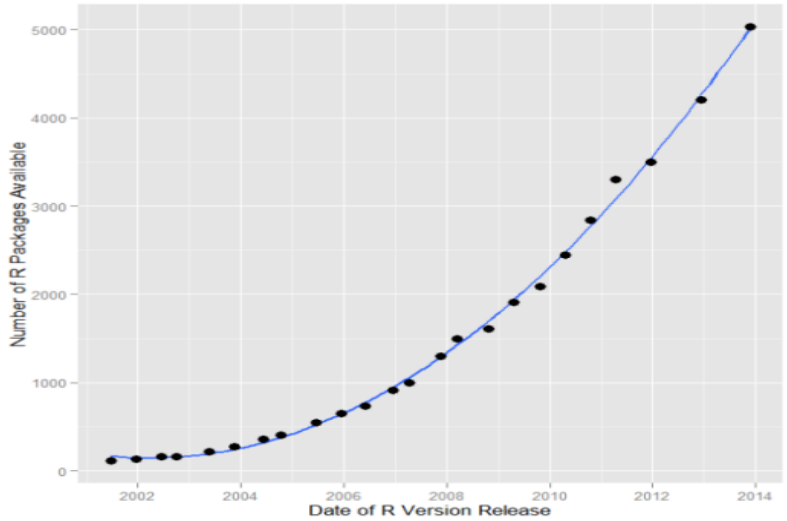
- if you haven't installed R yet, first install r-base-dev r-recommended (`sudo apt-get install r-base-dev r-recommended`)
- Open R and at the prompt enter: `install.packages(c("JGR","Deducer"))`
- Run `JGR()` to open JGR, and `library(Deducer)` to load Deducer

Table of Contents I

Packages

The capabilities of R are extended through user-created packages, which allow specialized statistical techniques, graphical devices, import/export capabilities, reporting tools, etc.

These packages are developed primarily in R, and sometimes in Java, C and Fortran. A core set of packages is included with the installation of R, 4300 (as of March 2011) with more than 7100 + 1000 (BioC) + 1100 (BioC Annotation/Exp.) (as of September 2015) available at the Comprehensive R Archive Network (CRAN), Bioconductor, and other repositories.



R taskviews

- you can google your problem or you use `http://www.rseek.org/` or `http://www.rdocumentation.org/` instead of `www.google.com`
- `http://cran.r-project.org/web/views/`
- before you install a new package: `help.search()` allows for searching the help system for documentation matching a given character string in the (file) name, alias, title, concept or keyword entries (or any combination thereof), using either fuzzy matching or regular expression matching.(installed help system)
- there are many blogs or forums, a very popular is `http://stackoverflow.com/` or `http://stackexchange.com/` (they are helpful for all other statistical packages as well)

Packages

- An R installation contains a library of packages. Some of these packages are part of the basic installation. These packages have the recommended status.
- others can be downloaded from CRAN or BioConductor or from other repositories
- A package is loaded into R using the `library()` or the `require()` command. For example to load the survival package you should enter

  ```
  > library(survival)
  ```

- you need to reload a package when you start a new R session

Getting Packages

- You can download a package from CRAN and install it by using the package menu.
- Another effective way to download and install a package is by command line. For example the following line install the R commander package with all its dependencies:

```
> install.packages("Rcmdr", dependencies=TRUE)
```

- bioconductor packages have their own installation routine, which is documented on the website `http://www.bioconductor.org/`

Software Container

- if you know about docker: there are R and BioConductor Docker container available:
  - `https://github.com/rocker-org/rstudio-daily`
  - `https://www.bioconductor.org/help/docker/`

Table of Contents I

First Session I

- start RStudio
- choose your working directory (via a menu or by typing `setwd('/your/Rworkdirectory/')`)
- R works fundamentally by the question-and-answer model: you enter a line with a command and press Enter (↩). Then the program dœs something and prints or stores the results. Then it asks for more input. When R is ready for input, it prints out its prompt, a ”>”. if you see a “+” R waits for you to end to line; with ESC you can go back to ”>”
- It is possible to use R as a text-only application, and also in batch mode (`Rscript skript.r arg1 arg2`)

The Workspace

- During a session you create a workspace. The workspace contains all variables created during the session
- for example typing

  ```
  > x <- rnorm(100, mean=2, sd=4)
  ```

  creates a variable `x` containing a vector with 100 random numbers normal distributed with mean 2 and standard deviation 4
- to see the contents of a variable just type its name

  ```
  > x
   [1]  2.663558 2.187709 -1.849147
                5.566364 2.5016523.046095  ...
  ```

