R introduction

Mandy Vogel

University Leipzig

September 6, 2015

Overview

Table of Contents I

Table of Contents II

Table of Contents III

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

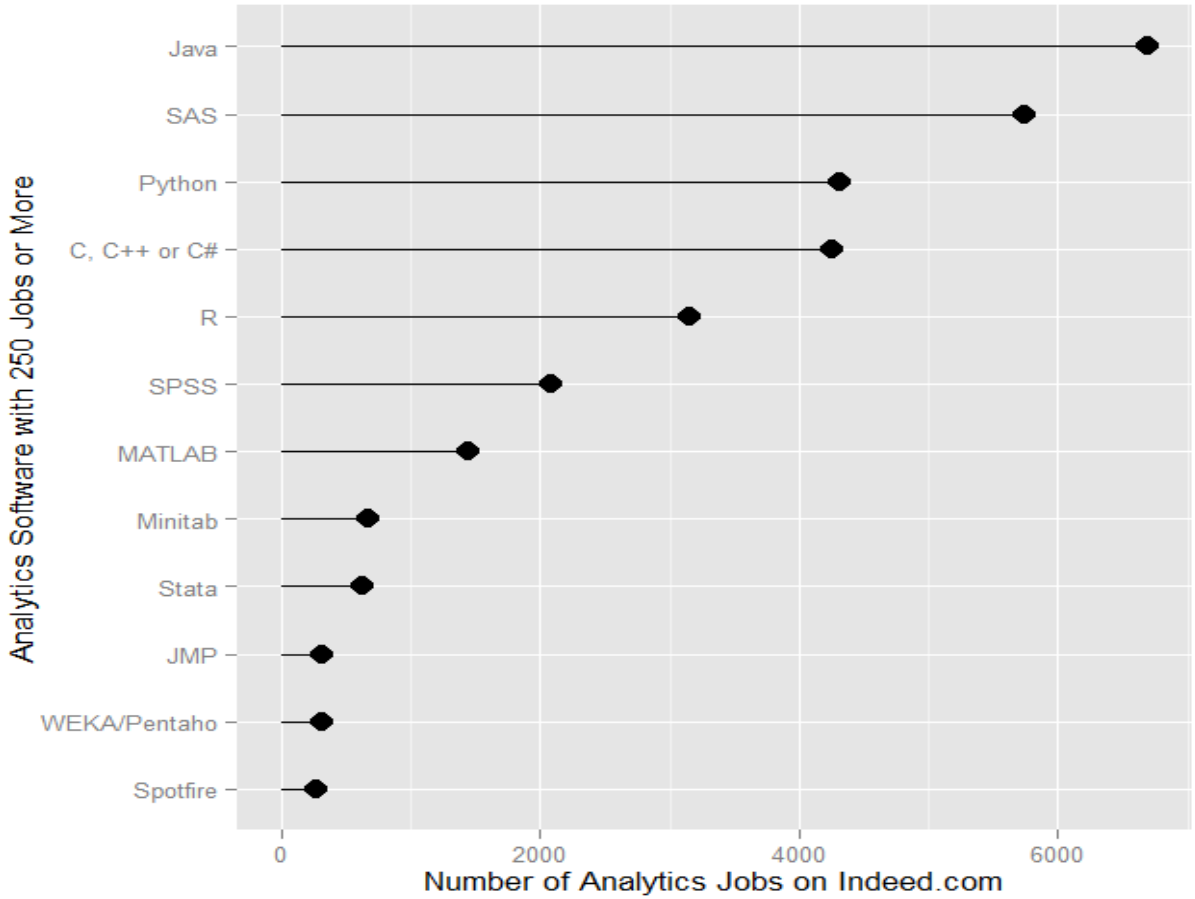
R is Different

- 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 is normal!
- so I hope this course help you up the initial curve so that you can be comfortable with R because - R is great, those who know it love it

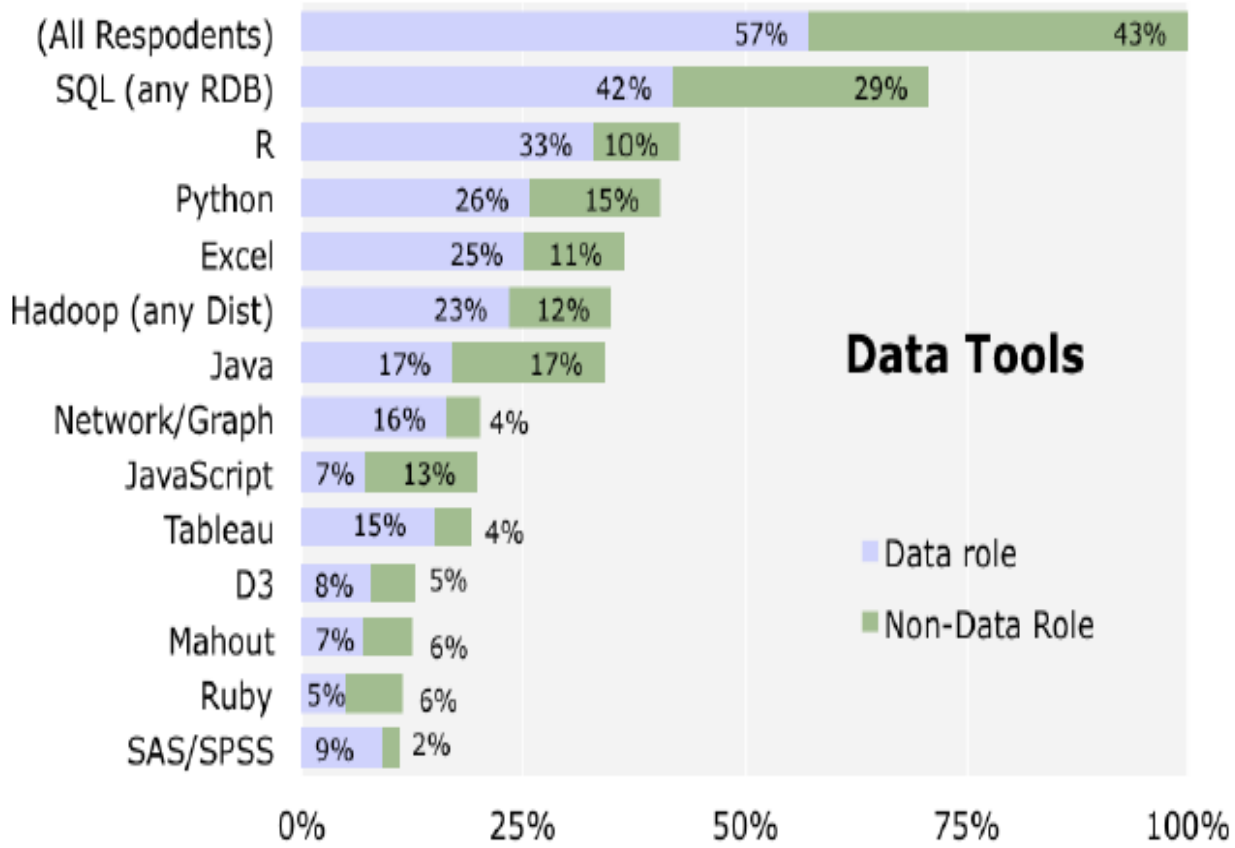
Nothing is lost or hidden

- 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

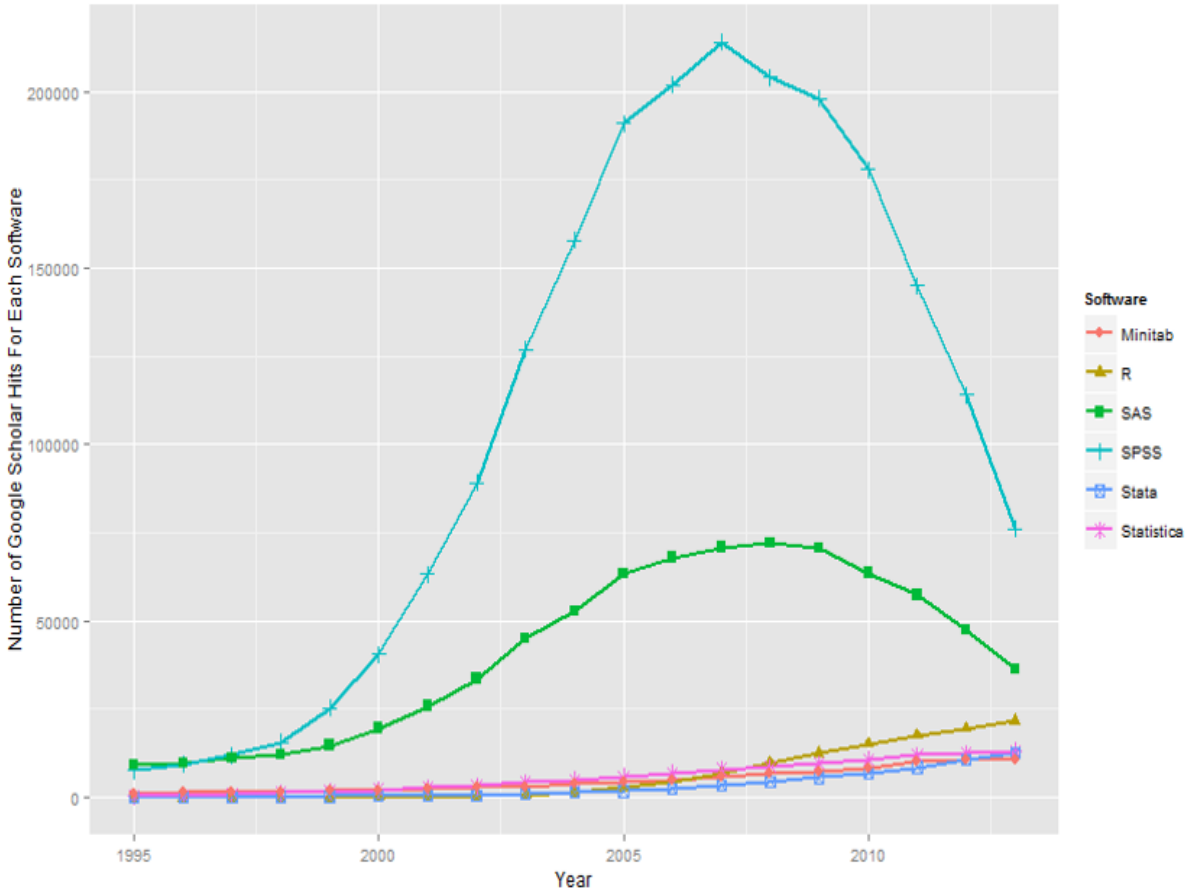
The number of analytics jobs for the more popular software (250 jobs or more, 2/2014).