Showing an Object

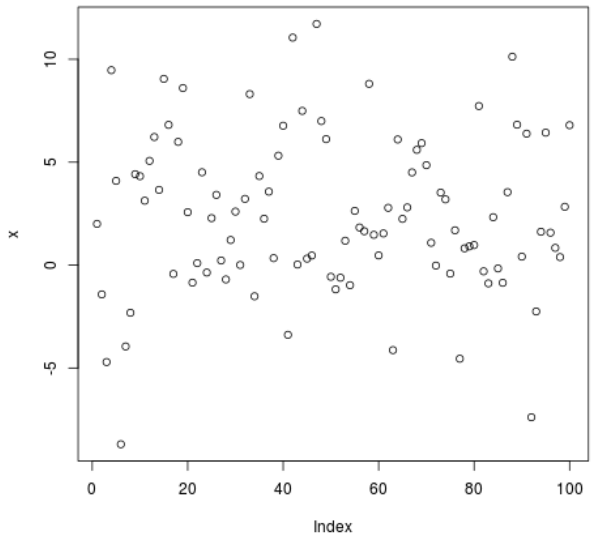
- more sophisticated R objects have print methods which do not show you the object itself but a kind of summary

```
> xx <- density(x)
> xx

Call:
density.default(x = x)

Data: x (100 obs.); Bandwidth 'bw' = 1.465

       x                   y
 Min.   :-9.711   Min.   :0.000034
 1st Qu.:-2.976   1st Qu.:0.005759
 Median : 3.759   Median :0.030555
 Mean   : 3.759   Mean   :0.037080
 3rd Qu.:10.495   3rd Qu.:0.068267
 Max.   :17.230   Max.   :0.088606
```

First Plot

- To plot the values contained in x type

```
> plot(x)
```



ls()/objects()

- All variables, functions and diverse objects can be seen by typing `ls()` and the newer, more verbose version of it `objects()` function. Thus in our example we will have

```
> ls()
[1] "x"  "xx"
```

- in R studio you see the content of the workspace in the Environment tab

`library()/require()`

- we have seen the use of `library()` to load a package
- typing `library()` without argument gives you a list of installed packages per installation path
- a more detailed list of installed packages including path as well but also a lot of additional information can be achieved by `installed.packages()`

Quitting

- quitting R is done with the `q()` function

  ```
  > q()
  ```

  at the command prompt. You will be asked to save your "workspace image".
- if you save the work space, all R objects can be reloaded in a new session, but be careful. It is recommended to rather save data explicitly

Help

- Entering the command

  ```
  > help.start()
  ```

  at the command line, will launch an extensive online help that can be read using a Web browser such as Firefox or Internet Explorer. Another way to access to these "help" pages is by the menu bar on Windows. Notice that the HTML version of the help system has a very useful "Search Engine and Keywords".
- typing `?command` gives the help for a specific command

First Session

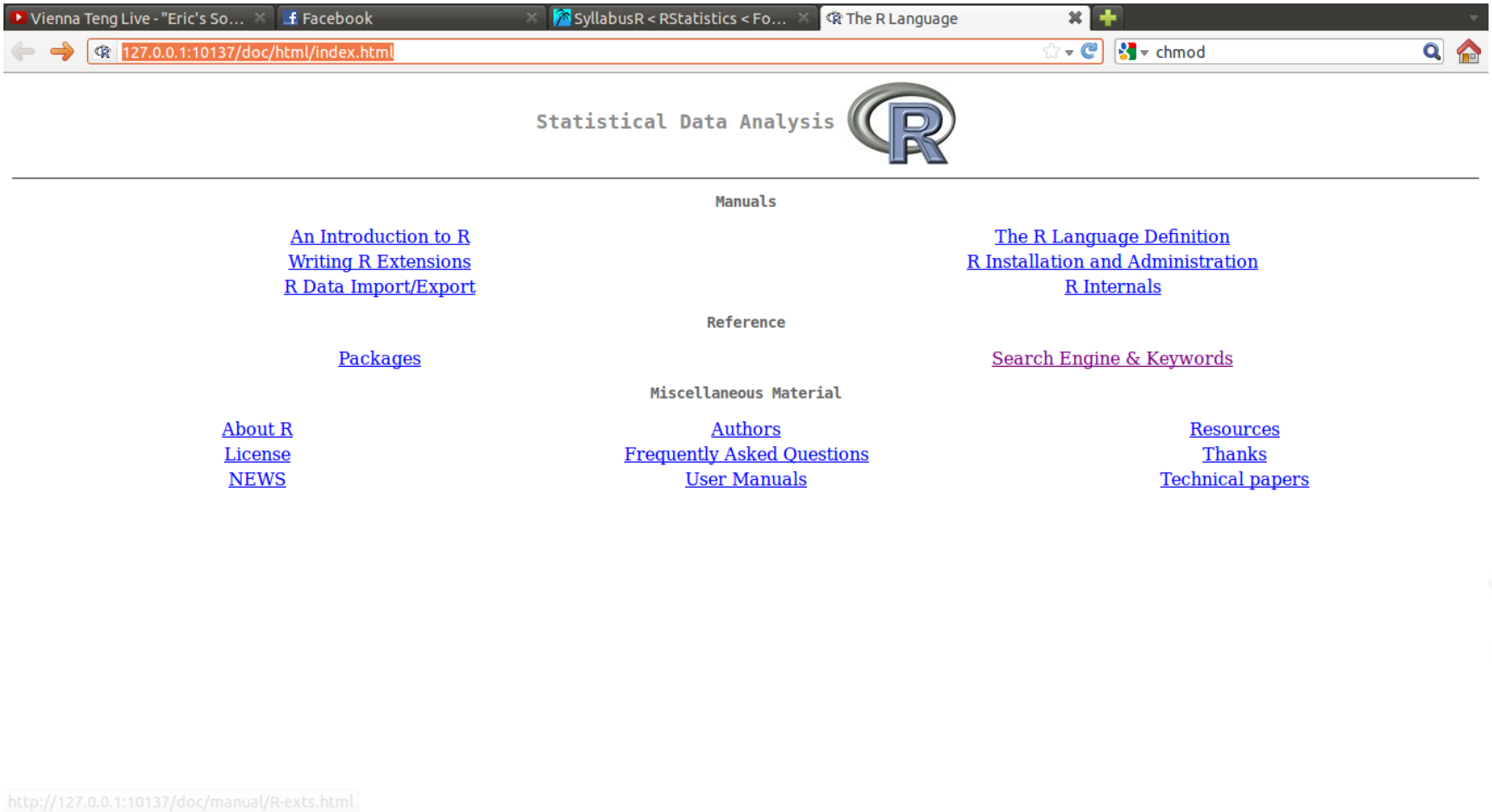


Table of Contents I

Citation

Input

```
> citation()
```

```
To cite R in publications use:

  R Development Core Team (2012). R: A language and environment for
  statistical computing. R Foundation for Statistical Computing,
  Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

A BibTeX entry for LaTeX users is

  @Manual,
    title = R: A Language and Environment for Statistical Computing,
    author = R Development Core Team,
    organization = R Foundation for Statistical Computing,
    address = Vienna, Austria,
    year = 2012,
    note = ISBN 3-900051-07-0,
    url = http://www.R-project.org/,


We have invested a lot of time and effort in creating R, please cite it
when using it for data analysis. See also ‘citation("pkgname")’ for
citing R packages.
```

Licence

Licence

R is mainly distributed under the terms of the GNU General Public License, either Version 2, June 1991 or Version 3, June 2007. Core Bioconductor packages are typically licensed under Artistic-2.0. You get detailed information with: `license()`, `RShowDoc("COPYING")`, `packageDescription("packagename")$License`

Table of Contents I

Data Structures

- R's base data structures can be organized by
  - their dimensionality and
  - whether they are homogeneous or heterogeneous

	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data frame
nd	Array	

`str()`

- given an object, the best way to understand its structure it's the `str()` command
- `str()` is short for structure and gives a compact, human readable description of any R data structure

str()