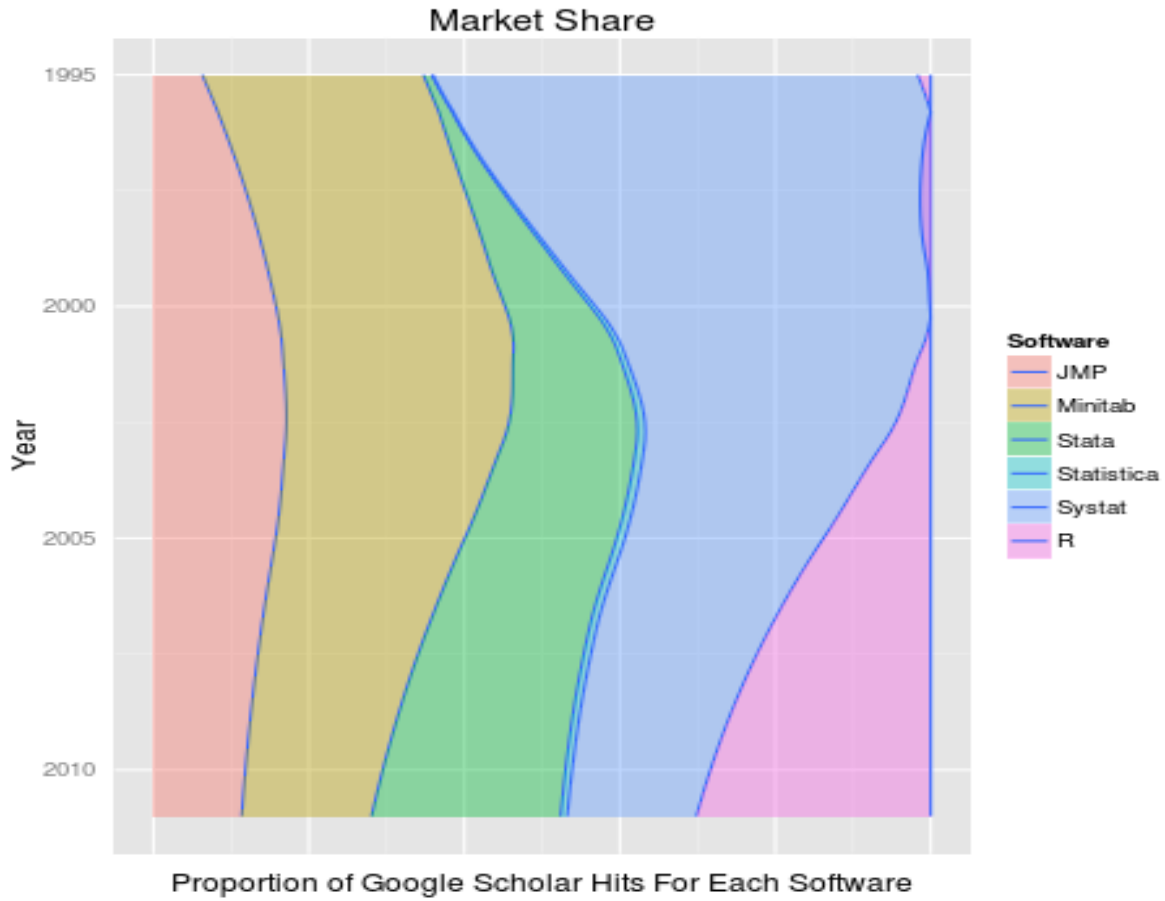
O'Reilly Data Science Survey results for 2012 and 2013 combined.



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



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 ;)

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

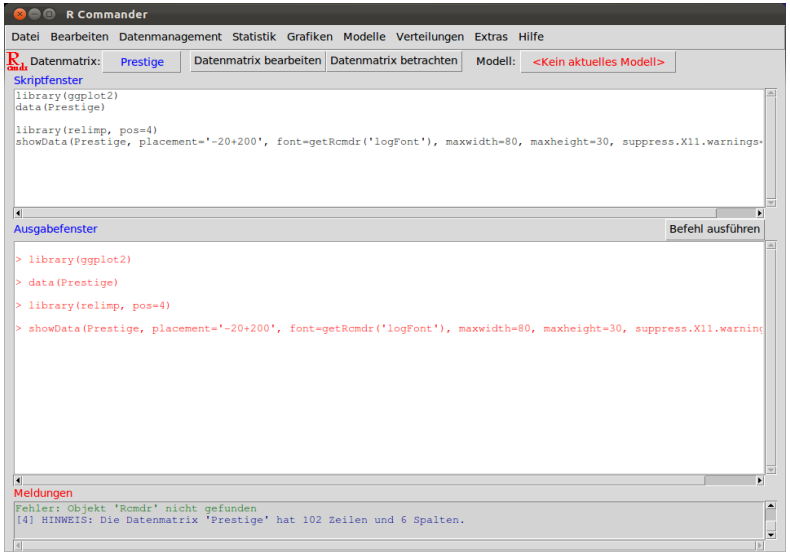
- CRAN stands for Comprehensive R Archive Network
- `http://cran.r-project.org`
  - Microsoft Windows: :: http://cran.r-project.org/bin/windows/base/
  - MacOS: :: http://cran.r-project.org/bin/macosx/
  - Linux: :: http://cran.r-project.org/bin/linux/
- for mac and pc users: just download and install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

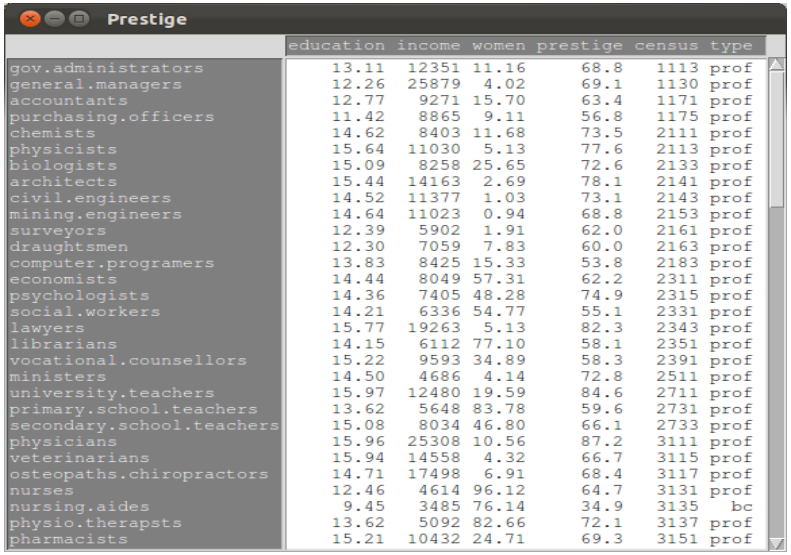
The R-Commander

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

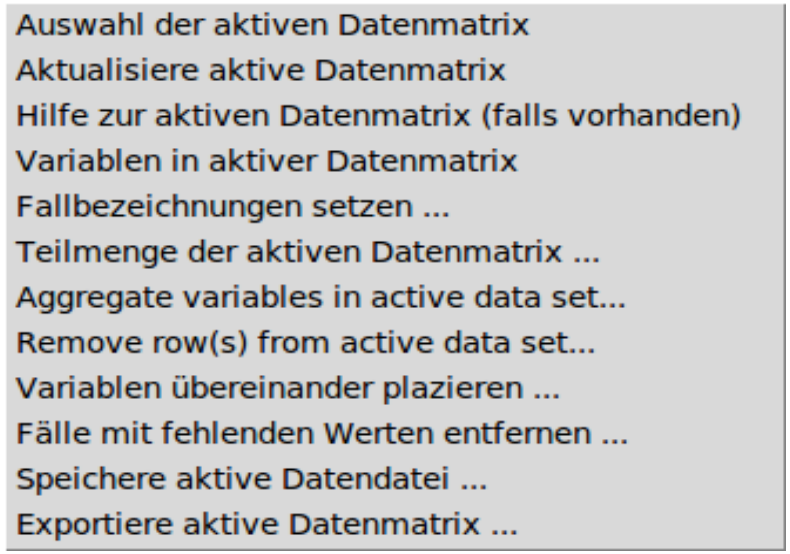
- `Rcmdr` has a comprehensive menu, which includes data reading, summaries, statistical analyses, etc.
- When the menu is activated, the `Rcmdr` will generate an R script. This script can be used as a log for documentation or for self learning.
- It has excellent graphical tools.

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

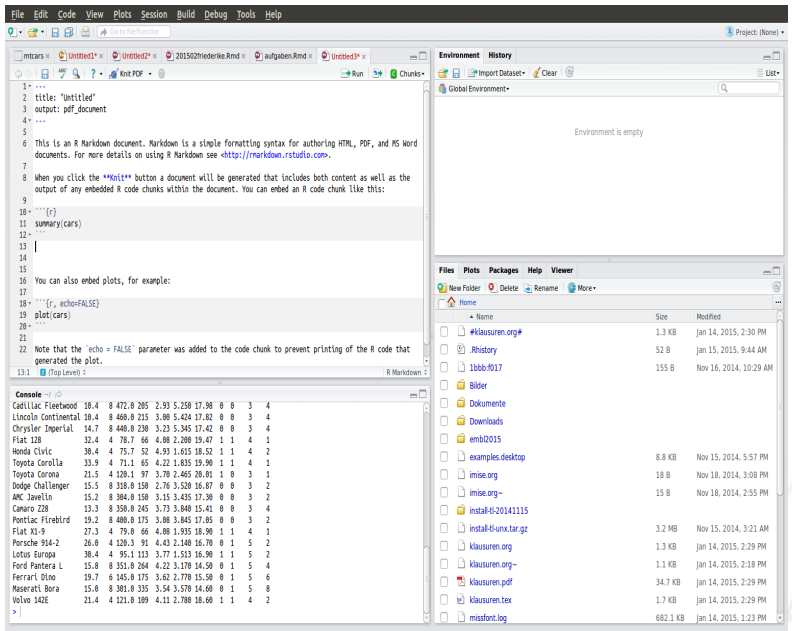
To install `Rcmdr` go to Packages $\rightarrow$ Install package(s) (or simply type `install.packages("Rcmdr")`), then choose a CRAN mirror close to you, than OK. A window with a list of packages will pop-up, on this list choose `Rcmdr` and OK. A bundle of packages will be automatically installed.
To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

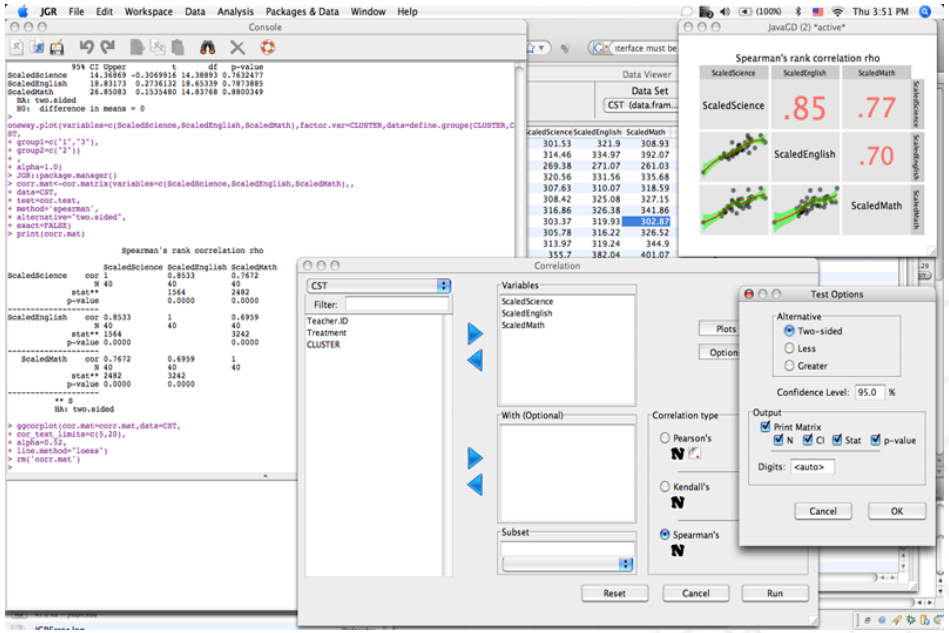
Getting R-Studio

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

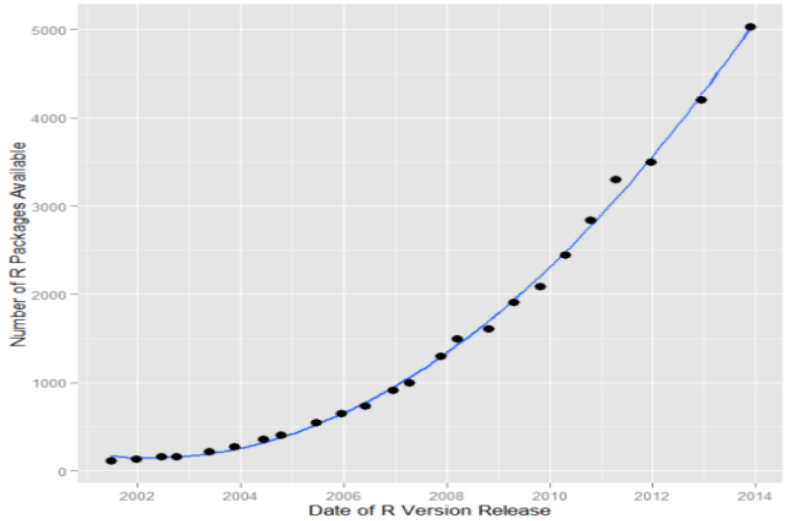
Table of Contents II

Table of Contents III

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 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/`
- if you are into it: there are R and BioConductor Docker container available

Table of Contents I

Table of Contents II

Table of Contents III

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
- It is possible to use R as a text-only application, and also in batch mode (Rscript)