Input/Output

```
> require(ggplot2)
> example(scale_colour_brewer)
...
> str(dsamp)
'data.frame': 1000 obs. of  10 variables:
 $ carat  : num  2.02 1.01 0.52 1.57 0.82 1.17 0.78 1.05 0.28 0.74 ...
 $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 3 5 5 4 5 3 4 4 5 3 ...
 $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 5 4 4 6 1 4 6 1 2 2 .
 $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 3 3 2 4 3 2 4 3 6 4
 $ depth  : num  61.1 63 59.7 62.5 61.6 61.2 62.7 60.5 62.4 61.8 ...
 $ table  : num  58 57 56 58 57 58 58 60 56 58 ...
 $ price  : int  18236 5555 1042 9847 4135 5595 2646 5762 828 3180 ...
 $ x      : num  8.07 6.37 5.28 7.5 6.02 6.84 5.89 6.62 4.2 5.79 ...
 $ y      : num  8.16 6.4 5.31 7.42 6.06 6.86 5.82 6.53 4.16 5.82 ...
 $ z      : num  4.96 4.02 3.16 4.66 3.72 4.19 3.68 3.98 2.61 3.59 ...
```

str()

Input/Output

```
> str(d)
List of 9
 $ data       :'data.frame': 1000 obs. of  10 variables:
  ..$ carat  : num [1:1000] 2.02 1.01 0.52 1.57 0.82 1.17 0.78
...
 $ mapping    :List of 3
  ..$ colour: symbol clarity
  ..$ x      : symbol carat
  ..$ y      : symbol price
...
 $ coordinates:List of 1
  ..$ limits:List of 2
  .. ..$ x: NULL
  .. ..$ y: NULL
  ..- attr(*, "class")= chr [1:2] "cartesian" "coord"
...
```

Vectors

- R's basic data structure
- three common properties:
  - type (`typeof()`)
  - lenth (`length()`)
  - attributes (`attributes()` - optional)
- there are four common types of atomic vectors:
  - logical
  - integer
  - double (numeric)
  - character
- usually created using `c()` (for concatenate/combine)
- flat structure

Types/Cœrcion

- while combining two or more vectors of different types, the will be cœrced to the most flexible type
- types from least to most flexible are: logical, integer, double, character

Cœrcion - Exercises

- Test your understanding of vector cœrcion rules by predicting the output of the following uses of c():

Input

```
> c(1, FALSE)
> c("a", 1)
> c(list(1), "a")
> c(TRUE, 1L)
```

- Why is 1 == "1" true? Why is -1 < FALSE true? Why is "one" < 2 false?
-

length

- should be self explanatory

attributes

The most important are:

- names
- dimensions
- class

factors

A factor is a vector that can contain only predefined values (categorical variable).

- built on top of integer values with the following attributes:
  - class: factor
  - levels: set of allowed values
  - labels

factors - Exercise

- What happens to a factor when you modify its levels? (Hint: use `summary()` and `as.numeric()`)

Input

```
> f1 <- factor(letters)
> levels(f1) <- rev(levels(f1))
> f2 <- rev(factor(letters))
> f3 <- factor(letters, levels = rev(letters))
> f4 <- c(f2,f3)
```

Lists

- lists are different from atomic vectors because their elements can be of any type
- constructing lists is done by `list()`
- they can be recursive, i.e. lists can contain lists can contain lists etc
- they can be combined by `c()`

Data Frames

- most common way to store data in R
- under the hood a data frame is a list of equal-length vectors
- so it is a 2-dimensional structure
- the `length()` of a data frame is the length of the underlying list (i.e. the number of columns)
- `ncol()`, `nrow()`, `dim()`, `names()`, `rownames()`, `colnames()`
- constructing is done by `data.frame()`

Table of Contents I

rbind()

- `rbind()` can be used to combine two dataframes (or matrices) in the sense of adding rows, the column names and types must be the same for the two objects

Input/Output

```
> x <- data.frame(id=1:3,score=rnorm(3))
> y <- data.frame(id=13:15,score=rnorm(3))
> rbind(x,y)
   id       score
1   1  0.71121163
2   2 -0.62973249
3   3  1.17737595
4  13 -0.45074940
5  14 -0.01044197
6  15 -1.05217176
```

cbind()

- `cbind()` can be used to combine two dataframes (or matrices) in the sense of adding columns, the number of rows must be the same for the two objects

Input/Output

```
> cbind(x,y)
  id      score1      score2     score3
1  1  0.11440705  0.14536778 -1.1773241
2  2 -1.62862651  0.02020604  0.5686415
3  3  0.05335811  0.25462270  0.8844987
4  4 -0.19931734  0.15625511  0.9287316
5  5 -1.15217836 -1.79804503 -0.7550234
```