First Session

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

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

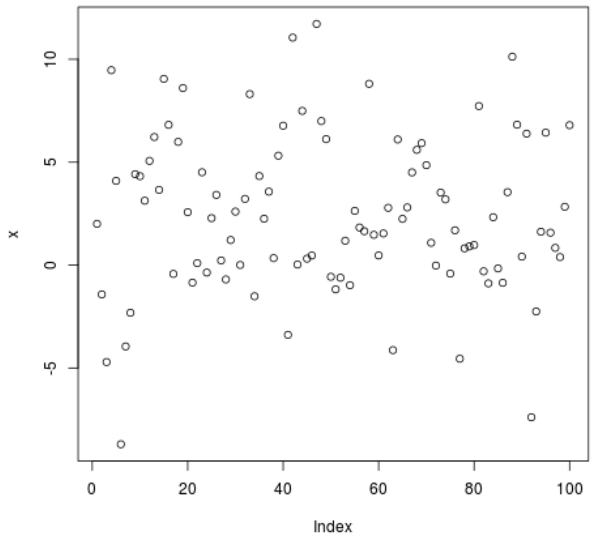
  creates a variable `x` containing a vector with 100 random numbers normal distributed with mean 2 and standard deviation 4
- to see the contents of this variable just type

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

First Session

- To plot these values type

```
> plot(x)
```



First Session

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

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

- 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

First Session

- 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

Table of Contents II

Table of Contents III

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

Table of Contents II

Table of Contents III

R as a calculator

One of the simplest possible tasks in R is to enter an arithmetic expression and receive a result.

```
> 2 + 2
[1] 4
> exp(-2)
[1] 0.1353353
> round(exp(-2),3)
[1] 0.135
>
```

Objects and Functions

R allows you to build powerful procedures from simple building blocks. These building blocks are objects and functions

- all data in R is represented by objects
- you - the user - call functions
- functions act on objects to create new objects

Assignments

So lets create an object. R, like other computer languages, has symbolic variables, that is names that can be used to represent values. To assign the value 2 to the variable `x` and then work with it, you can enter

```
> x <- 2
> x
[1] 2
> x+x
[1] 4
> 2*x+exp(x)
[1] 11.38906
>
```

Assignments

- Names of variables can be chosen quite freely in R. They can be built from letters, digit, and period (dot) symbol. Names that start with a period should be avoided.
- A typical variable name could be `height.y1`, which might be used to describe the height of a child at the age of 1 year.
- Names are case-sensitive: `Pablo` and `pablo` do NOT refer to the same variable.

Assignments names to avoid

- Some names are already used by the system!
- Please avoid the following names:
  - `c`, `q`, `s`, `t`, `C`, `D`, `F`, `I`, `T`
  - `diff`, `df`, `mean`, `pi`, `range`, `rank`, `var`, `sort`.

Vectorial Arithmetic

One strength of R is that it can handle entire data vectors as single objects. A data vector is simply an array of numbers, and a vector can be constructed like this:

```
> weight <- c(60, 72, 57, 90, 95, 72)
> weight
[1] 60 72 57 90 95 72
>
```

The construct `c(...)` is used to define vectors. It is neither the only way to enter data vectors into R, nor is it generally the preferred method, but short vectors are used for many purposes, and the `c(...)` construct is used extensively.

Vectorial Arithmetic

You can do calculations with vectors just like ordinary numbers.

```
> height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
> bmi <- weight/height^2
> bmi
[1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.7363
>
```

Note that the operation is carried out element-wise, that is, the first value of `bmi` is $60/1.75^2$

Some Calculations

This vector calculations make it very ease to specify typical statistical calculations. Consider, the calculation of the mean and the standard deviation of the `weight` variable. First, calculate $\overline{x} = \sum x_i/n$:

```
> xbar <- sum(weight)/length(weight)
> xbar
[1] 74.33333
>
```

Some Calculations

Now, we calculate $sd = \sqrt{(\sum(x_i - \overline{x})^2)/(n-1)}$:

```
>
> sum((weight - xbar)^2)
[1] 1189.333
> sqrt(sum((weight - xbar)^2)/(length(weight) - 1))
[1] 15.42293
>
```

Of course, R has all of these calculations implemented!!

```
> mean(weight)
[1] 74.33333
> sd(weight)
[1] 15.42293
```

Standard Procedures

The rule of thumb is that the BMI for a normal-wight individual should be between 20 to 25. We want to know if our data deviate systematically from that. You might use a one-sample t test to assess whether the 6 persons BMI can be assumed to have mean 22.5 given that they come from a normal distribution.
To this end we can use the function `t.test`.
This function has many arguments, we are going to specify two: the data name and the hypothetical value of μ, the theoretical mean.

Standard Procedures

```
> t.test(bmi, mu=22.5)

    One Sample t-test

data:  bmi
t = 0.3449, df = 5, p-value = 0.7442
alternative hypothesis: true mean is not equal to 22.5
95 percent confidence interval:
 18.41734 27.84791
sample estimates:
mean of x
 23.13262
```

Vectors

We have already seen numerical vectors. There are two further types: character vectors and logical vectors.
A character vector is a vector of text strings, whose elements are specified and printed in quotes:

```
> c("Pablo", "Ana", "Carlos", "Bill")
[1] "Pablo"  "Ana"    "Carlos" "Bill"
> colours<-c("red", "blue", "green", "white", "black")
> colours
[1] "red"   "blue"  "green" "white" "black"
> letters[1:5]
[1] "a" "b" "c" "d" "e"
```

Vectors

Logical vectors can take the value TRUE or FALSE (or NA). We may use a convenient abbreviation T and F.

```
> c(T,T,F,F,T)
[1]  TRUE  TRUE FALSE FALSE  TRUE
> bmi
[1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.7363
> bmi>25
[1] FALSE FALSE FALSE FALSE  TRUE FALSE
>
```

Functions to Create Vectors

- c is a short for concatenate, that is, joining items end to end:

```
> c(bmi[1:2],bmi[5:6])
[1] 19.59184 22.22222 31.37799 19.73630
```

- seq is a short for sequence, it is used for equidistant series of numbers.

```
> seq(11,100,by=10)
[1] 11 21 31 41 51 61 71 81 91
> seq(11,100,length=10)
 [1]  11.00000  20.88889  30.77778  40.66667  50.5555
 [8]  80.22222  90.11111 100.00000
> seq(1,10) ## the same like 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
```

Functions to Create Vectors

- `rep` is a short for "replicate". It is used to generate repeated values. It is used in two variants depending on whether the second argument is a vector or a single number:

```
> tmp <-1:5
> tmp
[1] 1 2 3 4 5
> rep(tmp,2)
 [1] 1 2 3 4 5 1 2 3 4 5
> rep(tmp,tmp)
 [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
>
```

Matrices and Arrays

In R, the matrix notion is extended to elements of any type, so you could have, for instance, a matrix of character strings. Matrices and arrays are represented as vectors with dimensions:

```
> x <- 1:12
> dim(x) <- c(3,4)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

The `dim()` assignment function sets the dimension attribute of `x`. Notice that the storage is column-major, i.e., the elements of the first column are followed by those of the second, etc.

Matrices and Arrays

We can change from two dimension to three dimension array with

```
> dim(x) <- c(3,2,2)
> x
, , 1
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
, , 2
...
```

Matrices and Arrays

A convenient way to create matrices is to use the `matrix` function:

```
 > matrix(letters[1:12], nrow=3, byrow=T)
     [,1] [,2] [,3] [,4]
[1,] "a"  "b"  "c"  "d"
[2,] "e"  "f"  "g"  "h"
[3,] "i"  "j"  "k"  "l"
>
```

The byrow=T causes the matrix to be filled by row.

Matrices and Arrays

Useful functions that operate on matrices are `rownames()`, `colnames()` and the transposition `t()`.

```
> x <- matrix(1:6, nrow=3, byrow=T)
> rownames(x) <-LETTERS[1:3]
> x
  [,1] [,2]
A    1    2
B    3    4
C    5    6
> t(x)
     A B C
[1,] 1 3 5
[2,] 2 4 6
```

Matrices and Arrays

We can "glue" vectors together, columnwise or rowwise, using the `cbind()` and `rbind()` functions.

```
 > cbind(Pablo=1:3,Maria=3:5,Elsa=5:7)
     Pablo Maria Elsa
[1,]     1     3    5
[2,]     2     4    6
[3,]     3     5    7
> rbind(Pablo=1:3,Maria=3:5,Elsa=5:7)
      [,1] [,2] [,3]
Pablo    1    2    3
Maria    3    4    5
Elsa     5    6    7
```

Factors

- Variables indicating some subdivision of data, such as social class, tumor stage, etc. should be specified as factors in R.
- It is essential in statistical analysis to distinguish between categorical codes and variables whose values have a direct numerical meaning.
- R automatically understands this issue when we use factors.

Factors

The main feature of factors is that they have levels. For example a four levels factor is represented by:

(a) a vector with values from 1 to 4 and

(b) a character vector of length 4 containing strings describing what the four levels are

Let's look at an example:

Factors

```
> pain <- c(0,0,1,1,2)
> fpain <- factor(pain, levels=0:2,
+                       labels=c("none","medium","severe"))
> fpain
[1] none   none   medium medium severe
Levels: none medium severe
> fpain <- factor(pain, levels=0:2,
+                       labels=c("none","medium","severe"),
+                       ordered=T)
> fpain
[1] none   none   medium medium severe
Levels: none < medium < severe
```

Factors

ordered factors

It is good to keep in mind whether your factors are ordered or not. Ordered and unordered factors are processed different in modelling (by using different kind of contrasts).

A Functions to Create Factors

This is the function `gl`:

```
> treatment <- gl(2, 4, label = c("Control", "Treat"))
> treatment
[1] Control Control Control Control Treat   Treat
[7] Treat   Treat
Levels: Control Treat
>
```

the first argument gives the number of levels and the second the number of replications.

Lists

A list is used in R to collect together items of different types. For example:

```
> Emp <- list(emp="Anna", spouse="Fred",
+             children=3, ages =c(4,7,9))
>
```

This example shows, the elements of a list do not have to be the same length. The list `Emp` is of length 4. To extract elements of a list:

```
> Emp$emp
[1] "Anna"
> Emp$ages[1:2]
[1] 4 7
>
```

Data Frames

A data frame in R corresponds to what other statistical packages call a "data matrix", i.e., a matrix with variables of the same length, but possibly of different type (numeric, character, factor or logical).
As an example, consider a data set concerning the maximum oxygen intake (VO_2max) for 8 healthy women. The data was obtained by the CNARD project in Argentina (thanks to Dr. Lentini).

```
> vo2A <- c(3.57, 3.333, 2.499, 2.846, 3.342, 2.646)
> vo2R <- c(54.1, 56.5, 50.7, 51.7, 49.9, 49.5)
```

Data Frames

To combine them as a data frame:

```
>
> dat <- data.frame(vo2A, vo2R)
>
> rm(vo2A, vo2R)
> dat
   vo2A vo2R
1 3.570 54.1
2 3.333 56.5
3 2.499 50.7
4 2.846 51.7
5 3.342 49.9
6 2.646 49.5
```

str()

We can access to the structure of any R object with the function `str()`. For example:

```
> str(dat)
'data.frame':   6 obs. of  2 variables:
 $ vo2A: num  3.57 3.33 2.5 2.85 3.34 ...
 $ vo2R: num  54.1 56.5 50.7 51.7 49.9 49.5
```

To access elements we use the $ symbol like:

```
> dat$vo2A[1:4]
[1] 3.570 3.333 2.499 2.846
> dat$vo2R
[1] 54.1 56.5 50.7 51.7 49.9 49.5
```

Missing Values

R allows vectors to contain a special NA value. This value is carried through in computations so that operations on NA yield as the result NA. For example:

```
> x<-c(rnorm(5,mean=5,sd=2),NA)
> x
[1] 4.873961 3.090246 8.109162 2.050998 5.740921 NA
> mean(x)
[1] NA
> is.na(mean(x))
[1] TRUE
> mean(x, na.rm = T)
[1] 4.773057
>
```

Infinity and Things that Are Not a Number (NaN)

Calculations can lead to answers that are infinity, represented in R by `Inf`, or minus infinity, which is represented as `-Inf`. Or Calculations involving infinity can be evaluated:

```
> 3/0
[1] Inf
> -12/0
[1] -Inf
>
```

```
> exp(-Inf)
[1] 0
> 0/Inf
[1] 0
> (0:3)^Inf
[1] 0 1 Inf Inf
>
```

Infinity and Things that Are Not a Number (NaN)

Other calculations, however, lead to quantities that are not numbers. These are represented in R by NaN ('not a number'). Here are some of the classic cases:

```
> 0/0
[1] NaN
> Inf-Inf
[1] NaN
> Inf/Inf
[1] NaN
```

Infinity and Things that Are Not a Number (NaN)

You need to understand clearly the distinction between NaN and NA. The function is.nan() is provided to check specifically for NaN, and is.na() also returns TRUE for NaN. Cœrcing NaN to logical or integer type gives an NA of the appropriate type. There are built-in tests to check whether a number is finite or infinite:

```
> is.finite(10)
[1] TRUE
> is.infinite(10)
[1] FALSE
> is.infinite(Inf)
[1] TRUE
>
```

Sorting Vectors

Sorting vectors is trivially done with the function `sort()`:

```
> sort(c(4,3,1,9,4))
[1] 1 3 4 4 9
>
> sort(c("abc", "ab", "aba", "ab.", ".ab", "-a"))
[1] "-a"  ".ab" "ab"  "ab." "aba" "abc"
```

`order()` returns a permutation which rearranges its first argument into ascending or descending order

```
>  order(c(4,3,1,9,4))
[1] 3 2 1 5 4
> order(c("abc", "ab", "aba", "ab.", ".ab", "-a"))
[1] 6 2 4 5 3 1
```

Sorting Data Frames

For sorting data frames by a column, the function `order()` is applied to the this column

```
dd <- data.frame(
+      b = factor(c("Hi", "Med", "Hi", "Low"),
+         levels = c("Low", "Med", "Hi"), ordered = TRUE)
+      x = c("A", "D", "A", "C"),
+      y = c(8, 3, 9, 9),
+      z = c(1, 1, 1, 2))
> dd
     b x y z
1   Hi A 8 1
2  Med D 3 1
3   Hi A 9 1
4  Low C 9 2
```

Sorting Data Frames - Old Style

```
> dd[order(dd$b), ]     # Here we order all by variable '
    b x y z
4 Low C 9 2
2 Med D 3 1
1  Hi A 8 1
3  Hi A 9 1
```

Sorting Data Frames - New Style

- nowadays there are convenience packages out there; one of them is the `dplyr` package (Hadley Wickham)
- sorting can now be done in the following way:

```
> arrange(dd,b,desc(x))
    b x y z
1 Low C 9 2
2 Med D 3 1
3  Hi A 8 1
4  Hi A 9 1
```

Table of Contents I

Table of Contents II

Table of Contents III

Tests

- Parametric classical tests

Tests

- Parametric classical tests

- Distribution-free tests

Tests

- Parametric classical tests
- Distribution-free tests
- Sequential tests

Tests

- Parametric classical tests
  - for central tendency
- Distribution-free tests
  - for central tendency
- Sequential tests
  - for central tendency

Tests

- Parametric classical tests
  - for central tendency
  - for proportion
- Distribution-free tests
  - for central tendency
- Sequential tests
  - for central tendency
  - for proportion

Tests

- Parametric classical tests
  - for central tendency
  - for proportion
  - for variability
- Distribution-free tests
  - for central tendency
  - for variability
- Sequential tests
  - for central tendency
  - for proportion
  - for variability

Tests

- Parametric classical tests
  - for central tendency
  - for proportion
  - for variability
  - for distribution functions
- Distribution-free tests
  - for central tendency
  - for variability
  - for distribution functions
- Sequential tests
  - for central tendency
  - for proportion
  - for variability

Tests

- Parametric classical tests
  - for central tendency
  - for proportion
  - for variability
  - for distribution functions
  - for association
- Distribution-free tests
  - for central tendency
  - for variability
  - for distribution functions
  - for association
- Sequential tests
  - for central tendency
  - for proportion
  - for variability

Tests

- Parametric classical tests
  - for central tendency
  - for proportion
  - for variability
  - for distribution functions
  - for association
- Distribution-free tests
  - for central tendency
  - for variability
  - for distribution functions
  - for association
- Sequential tests
  - for central tendency
  - for proportion
  - for variability

For the first two:

1 sample

2 samples

k samples

four possible situations

		Situation	
		H_0 is true	H_0 is false
Conclusion	H_0 is not rejected	Correct decision	Type II error
	H_0 is rejected	Type I error	Correct decision

Common symbols

symbol	meaning

Common symbols

`symbol`	meaning
n	number of observations (sample size)
K	number of samples (each having n elements)

Common symbols

`symbol`	meaning
n	number of observations (sample size)
K	number of samples (each having n elements)
α	level of significance
ν	degrees of freedom
σ	standard deviation (population)
s	standard deviation (sample)

Common symbols

`symbol`	meaning
n	number of observations (sample size)
K	number of samples (each having n elements)
α	level of significance
ν	degrees of freedom
σ	standard deviation (population)
s	standard deviation (sample)
μ	population mean
$\bar{x}$	sample mean
ρ	population correlation cœfficient
r	sample correlation cœfficient

Common symbols

`symbol`	meaning
n	number of observations (sample size)
K	number of samples (each having n elements)
α	level of significance
ν	degrees of freedom
σ	standard deviation (population)
s	standard deviation (sample)
μ	population mean
$\bar{x}$	sample mean
ρ	population correlation cœfficient
r	sample correlation cœfficient
Z	standard normal deviate

Numeric summaries

To describe data we need a proper way to summarize them for easier understanding. Therefore we focus on three main areas:

- parameters of location (today)
- spread (today) and
- shape (maybe later)

Exercises

1. load the data ZA5240_v2-0-0.sav from the data directory using the `spss.get()` function from the Hmisc package (data source: gesis.org, General Social Survey 2014), assign the data set to a variable with a appropriate name.
2. the file variablenliste.txt contains a list with the variable description (only available in German)
3. how many rows? (`nrow()`)
4. how many columns? (`ncol()`)

Location parameters

- a location parameter is a central or typical value for a distribution
- typical location parameters are:
  - mean (`mean()`)
  - trimmed means (`mean()`)
  - median (`median()`)
  - mode

Mean and median I

How to interpret the mean?

- graphically, it is the visual balance point of the given values (physics formula for the center of mass)
- this demonstrates a weakness of the mean when used to represent center
- the trimmed mean tries to make the mean more stable by trimming from both sides a certain percentage of the most extreme values
- if mean and the trimmed mean are substantially different the data are very likely to be skewed
- the trimmed mean pushed to its limits (by trimming 50% of the data from each end) leaves us with basically a single value: the median
- so the median is the value which divides the values into the 50% lowest and the 50% highest values

Exercises

1. the column V417 contains the net income; calculate the mean using the `mean()` function! What is the problem?
2. again calculate the mean but now the trimmed version (using the `trim` argument). What is the conclusion?

Other measures of position

- the concept of the median can be generalized: as the median splits the data in half (with half the data smaller and the other half larger), the pth quantile is basically the value in the data set for which $100 \cdot p$ is less than the value and $100 \cdot (1 - p)$ is more (so the median is the 0.5 quantile); special cases of quantiles are percentiles, quartiles and quintiles (`quantiles()`)
- hinges (not often mentioned but used in boxplots; `fivenum()`)
- and of course min and max (`min()`, `max()`)

Exercises

1. summarize the net income using `summary()`, `quantile()` and `fivenum`!
2. make a boxplot by using the following syntax! (we also use column V81 - gender and V86 - graduation)

```
require(ggplot2)
ggplot(x, aes(x=V86, y=V417)) +
    geom_boxplot()
```

3. add gender as coloring

```
ggplot(x, aes(x=V86, y=V417, fill=V81)) +
    geom_boxplot()
```

So what now?

- so we se - there is a difference in income between male and female people
- the next step would be to test if this difference is statistically significant, but therefore we need also the measure of spread (even if you do not use it explicitly in testing) so there are

Spread I

- these are parameters which measure the variability in the data
- there is e.g. the range (`range()`)
- the sample variance (`var()`) and
- the sample standard deviation (`sd()`) which is simply the square root of the variance
- the cœfficient of variation which is the standard deviation normalized by the mean
- the IQR (interquartile range; `IQR` - there are nine ways to calculate it - so different statistics software can have different IQRs - depending on the method)
- the mad (median absolute deviation)

Table of Contents I

Table of Contents II

Table of Contents III

Hypothesis Testing

- R. A. Fischer
- Karl Popper

The statistical null hypothesis testing can be regarded as a mathematical response to the philosophical need for falsification in hypothesis testing.

Hypothesis Testing

- the most common use of statistics by biologists is for null hypothesis tests
- biologists generally use null hypothesis tests based on parametric assumptions
- a large number of null hypothesis procedures have been developed to address particular applications and assumptions, but all take the same approach: They address the question:

How probable are the data if the null hypothesis is true?

Hypothesis Testing

A adherence to this framework can be traced to the following causes:

- there is a strong need for hypothesis testing in the biological science
- these methods generally produce plausible results
- this approach is strongly emphasized in biometric and introductory statistics

But: justification is seldom considered.

Hypothesis Testing

- you can never prove something
- we can only reject (modus tollens) or fail to reject

nothing in science is proven until it is disproved

The Null Hypothesis

- the null hypothesis is often a statement of no effect or no difference
- generally constructed to encompass all possible outcomes except an expected effect
- as a result the rejection of the null supports the expected effect (reductio ad absurdum)

The Null Hypothesis

We look at H_0 and not at the research hypothesis directly because

- as noted above we cannot prove that a hypothesis is true
- it is simply easier to consider statistical evidence from the perspective of H_0
  - the research hypothesis is often no more than an inexact supposition
  - the null hypothesis can often be expressed in exact mathematical terms

Z-test for a population mean

The z-test is something like a t-test (it is like you would know almost everything about the perfect conditions (and therefore it has more power). It uses the normal distribution as test statistic and is therefore a good example.

Objective

To investigate the significance of the difference between an assumed population mean μ_0 and a sample mean $\bar{x}$.

Limitations

1. It is necessary that the population variance σ^2 is known.
2. The test is accurate if the population is normally distributed. If the population is not normal, the test will still give an approximate guide.

Z-test for a population mean

Test statistic

$$Z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$$

1. Write a function which takes a vector, the population standard deviation and the population mean as arguments and which returns the Z score.
   - name the function `ztest`
   - set a default value for the population mean

You can always test your function using simulated values:
`rnorm(100,mean=0)` gives you a vector containing 100 normal distributed values with mean 0.

Z-test for a population mean

Write a function which takes a vector, the population standard deviation and the population mean as arguments and which gives the Z score as result.

```
> ztest <- function(x,x.sd,mu=0){
+     sqrt(length(x)) * (mean(x)-mu)/x.sd
+ }
> set.seed(1)
> ztest(rnorm(100),x.sd = 1)
[1] 1.088874
```

Z-test for a population mean

Add a line to your function that allows you to also process numeric vectors containing missing values!

```
> ztest <- function(x,x.sd,mu=0){
+     x <- x[!is.na(x)]
+     if(length(x) < 3) stop("too few values in x")
+     sqrt(length(x)) * (mean(x)-mu)/x.sd
+ }
```

Z-test for a population mean

The function `pnorm(Z)` gives the probability of $x \leq Z$. Change your function so that it has the p-value (for a two sided test) as result.

```
> ztest <- function(x,x.sd,mu=0){
+     x <- x[!is.na(x)]
+     if(length(x) < 3) stop("too few values in x")
+     z <- sqrt(length(x)) * (mean(x)-mu)/x.sd
+     2*pnorm(-abs(z))
+ }
> set.seed(1)
> ztest(rnorm(100),x.sd = 1)
[1] 0.2762096
```

Z-test for a population mean

Now let the result be a named vector containing the estimated difference, Z, p and the n.