- it is not recommended to use `cbind()` to combining data frames

merge() I

- `merge()` is the command of choice for merging or joining data frames
- it is the equivalent of join in sql
- there are four cases
  - inner join
  - left outer join
  - right outer join
  - full outer join

merge() II

Input/Output

```
> (d1 <- data.frame(id=LETTERS[c(1,2,3)],day1=sample(10,3
  id day1
1  A    3
2  B    4
3  C    5
> (d2 <- data.frame(id=LETTERS[c(1,3,5,6)],day2=sample(10
  id day2
1  A    7
2  C   10
3  E    3
4  F    6
```

inner join

- inner join means: keep only the cases present in both of the data frames

Input/Output

```
> merge(d1,d2)
  id day1 day2
1  A    3    7
2  C    5   10
```

left outer join

- left outer join means: keep all cases of the left data frame no matter if they are present in the right data frame (`all.x=T`)

Input/Output

```
> merge(d1,d2,all.x = T)
  id day1 day2
1  A    3    7
2  B    4   NA
3  C    5   10
```

right outer join

- right outer join means: keep all cases of the right data frame no matter if they are present in the left data frame (`all.y=T`)

Input/Output

```
> merge(d1,d2,all.y = T)
  id day1 day2
1  A    3    7
2  C    5   10
3  E   NA    3
4  F   NA    6
```

full outer join

- full outer join means: keep all cases of both data frames (`all=T`)

Input/Output

```
> merge(d1,d2,all = T)
  id day1 day2
1  A    3    7
2  B    4   NA
3  C    5   10
4  E   NA    3
5  F   NA    6
```

merge()

- if not stated otherwise R uses the intersect of the names of both data frames, in our case only id
- you can specify these columns directly by `by=c("colname1","colname2")` if the columns are named identical or
- using `by.x=c("colname1.x","colname2.x"), by.y=c("colname1.y","colname2.y")` if they have different names in the data frames

merge()

- now read in the file personendaten.txt using the appropriate command
- join the demographics with our pre1 data frame (even though it does not make sense now)

`Reduce()`

- is a higher order function (functional)
- `Reduce()` uses a binary function (like `rbind()` or `merge()`) to combine successively the elements of a given list
- it can be used if you have not only two but many data frames

Reduce()

- first we make up 4 artifical data frames

Reduce()

Input/Output

```
> (d1 <- data.frame(id=LETTERS[c(1,2,3)],day1=sample(10,3)))
  id day1
1  A    3
2  B    1
3  C    7
> (d2 <- data.frame(id=LETTERS[c(1,3,5,6)],day2=sample(10,4)))
  id day2
1  A    8
2  C    2
3  E    5
4  F    3
> (d3 <- data.frame(id=LETTERS[c(2,4:6)],day3=sample(10,4)))
  id day3
1  B    8
2  D    3
3  E    4
4  F   10
> (d4 <- data.frame(id=LETTERS[c(1:5)],day4=sample(10,5)))
  id day4
1  A    2
2  B    7
3  C    8
4  D    9
5  E    1
```

Reduce()

- now we use `Reduce()` in combination with `merge()`

Input/Output

```
> Reduce(merge,list(d1,d2,d3,d4))
[1] id   day1 day2 day3 day4
<0 Zeilen> (oder row.names mit Länge 0)
```

- and what we get is an empty data frame
- well this isn't exactly what we wanted, so why?
- it is because the default behavior of `merge()` is set `all=F`, so we get only complete lines which is in this case - none
- so we have to define a wrapper function which only change this argument to `all=T`

Reduce()

- now we use `Reduce()` in combination with `merge()`

Input/Output

```
> Reduce(function(x,y) { merge(x,y, all=T) },
+        list(d1,d2,d3,d4))
  id day1 day2 day3 day4
1  A    3    8   NA    2
2  B    1   NA    8    7
3  C    7    2   NA    8
4  E   NA    5    4    1
5  F   NA    3   10   NA
6  D   NA   NA    3    9
```

- which is exactly what we want

Reduce()

- a second example in combination with `rbind()`

Input/Output

```
> d4$day <- names(d4)[2]
> names(d4)[2] <- "score"
> Reduce(function(x,y) { y$day <- names(y)[2]
+                        names(y)[2] <- "score"
+                        rbind(x,y) } ,
+        list(d1,d2,d3), init = d4)
   id score  day
1   A     2 day4
2   B     7 day4
3   C     8 day4
4   D     9 day4
...
```