```
> ztest <- function(x,x.sd,mu=0){
+     x <- x[!is.na(x)]
+     if(length(x) < 3) stop("too few values in x")
+     est.diff <- mean(x)-mu
+     z <- sqrt(length(x)) * (est.diff)/x.sd
+     round(c(diff=est.diff,Z=z,pval=2*pnorm(-abs(z)),n=length(x)
+ }
> set.seed(1)
> ztest(rnorm(100),x.sd = 1)
    diff         Z      pval         n
  0.1089    1.0889    0.2762  100.0000
```

Z-test for a population mean

Variants

1. Z-test for two population means (variances known and equal)
2. Z-test for two population means (variances known and unequal)

To investigate the statistical significance of the difference between an assumed population mean μ_0 and a sample mean $\bar{x}$. There is a function `z.test()` in the `BSDA` package.

Limitations (again)

1. It is necessary that the population variance σ^2 is known.
2. The test is accurate if the population is normally distributed. If the population is not normal, the test will still give an approximate guide.

Significance Testing

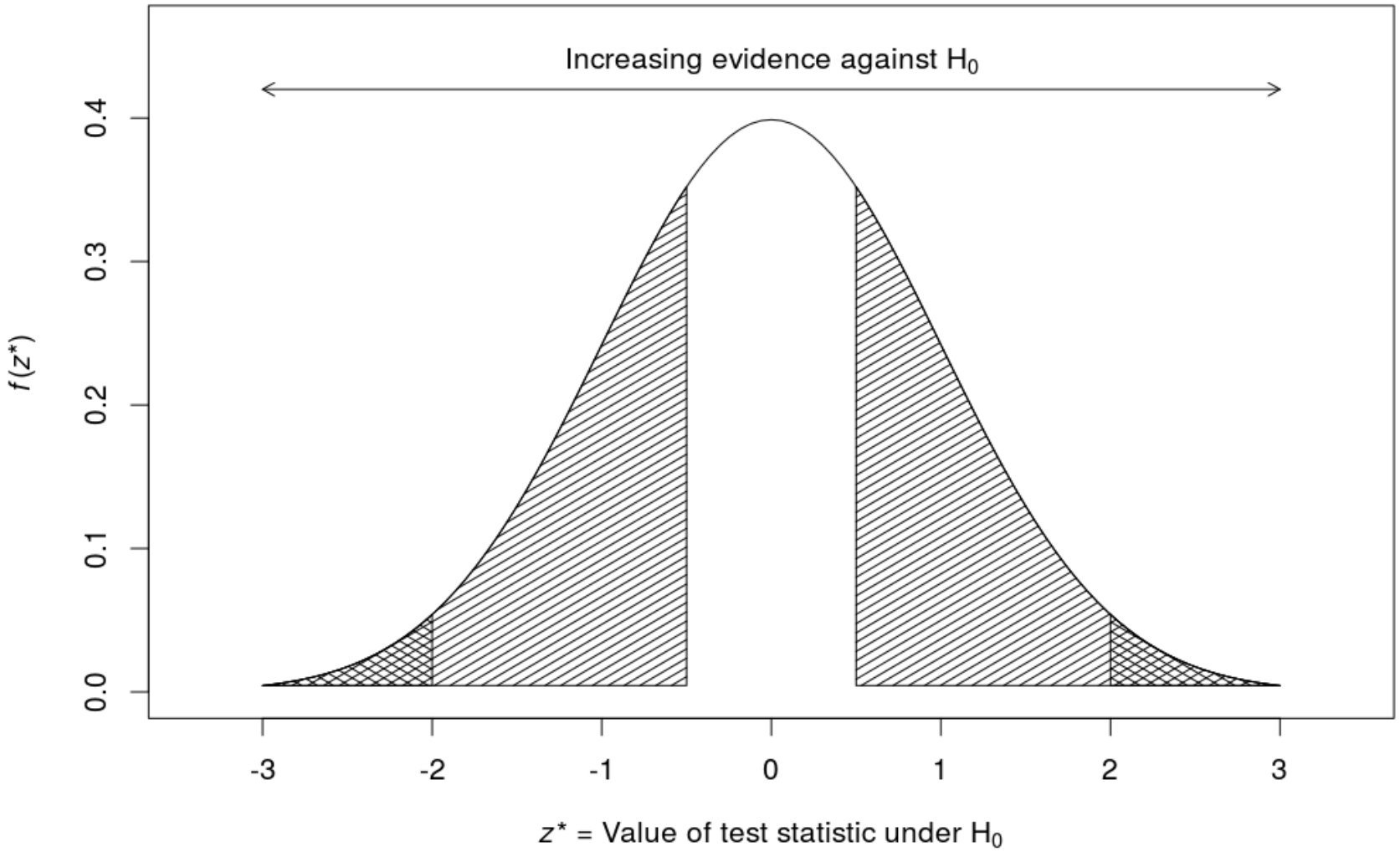
- we are limited to two possible decisions
  1. Reject H_0 or
  2. Fail to reject H_0

Significance Testing – Example

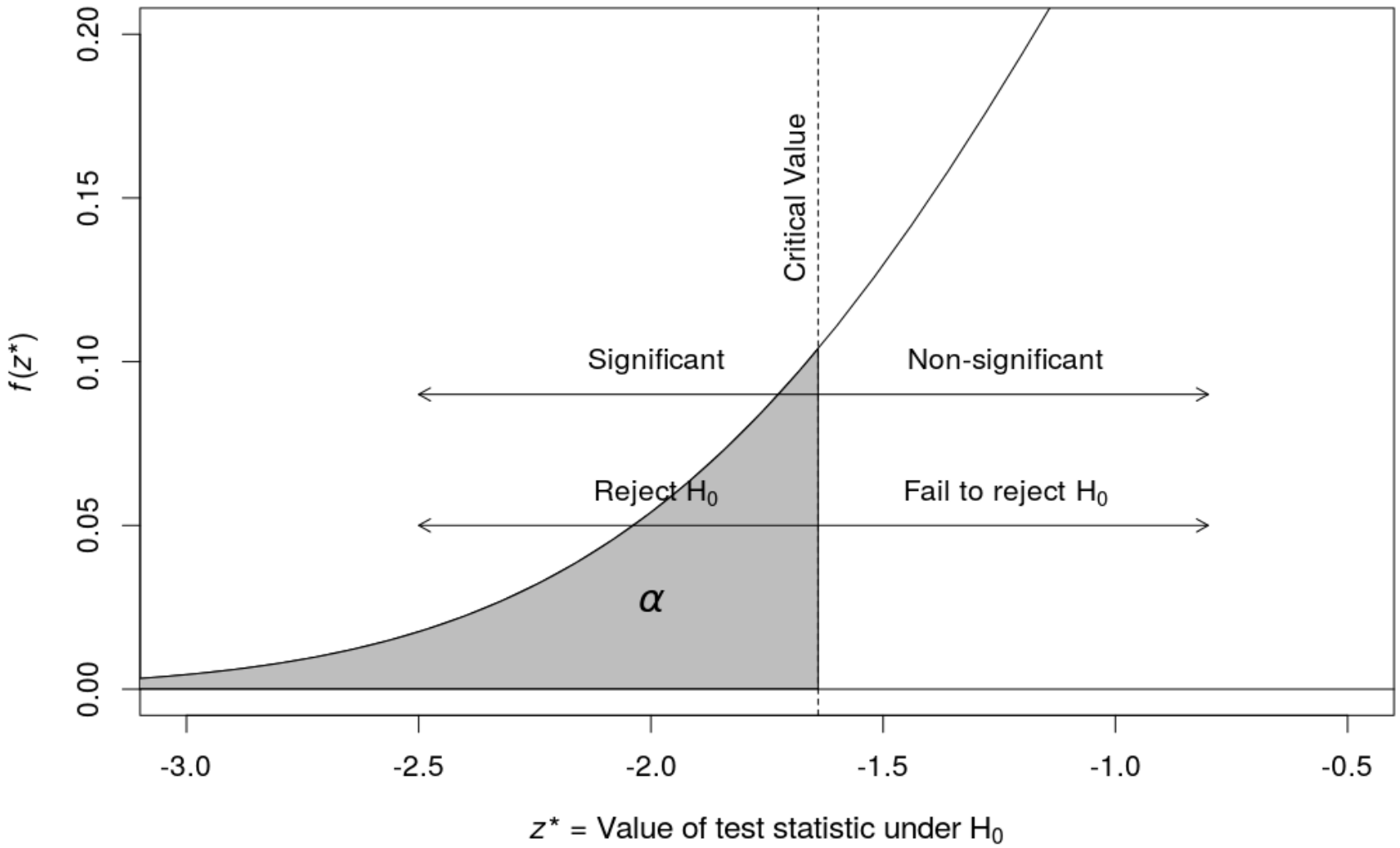
As null hypothesis we might predict that a parameter describing the difference between two populations is zero

- when H_0 is true we would expect an estimate of the parameter to take a value near zero
- an estimator called test statistic is used to quantify the difference between the parameter value quantified in H_0 and the estimated parameter based on the data
- the distribution of the test statistic under H_0 is known
- the p-value is of seeing a test statistic as or more extreme than the test statistic observed if H_0 is true
- smaller p-values designate stronger evidence against H_0

Significance Testing – p-val



Significance Testing – Critical Value



Significance Testing – Question

Question

If we perform a t-test to assess whether there is a difference in means between two groups a test statistic is calculated.
Why the difference not used directly?

Model of Null Hypothesis Testing (Fisher)

- State H_0
- Conduct an investigation that produces data concerning H_0
- Choose an appropriate test and test statistic, and calculate the test statistic
- Determine the p-value
- Reject H_0 if the p-value is small; otherwise, retain H_0

Fisher proposed that a threshold be established describing outcomes that would be extremely improbable if H_0 were true, allowing rejection of H_0. (α - level)

Model of Null Hypothesis Testing (Fisher)

- outcomes meeting this criterion are said to be statistically significant
- Fisher also proposed $\alpha = 0.05$
- later he recommended that fixed significance levels be too restrictive

Model of Null Hypothesis Testing (Neyman & Pearson)

Different in three important aspects:

1. the significance level should be chosen before beginning the data collection
2. incorporate explicitly the alternative hypothesis (Fisher opposed this)
3. introduction of type I and II errors

Non-significant Results

- failure to reject H_0 does not mean that H_0 is true
- different approaches in literature

Significant Results

- having sufficient evidence to reject H_0 does not mean H_0 is untrue
- strict Fisherian view: rejecting the null does not imply anything about H_A

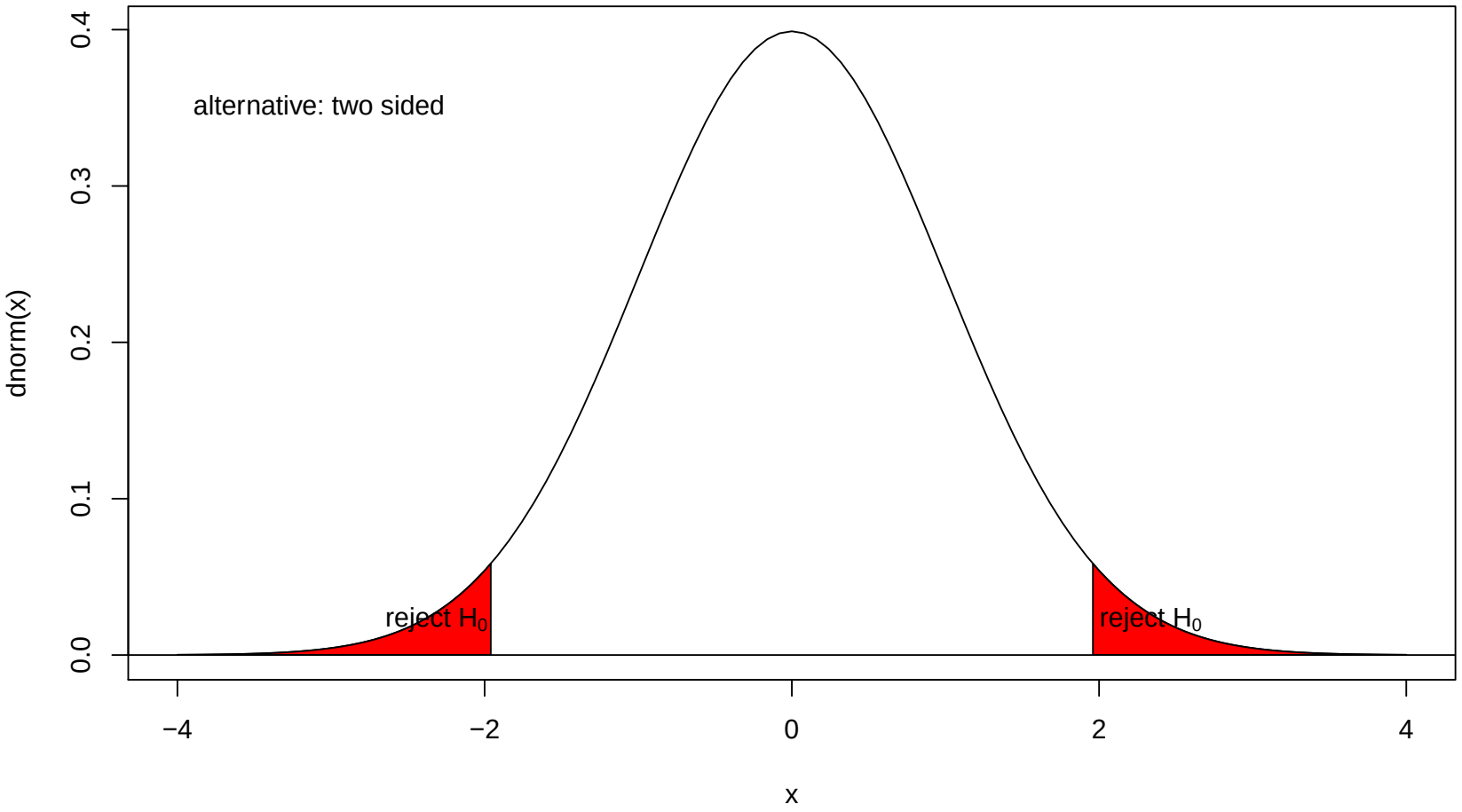
Alternatives

- it may be possible to anticipate directionality in the measured effect
- such scenarios can be accessed with upper- and lower-tailed tests
- H_0 stays $H_0 : X = Y$ or change to $H_0 : X \leq Y$ whereas $H_A : X > Y$

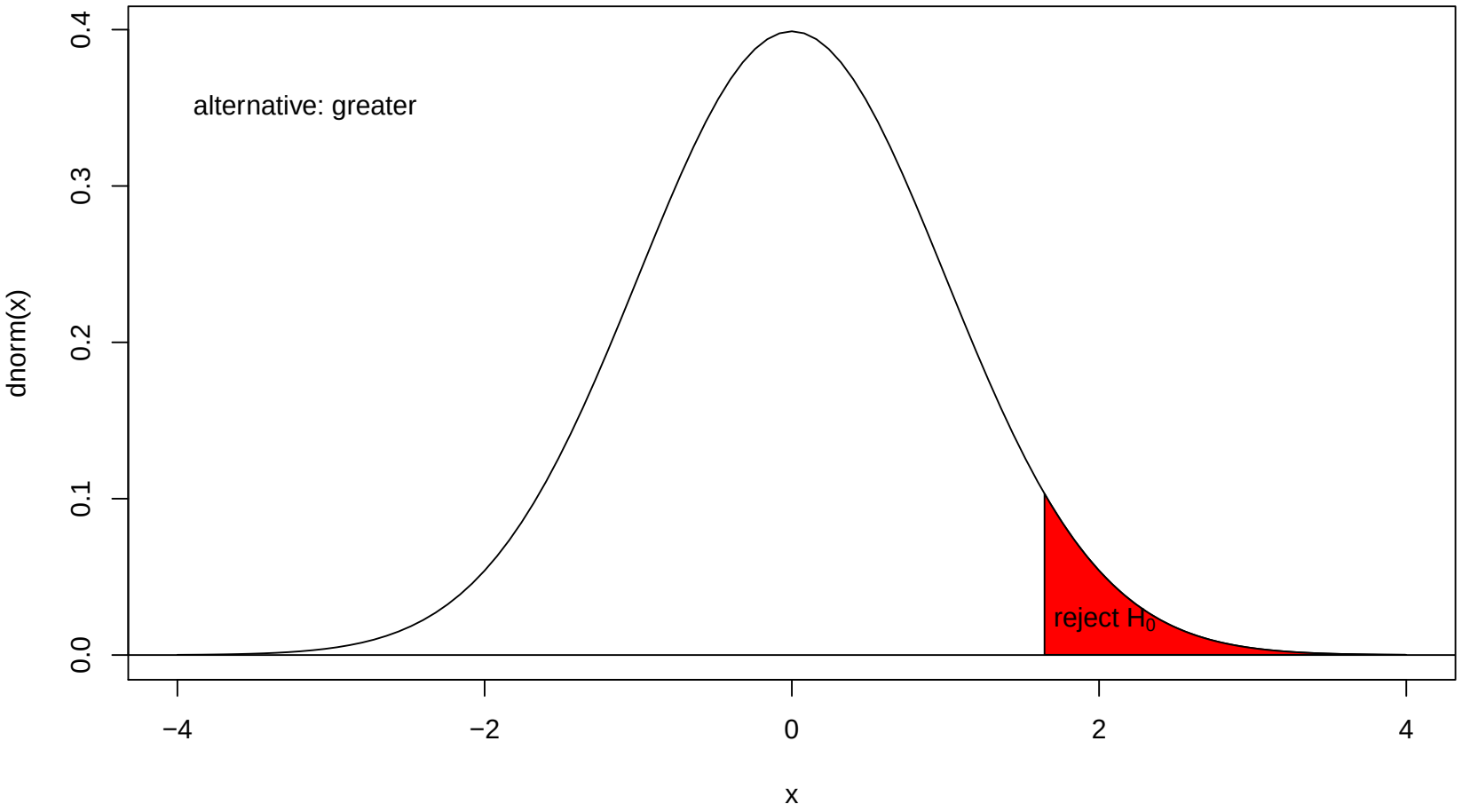
Question

Why – in a strict approach – does H_0 does not change to $H_0 : X \leq Y$. What could be the reason?

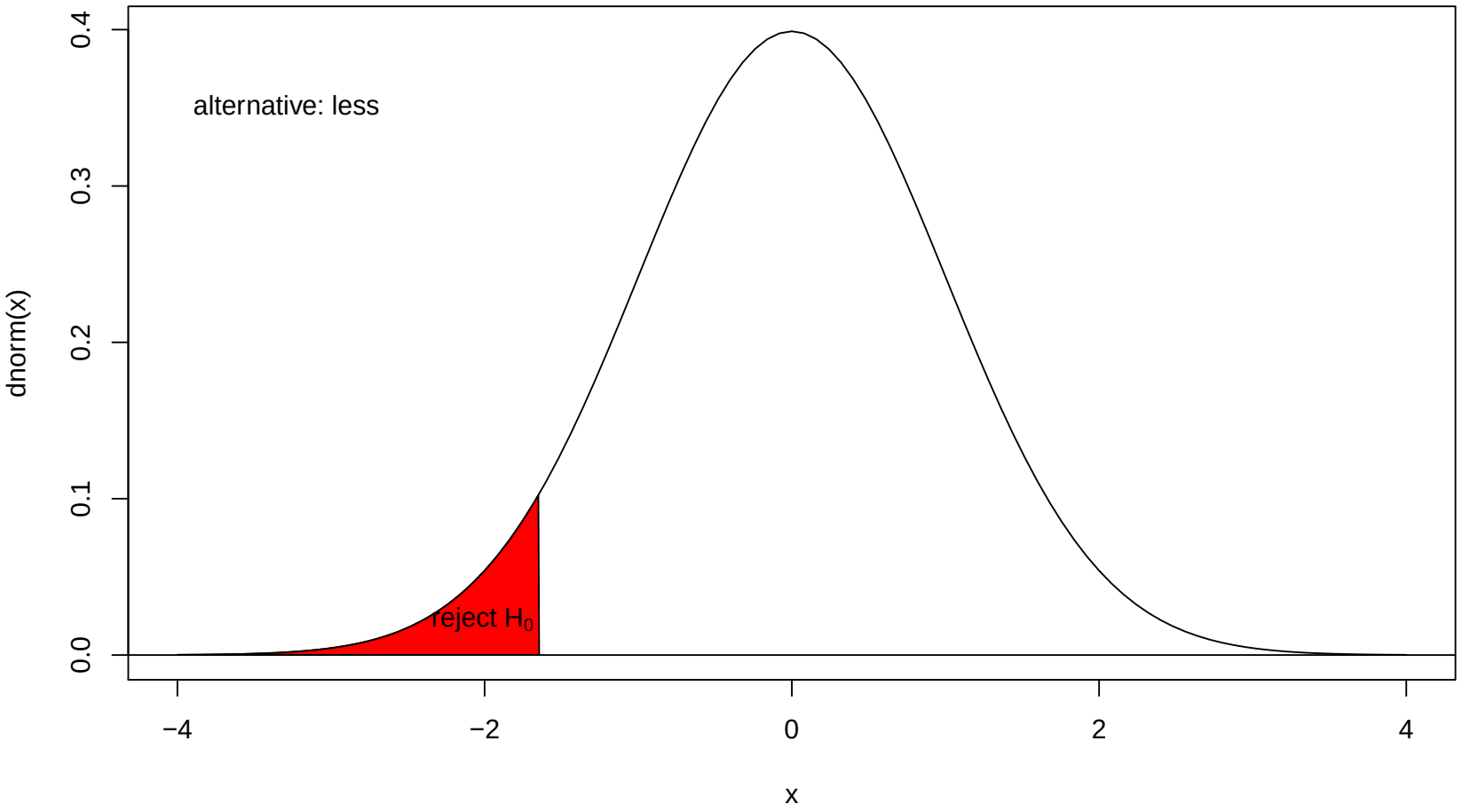
Alternatives



Alternatives



Alternatives



Alternatives

Note:

The p-value is the probability of the sample estimate (of the respective estimator) under the null.

Simulation Exercises

1. Now sample 100 values from a Normal distribution with mean 10 and standard deviation 2 and use a z-test to compare it against the population mean 10. What is the p-value?
2. Now do the sampling and the testing 1000 times, what would be the number of statistically significant results? Use `replicate()` (which is a wrapper of `tapply()`) or a `for()` loop! Record at least the p-values and the estimated differences! Use `table()` to count the p-vals below 0.05. What type of error do you associate with it? What is the smallest absolute difference with a p-value below 0.05?
3. Repeat the simulation above, change the sample size to 1000 in each of the 1000 samples! How many p-values below 0.05? What is now the smallest absolute difference with a p-value below 0.05?

Simulation Exercises – Solutions

- Now sample 100 values from a Normal distribution with mean 10 and standard deviation 2 and use a z-test to compare it against the population mean 10. What is the p-value? What the estimated difference?