- which is exactly what we want

Excercises - Merging

- in the `rstuff.r` file you find the code to create three data frames: `m1`, `m2` and `subjdata`
- combine these three data sets using `merge()`

Indexing with Positive Integers

- there are circumstances where we want to select only some of the elements of a vector/array/dataframe/list
- this selection is done using subscripts (also known as indices)
- subscripts have square brackets [2] while functions have round brackets (2)
- Subscripts on vectors, matrices, arrays and dataframes have one set of square brackets [6], [3,4] or [2,3,2,1]
- while subscripts on lists have double square brackets [[2]] or [[i,j]]
- when a subscript appears as a blank it is understood to mean all of thus
  - `[,4]` means all rows in column 4 of an object
  - `[2,]` means all columns in row 2 of an object.

Indexing with Positive Integers

- A vector of positive integers as index:The index vector can be of any length and the result is of the same length as the index vector. For example,

Input/Output

```
> letters[1:3]
[1] "a" "b" "c"
> letters[c(1:3,1:3)]
[1] "a" "b" "c" "a" "b" "c"
>
```

Logical Indexing

- A logical vector as index: Values corresponding to T values in the index vector are selected and those corresponding to F or NA are omitted. For example,

Input/Output

```
> x<-c(1,2,3,NA)
> x[!is.na(x)]
[1] 1 2 3
```

creates a vector without missing values. Also

```
> x[is.na(x)] <- 0
> x
[1] 1 2 3 0
```

replaces the missing value by zeros.

Logical Indexing

A common operation is to select rows or columns of data frame that meet some criteria. For example, to select those rows of `painters` data frame with `Colour` ≥ 17:

Input/Output

```
> library(MASS)
> attach(painters)
> painters[Colour >=17,]
           Composition Drawing Colour Expression School
Bassano              6       8     17          0      D
Giorgione            8       9     18          4      D
Pordenone            8      14     17          5      D
...
```

Logical Indexing

We may want to select on more than one criterion. We can combine logical indices by the 'and', 'or' and 'not' operators $\&$, $|$ and !. For example,

Input/Output

```
> painters[Colour >=17 & Composition > 10, c(1,2,3)]
           Composition Drawing Colour
Titian              12      15     18
Rembrandt           15       6     17
Rubens              18      13     17
Van Dyck            15      10     17
```

List of Logical Operations

Operation	Description
!	logical NOT
&	logical AND
|	logical OR
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	logical equals (double =)
!=	not equal
&&	AND with IF
||	OR with IF
`xor(x,y)`	exclusive OR
`isTRUE(x)`	an abbreviation of identical(TRUE,x)

Logical Indexing

If we want to select a subgroup, for example those with schools A, B, and D. We can generate a logical vector using the %in% operator as follows:

Input/Output

```
> painters[School %in% c("A","C","D"),]
Da Udine            10       8      16         3       A
Da Vinci            15      16       4        14       A
Del Piombo           8      13      16         7       A
...
```

Indexing

A vector character strings with variable names can be used to extract those variables relevant for analysis. This is very useful when we have a large number of variables and we need to work with a few ones. For example,

Input/Output

```
> names(painters)
[1] "Composition" "Drawing" "Colour" "Expression" "Schoo
> painters[1:3,c("Drawing","Expression")]
           Drawing Expression
Da Udine         8          3
Da Vinci        16         14
Del Piombo      13          7
```

Indexing with Characters

- a vector of character strings could a index on a vector when the vector has names:

Input/Output

```
> x <- c(1:3,NA)
> names(x)<-letters[1:4]
> x
 a  b  c  d
 1  2  3 NA
> x[c("a","c")]
a c
1 3
```

Trimming Vectors Using Negative Indices

- an extremely useful facility is to use negative indices to drop terms from a vector
- suppose we wanted a new vector, z, to contain everything but the first element of x

Input/Output

```
> x<- c(5,8,6,7,1,5,3)
> (z <- x[-1])
[1] 8 6 7 1 5 3
```

Indexing - Exercises

First try to understand the following commands:

Input

```
> x <- c(2, 7, 0, 9, 10, 23, 11, 4, 7, 8, 6, 0)
> x[4]
> x[3:5]
> x[c(1, 5, 8)]
> x[x > 10]
> x[(1:6) * 2]
> x[x == 0] <- 1
> x
> ifelse(round(x/2) == x/2, "even", "odd")
```

Indexing - Exercises

Now try the following:

1. Display every third element in x
2. Display elements that are less than 10, but greater than 4
3. Modify the vector x, replacing by 10 all values that are greater than 10
4. Modify the vector x, multiplying by 2 all elements that are smaller than 5
5. Create a new vector y with elements 0,1,0,1, . . . (12 elements) and a vector z that equals x when y=0 and 3x when y=1. (You can do it using ifelse, but there are other possibilities)

Table of Contents I

Reading Data

The most convenient way of reading data into R is via the function called `read.table()`. It requires that the data is in "ASCII format", or a "flat file" as created with Windows' NotePad or any plain-text editor. The result of `read.table()` is a data frame.

It is expected that each line of the data file corresponds to a subject information, that the variables are separated by blanks or any other separator symbol (e.g., ",", ";"). The first line of the file can contain a header (`header=T`) giving the names of the variables, which is highly recommended!

read.table()

As an example we read in the data contained in the file `fishercats.txt`

Input/Output

```
> read.table("week1/data/fishercats.txt",
+              sep=" ",header=T)
  Sex Bwt Hwt
1   F 2.0 7.0
2   F 2.0 7.4
3   F 2.0 9.5
4   F 2.1 7.2
5   F 2.1 7.3
....
```

These data correspond to the heart and body weights of samples of male and female cats (R. A. Fisher, 1947).

read.table()

The first argument corresponds to the data file, the second to the fields separator and the third `header=T` specifies that the first line is a header with variable names. Important: the character variables will be automatically read as factors. There is a variant for reading data from an url:

Input/Output

```
> winer <- read.table(
+ "http://socserv.socsci.mcmaster.ca/jfox/Courses/R/ICPSR/Wine
+ header=T)
```

read.table()

There are other variants of `read.table` function alike :

- `read.csv()` this function assumes that fields are separated by a comma instead of whites spaces
- `read.csv2()` this function assumes that the separate symbol is the semicolon, but use a comma as the decimal point (some programs, e.g., Microsoft Excel, generate this format when running in European systems)
- the function `scan()` is a powerful, but less friendly, way to read data in R; you may need it, if you want to read files with different numbers ov values per line

Reading data from the clipboard

With the function `read.delim()` or also `read.table()` it is possible to read data directly from the clipboard.
For example mark and copy some columns from an Excel spreadsheet and transfer this content to an R by

Input/Output

```
> mydata <- read.delim("clipboard",na.strings=".")
> str(mydata) # structure of the data
```

The Data Editor

To interactively edit a data frame in R you can use the edit function. For example:

Input/Output

```
> data(airquality)
> aq <-edit(airquality)
```

This brings up a spreadsheet-like editor with a column for each variable in the data frame. See `help(airquality)` for the contents of this data set. The function `edit()` leaves the original data frame unchanged, the changed data frame is assigned to `aq`. The function `fix(x)` invokes the function `edit(x)` on `x` and assign the new (edited) version of `x` to `x`

Reading Data from Other Programs

You can always use the export function from other (statistical) software to export data from other statistical systems to a tab or comma-delimited file and use the `read.table()`. However, R has some direct methods.
The `foreign` package is one of the "recommended" packages in R. It contains routines to read files from SPSS (`.sav` format), SAS (export libraries), EpiInfo (.rec), Stata, Minitab, and some S-PLUS version 3 dump files. For example

Input/Output

```
> library(foreign)
> mydata <- read.spss("test.sav", to.data.frame=T)
```

read the `test.sav` SPSS data set and convert it to a data.frame.

Reading Data from Excel Files

Input/Output

```
> library(XLConnect)
> setwd("/media/TRANSCEND/mpicbs/data/")
> my.wb <- loadWorkbook("Duncan.xls")
> sheets <- getSheets(my.wb)
> content <- readWorksheet(my.wb, sheet=1)
> head(content)
       Col0 type income education prestige
1 accountant prof     62        86       82
2     pilot prof     72        76       83
3  architect prof     75        92       90
4    author prof     55        90       76
5   chemist prof     64        86       90
6  minister prof     21        84       87
>
```

Reading Data from Excel Files

- whereas `XLConnect` is the most sophisticated R package to read and write Excel files it depends on java and is therefore a bit clumsy
- the relatively new package `readxl` does not depend on java nor on perl
- up to now two commands: `excelsheets()`, `read_excel()`