```
> ztest(rnorm(100,mean=10,sd=2),x.sd=2,mu=10)["pval"]
  pval
0.0441
> ztest(rnorm(100,mean=10,sd=2),x.sd=2,mu=10)["diff"]
   diff
-0.0655
> ztest(rnorm(100,mean=10,sd=2),x.sd=2,mu=10)[c("pval","
  pval   diff
0.4515 0.1506
```

Simulation Exercises – Solutions

- Now do the sampling and the testing 1000 times, what would be the number of statistically significant results? Use `replicate()` (which is a wrapper of `tapply()`) or a `for()` loop. Record at least the p-values and the estimated differences! Transform the result into a data frame.

using `replicate()`

```
> res <- replicate(1000, ztest(rnorm(100,mean=10,sd=2),x.sd=2,mu=
> res <- as.data.frame(t(res))
> head(res)
     diff       Z   pval   n
1 -0.2834 -1.4170 0.1565 100
2  0.2540  1.2698 0.2042 100
3 -0.1915 -0.9576 0.3383 100
4  0.1462  0.7312 0.4646 100
5  0.1122  0.5612 0.5747 100
6 -0.0141 -0.0706 0.9437 100
```

Simulation Exercises – Solutions

- Now do the sampling and the testing 1000 times, what would be the number of statistically significant results? Use `replicate()` (which is a wrapper of `tapply()`) or a `for()` loop. Record at least the p-values and the estimated differences! Transform the result into a data frame.

using `replicate()` II

```
> res <- replicate(1000, ztest(rnorm(100,mean=10,sd=2),x.sd=2,mu=
+                          simplify = F)
> res <- as.data.frame(Reduce(rbind,res))
> head(res)
        diff       Z   pval   n
init -0.0175 -0.0874 0.9304 100
     -0.0751 -0.3757 0.7072 100
.1    0.0446  0.2232 0.8234 100
.2   -0.3642 -1.8209 0.0686 100
.3   -0.2039 -1.0195 0.3080 100
.4   -0.1872 -0.9359 0.3493 100
```

Simulation Exercises – Solutions

- Now do the sampling and the testing 1000 times, what would be the number of statistically significant results? Use `replicate()` (which is a wrapper of `tapply()`) or a `for()` loop. Record at least the p-values and the estimated differences! Transform the result into a data frame.

using `for()`

```
> res <- matrix(numeric(2000),ncol=2)
> for(i in seq.int(1000)){
+     res[i,] <- ztest(rnorm(100,mean=10,sd=2),x.sd=2,mu=10)[c("pval","diff")] }
> res <- as.data.frame(res)
> names(res) <- c("pval","diff")
> head(res)
    pval    diff
1 0.0591 -0.3775
2 0.2466  0.2317
3 0.6368  0.0944
4 0.5538 -0.1184
5 0.9897 -0.0026
6 0.7748  0.0572
```

Simulation Exercises – Solutions

- Use `table()` to count the p-vals below 0.05. What type of error do you associate with it? What is the smallest absolute difference with a p-value below 0.05?