Input/Output

```
> require(readxl)
Lade nötiges Paket: readxl
> x <- read_excel("week1/data/Duncan.xls")
> head(x)
          NA type income education prestige
1 accountant prof     62        86       82
2      pilot prof     72        76       83
3  architect prof     75        92       90
4     author prof     55        90       76
```

Reading Data from Excel Files

If someone is really fond of Excel, RExcel (http://rcom.univie.ac.at/download.html) is really worth the effort. There is also a function reading MSAccess files (`mdb.get()` from the Hmisc package)

Something on Connections

The function `read.table()` opens a connection to a file, read the file, and close the connection. However, for data stored in databases, there exists a number of interface packages on CRAN.
The RODBC package can set up ODBC connections to data stored by common applications including Excel and Access (for Excel and Access RODBC dœsn't work on Unix but it is great for data base connections). There are also more general ways to build connections to data bases.
For up-to-date information on these matters, consult the "R Data Import/Export" manual that comes with the system.

Table of Contents I

Implicit Loops

A common application of loops is to apply a function to each element of a set of values and collect the results in a single structure.
In R this is done by the functions (but there is of course also `for()`):

- `lapply()`
- `sapply()`
- `apply()`
- `tapply()`

lapply()

Input/Output

```
> lapply(mtcars,mean)
$mpg
[1] 20.09062

$cyl
[1] 6.1875

$disp
[1] 230.7219

$hp
[1] 146.6875
...
```

lapply()

Input/Output

```
> sapply(mtcars,mean)
       mpg        cyl       disp         hp       drat
 20.090625   6.187500 230.721875 146.687500   3.596563   3.217
        vs         am       gear       carb
  0.437500   0.406250   3.687500   2.812500
```

apply()

- apply() this function can be applied to an array. Its argument is the array, the second the dimension/s where we want to apply a function and the third is the function. For example

Input/Output

```
> x <- 1:12
> dim(x)<-c(2,2,3)
> apply(x,3,quantile) #calculate the quantiles
     [,1] [,2]  [,3]  #for each 2x2 matrix
0%   1.00 5.00  9.00
25%  1.75 5.75  9.75
50%  2.50 6.50 10.50
75%  3.25 7.25 11.25
100% 4.00 8.00 12.00
```

tapply()

- The function `tapply()` allows you to create tables (hence the "t") of the value of a function on subgroups defined by its second argument, which can be a factor or a list of factors. For example in the `quine` data frame, we can summarize `Days` classify by `Eth` and `Lrn` as follows:

Input/Output

```
> tapply(mtcars$mpg,mtcars$cyl,mean)
       4        6        8
26.66364 19.74286 15.10000
> tapply(mtcars$mpg,list(mtcars$cyl,mtcars$vs),mean)
         0      1
4 26.00000 26.730
6 20.56667 19.125
8 15.10000     NA
```

Table of Contents I

Reading the Data

Input/Output

```
> tmp <- lapply(files,function(filename){
+         xx <- readLines(filename)
+         d1 <- read.table(text = xx[4:length(xx)],fill = T,hea
+         d2 <- read.table(text = xx[1:2],fill = T,header=T)
+         d1$subject <- rownames(d2)
+         d1$timepoint <- d2$subject
+         d1$date <- d2$timepoint
+         d1$time <- d2$date
+         d1$no.trials <- d2$no_trials
+         return(d1)
+     })
>
> system.time(result <- Reduce(rbind,tmp))
Fehler in match.names(clabs, names(xi)) :
  Namen passen nicht zu den vorhergehenden Namen
Timing stopped at: 0.01 0 0.01
```

Reading the Data

Input/Output

```
> tmp <- lapply(files,function(filename){
+         xx <- readLines(filename)
+         d1 <- read.table(text = xx[4:length(xx)],fill = T,hea
+         names(d1)[3] <- "trial"
+         d2 <- read.table(text = xx[1:2],fill = T,header=T)
+         d1$subject <- rownames(d2)
+         d1$timepoint <- d2$subject
+         d1$date <- d2$timepoint
+         d1$time <- d2$date
+         d1$no.trials <- d2$no_trials
+         return(d1)
+     })
>
> system.time(result <- Reduce(rbind,tmp))
       User      System verstrichen
     57.812       0.017      57.765
```