```
> table(res$pval < 0.05)

FALSE  TRUE
  960    40
> tapply(abs(res$diff),res$pval < 0.05,summary)
$`FALSE`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0002  0.0585  0.1280  0.1411  0.2068  0.3847

$`TRUE`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.3928  0.4247  0.4408  0.4694  0.5102  0.6859

> min(abs(res$diff[res$pval<0.05]))
[1] 0.3928
```

Simulation Exercises – Solutions

- Repeat the simulation above, change the sample size to 1000 in each of the 1000 samples! How many p-values below 0.05? What is now the smallest absolute difference with a p-value below 0.05?

```
> res2 <- replicate(1000, ztest(rnorm(1000,mean=10,sd=2),
+                                x.sd=2,mu=10))
> res2 <- as.data.frame(t(res2))
> head(res2)
     diff       Z   pval    n
1 -0.0731 -1.1559 0.2477 1000
2  0.0018  0.0292 0.9767 1000
3  0.0072  0.1144 0.9089 1000
4 -0.1145 -1.8100 0.0703 1000
5 -0.1719 -2.7183 0.0066 1000
6  0.0880  1.3916 0.1640 1000
```

Simulation Exercises – Solutions

- Repeat the simulation above, change the sample size to 1000 in each of the 1000 samples! How many p-values below 0.05? What is now the smallest absolute difference with a p-value below 0.05?

```
> table(res2$pval < 0.05)

FALSE  TRUE
  946    54
> tapply(abs(res2$diff),res$pval < 0.05,summary)
$`FALSE`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00010 0.02092 0.04285 0.05149 0.07400 0.22610

$`TRUE`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00240 0.02115 0.04535 0.05435 0.08433 0.14760
```

Simulation Exercises Part II

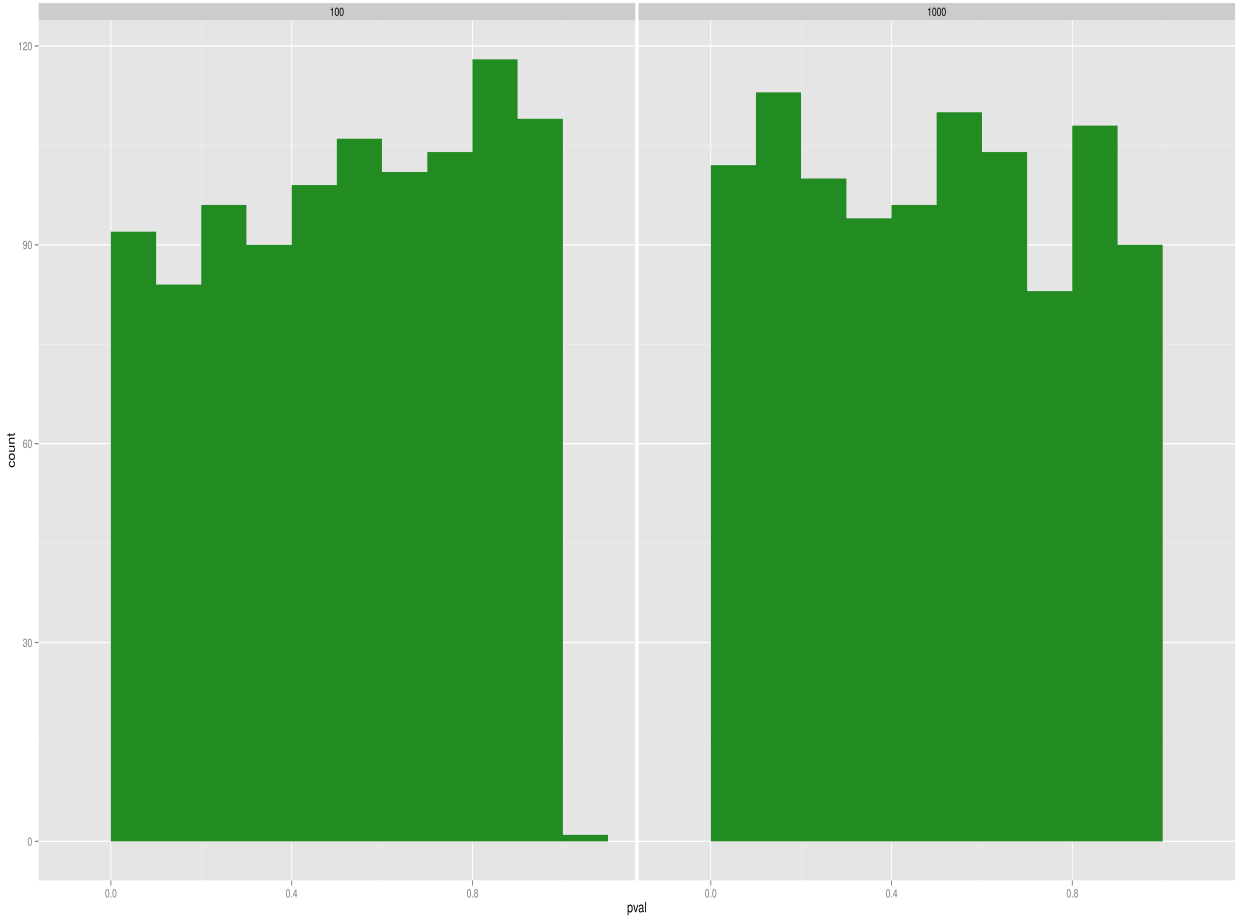
1. Concatenate the both resulting data frames from above using `rbind()`
2. Plot the distributions of the pvals and the difference per sample size. Use `ggplot2` with an appropriate geom (density/histogram)
3. What is the message?

Simulation Exercises – Solutions

- Concatenate the both resulting data frames from above using `rbind()`
- Plot the distributions of the pvals and the difference per sample size. Use `ggplot2` with an appropriate geom (density/histogram)

```
> res <- rbind(res,res2)
> require(ggplot2)
> ggplot(res,aes(x=pval)) +
+     geom_histogram(bin=0.1,fill="forestgreen") +
+     facet_grid(~ n)
> ggsave("hist.png")
```

Simulation Exercises – Solutions

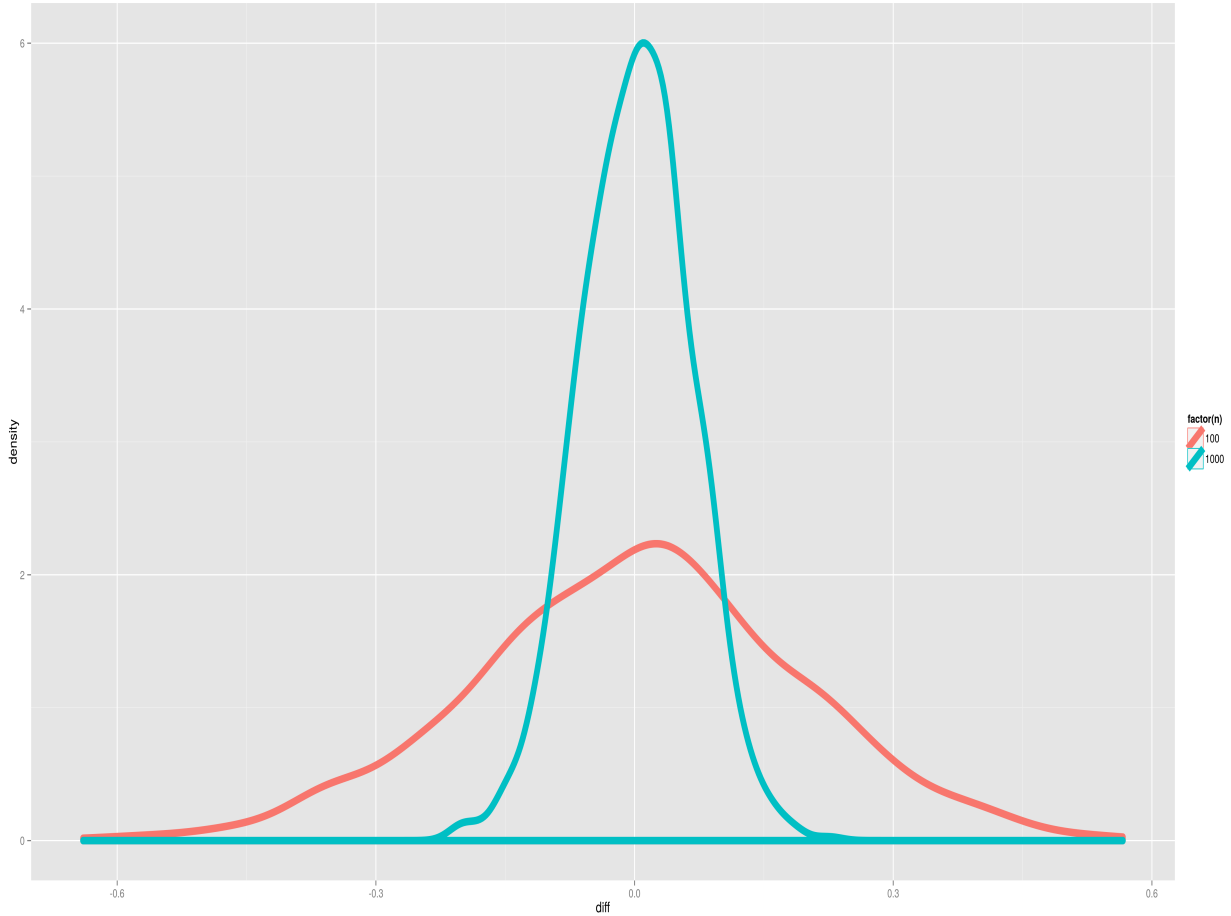


Simulation Exercises – Solutions

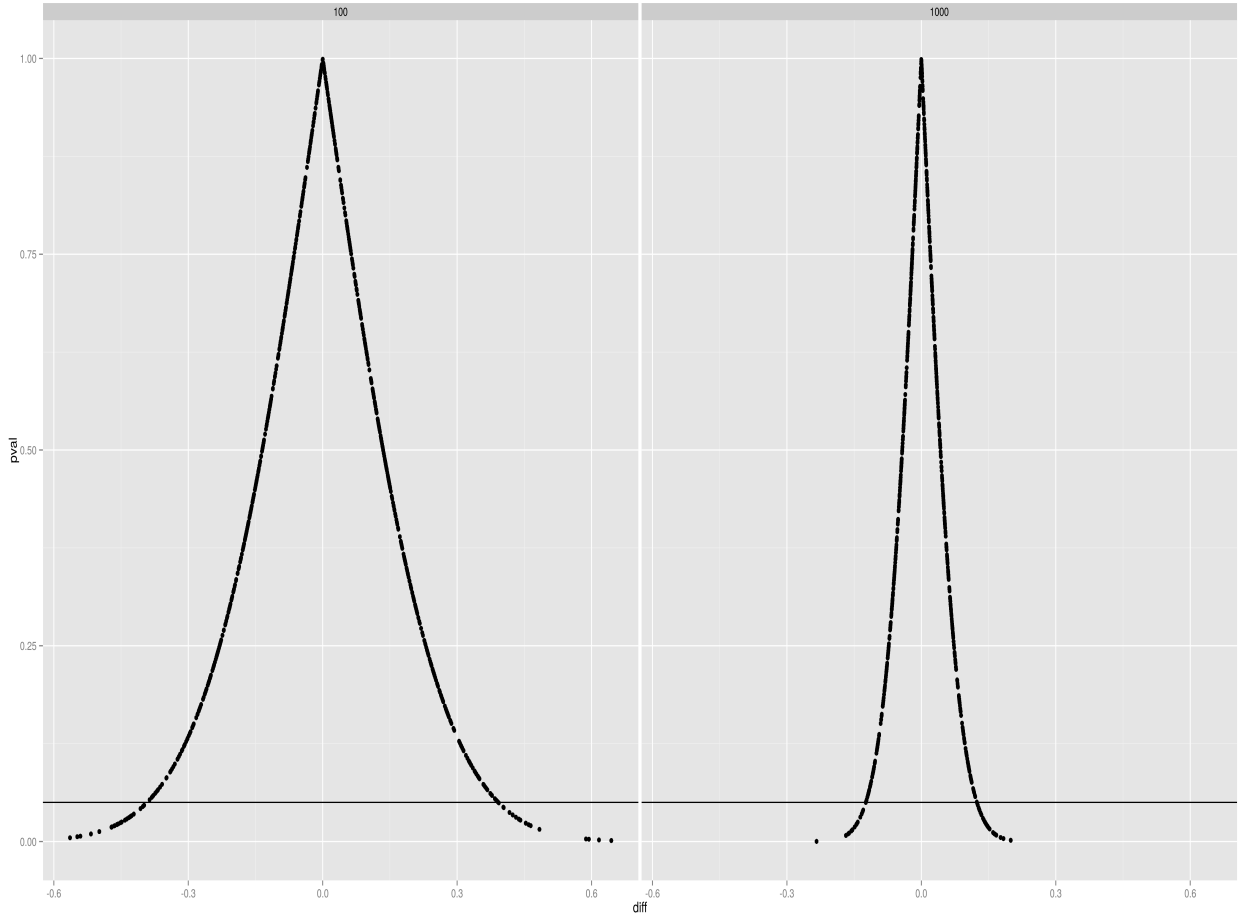
- Plot the distributions of the pvals and the difference per sample size. Use `ggplot2` with an appropriate geom (density/histogram)

```
> ggplot(res,aes(x=diff,colour=factor(n))) +
+     geom_density(size=3)
> ggsave("dens.png")
```

Simulation Exercises – Solutions



Simulation Exercises – Solutions



Simulation Exercises – Solutions

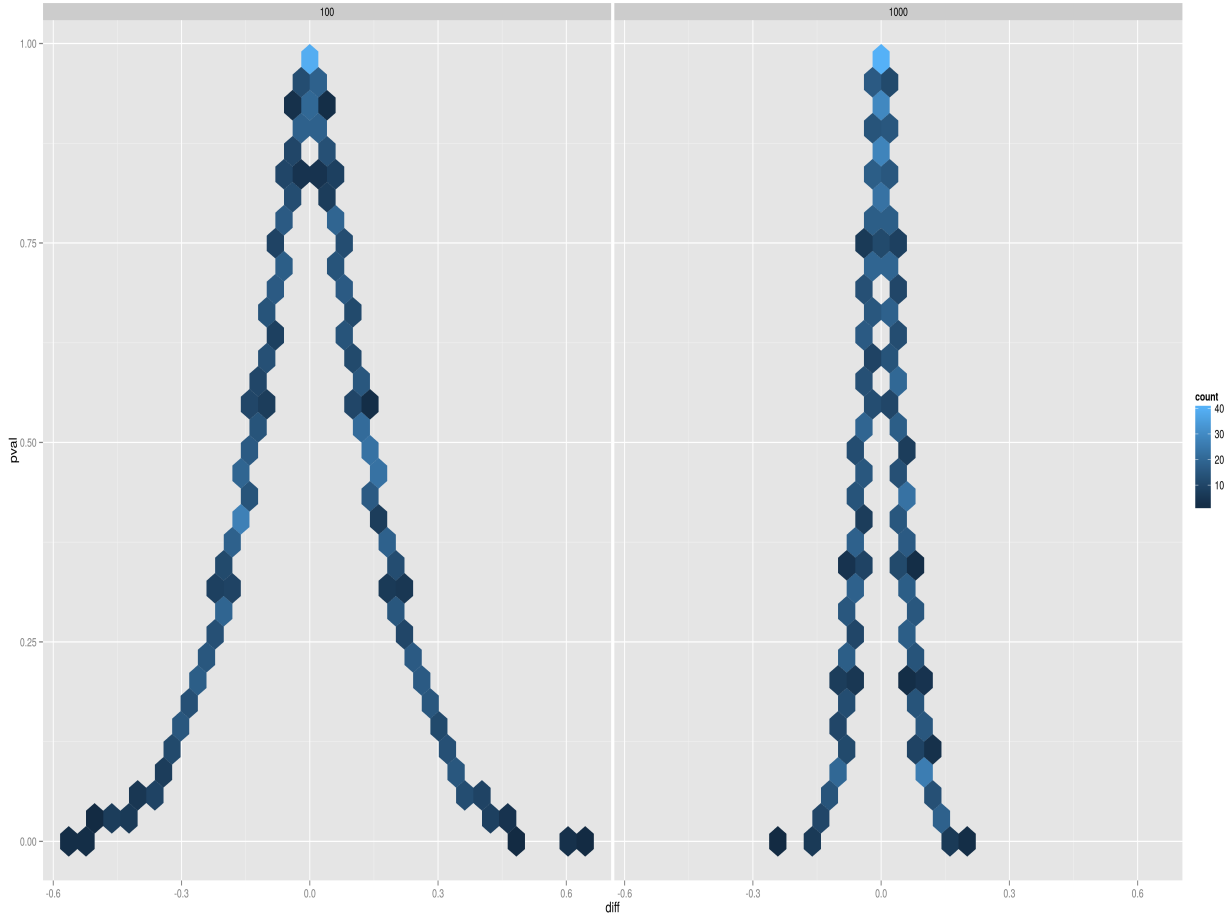


Table of Contents I

Table of Contents II

Table of Contents III

t-tests

A t-test is any statistical hypothesis test in which the test statistic follows a Student's t distribution if the null hypothesis is supported.

- one sample t-test: test a sample mean against a population mean

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

where $\bar{x}$ is the sample mean, s is the sample standard deviation and n is the sample size. The degrees of freedom used in this test is $n - 1$

One Sample t-test

```
> set.seed(1)
> x <- rnorm(12)
> t.test(x,mu=0) ## population mean 0

One Sample t-test

data:  x
t = 1.1478, df = 11, p-value = 0.2754
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.2464740  0.7837494
sample estimates:
mean of x
0.2686377
```

One Sample t-test

```
> t.test(x,mu=1) ## population mean 1

One Sample t-test

data:  x
t = -3.125, df = 11, p-value = 0.009664
alternative hypothesis: true mean is not equal to 1
95 percent confidence interval:
 -0.2464740  0.7837494
sample estimates:
mean of x
0.2686377
```

Two Sample t-tests

There are two ways to perform a two sample t-test in R:

- given two vectors `x` and `y` containing the measurement values from the respective groups `t.test(x,y)`
- given one vector `x` containing all the measurement values and one vector `g` containing the group membership $t.test(x \sim g)$ (read: x dependend on g)

Two Sample t-tests: two vector syntax

```
> set.seed(1)
> x <- rnorm(12)
> y <- rnorm(12)
> g <- sample(c("A","B"),12,replace = T)
> t.test(x,y)

Welch Two Sample t-test

data:  x and y
t = 0.5939, df = 20.012, p-value = 0.5592
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.5966988  1.0717822
sample estimates:
 mean of x  mean of y
0.26863768 0.03109602
```

Two Sample t-tests: formula syntax

```
> t.test(x ~ g)

Welch Two Sample t-test

data:  x by g
t = -0.6644, df = 6.352, p-value = 0.5298
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.6136329  0.9171702
sample estimates:
mean in group A mean in group B
      0.1235413       0.4717726
```

Welch/Satterthwaite vs. Student

- if not stated otherwise `t.test()` will not assume that the variances in the both groups are equal
- if one knows that both populations have the same variance set the `var.equal` argument to TRUE to perform a student's t-test

Student's t-test

```
> t.test(x, y, var.equal = T)

Two Sample t-test

data:  x and y
t = 0.5939, df = 22, p-value = 0.5586
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.5918964  1.0669797
sample estimates:
 mean of x  mean of y
0.26863768 0.03109602
```

t-test

- the t-test, especially the Welch test is appropriate whenever the values are normally distributed
- it is also recommended for group sizes ≥ 30 (robust against deviation from normality)

What we've learned

- if we talk about testing you should know the definition of the following terms, (a) for a one sample (b) for a two sample t-test
  - Null hypothesis
  - Alternative hypothesis
  - Test statistic
  - Significance level
  - Critical value
  - Decision rule
  - Type I error
  - Type II error

Exercises I

1. use a t-test to compare `TTime` according to `Stim.Type`, visualize it. What is the problem?
2. now do the same for Subject 1 on pre and post test (use `filter()` or indexing to get the resp. subsets)
3. use the following code to do the test on every subset `Subject` and `testid`, try to figure what is happening in each step:

```
data.l <- split(data,list(data$Subject,data$testid),drop=T)
tmp.l <- lapply(data.l,function(x) {
    if(min(table(x$Stim.Type)) < 5) return(NULL)
    tob <- t.test(x$TTime ~ x$Stim.Type)
    tmp <- data.frame(
        Subject = unique(x$Subject),
        testid = unique(x$testid),
        mean.group.1 = tob$estimate[1],
        mean.group.2 = tob$estimate[2],
        name.test.stat = tob$statistic,
        conf.lower = tob$conf.int[1],
        conf.upper = tob$conf.int[2],
        pval = tob$p.value,
        alternative = tob$alternative,
        tob$method)})
res <- Reduce(rbind,tmp.l)
```

Exercises II

4. make plots to visualize the results.
5. how many tests have a statistically significant result? How many did you expect? Is there a tendency? What could be the next step?

Exercises - Solutions

- use a t-test to compare `TTime` according to `Stim.Type`, visualize it. What is the problem?

```
> t.test(data$TTime ~ data$Stim.Type)

Welch Two Sample t-test

data:  data$TTime by data$Stim.Type
t = -6.3567, df = 9541.891, p-value = 2.156e-10
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2773.574 -1466.161
sample estimates:
      mean in group hit mean in group incorrect
               17579.77                 19699.64
> ggplot(data,aes(x=Stim.Type,y=TTime)) +
+     geom_boxplot()
```

Exercises - Solutions I

- now do the same for Subject 1 on pre and post test (use `filter()` or indexing to get the resp. subsets)

```
> t.test(data$TTime[data$Subject==1 & data$testid=="test1"] ~
+        data$Stim.Type[data$Subject==1 & data$testid=="test1"])

Welch Two Sample t-test

data:  data$TTime[data$Subject == 1 & data$testid == "test1"] by
t = -0.5846, df = 44.183, p-value = 0.5618
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
 -4930.842  2713.191
sample estimates:
      mean in group hit mean in group incorrect
               8248.175                9357.000

> t.test(data$TTime[data$Subject==1 & data$testid=="test2"] ~
```

Exercises - Solutions II

```
+         data$Stim.Type[data$Subject==1 & data$testid=="test2"])

Welch Two Sample t-test

data:  data$TTime[data$Subject == 1 & data$testid == "test2"] by
t = -1.7694, df = 47.022, p-value = 0.08332
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
 -7004.4904   448.9388
sample estimates:
      mean in group hit mean in group incorrect
               4012.480                 7290.256
```

Exercises - Solutions

- make plots to visualize the results

```
> ggplot(data,aes(x=testid,y=TTime)) +
+     geom_boxplot(aes(fill=Stim.Type)) +
+     facet_wrap(~Subject)
> ggplot(data,aes(x=factor(Subject),y=TTime)) +
+     geom_boxplot(aes(fill=Stim.Type)) +
+     facet_wrap(~testid)
```

Exercises - Solutions

- how many tests have an statistically significant result? How many did you expect?

```
> table(res$pval < 0.05)

FALSE  TRUE
  165    22
> prop.table(table(res$pval < 0.05))

    FALSE      TRUE
0.8823529 0.1176471
>
```

Exercises - Solutions I

- What could be the next step?

```
tmp.l <- lapply(data.l,function(x) {
    if(min(table(x$Stim.Type)) < 5) return(NULL)
    tob <- t.test(x$TTime ~ x$Stim.Type)
    tmp <- data.frame(
        Subject = unique(x$Subject),
        testid = unique(x$testid),
        perc.corr = sum(x$Stim.Type=="hit")/sum(!is.na(x$Stim.Type)),
        mean.group.1 = tob$estimate[1],
        mean.group.2 = tob$estimate[2],
        name.test.stat = tob$statistic,
        conf.lower = tob$conf.int[1],
        conf.upper = tob$conf.int[2],
        pval = tob$p.value,
        alternative = tob$alternative,
        tob$method)})

res <- Reduce(rbind,tmp.l)

ggplot(res,aes(x=perc.corr,y=mean.group.1 - mean.group.2)) +
    geom_point() +
    geom_smooth()
```