Reader belonging to the workshop

for beginners

Held on 6 July 2015 at HKU

The Popularity of SPSS, Stata, SAS, R, Statistica and S-Plus from 1995-2010

graph made by R.A. Muenchen

Rinus G. Verdonschot, PhD

by Rinus G. Verdonschot, PhD – Waseda University (WIAS), Japan
Contents
Preface ................................................................................................................................. 3
Chapter 1 – Installation .................................................................................................... 4
  1.1 Introduction ............................................................................................................... 4
  1.2. Requirements and installation ............................................................................. 4
  1.3. Install packages ..................................................................................................... 5
Chapter 2 – Basic R Operations ....................................................................................... 8
  2.1. Brief Introduction of R-Studio ............................................................................. 8
  2.2. R-Expressions ...................................................................................................... 8
  2.3. R-Functions ......................................................................................................... 12
  2.4. Working with vectors ......................................................................................... 13
  2.5. Working with Matrices ....................................................................................... 14
  2.6. Working with Data Frames .................................................................................... 15
Chapter 3 – Loading data into R and working with it. ................................................... 19
  3.1. Introduction ......................................................................................................... 19
  3.2. Loading data from comma-separated files into a data frame ......................... 19
  3.3. Loading data from a portable SPSS (.por) file into a R data frame ............... 21
Chapter 4 – Creating some basic graphs in R ................................................................ 26
  4.1. Introduction ......................................................................................................... 26
  4.2. Simple bar chart created from two averages. ...................................................... 26
Chapter 5 – Basic Analyses in R compared with SPSS ................................................ 34
  5.1. T-Tests ............................................................................................................... 34
  5.2. Analysis of Variance (ANOVA) ......................................................................... 37
  5.3. Repeated measures ANOVA (2x2) ..................................................................... 38
Chapter 6 – Linear mixed effects modeling (LME) ...................................................... 43
  6.1. A bit of background on F1/F2/F’/minF’ and LME .............................................. 43
  6.2. Analyzing a semantic blocking experiment using what we know + LME ........... 43
Appendix - Index ............................................................................................................... 57
Preface

Thanks for taking part in this short workshop on getting to know R a little bit and work with it. I have to tell you honestly that I am not an expert in R, far from it actually. I only recently got to know R and I am in the process of appreciating its many features and I am learning (like many of you) how to do my own analyses with R.

There are a couple of points I would like to make. Most importantly, R is open source and open development, this means that you do not have to pay for it and its continually updated with the latest statistical knowledge. This is a huge plus for me as yearly commercial licenses can be very expensive.

Second, the biggest drawback to R most people seem to have is that it’s not as clickable as SPSS and many other commercial packages. In R one has not only to learn its language and learn some essential programming skills but one also needs to know what one is doing to analyze data properly. This is not a disadvantage or even mendokusai (as Japanese may put it) in my opinion, but rather a nice strength. It will be difficult with R to be oblivious to statistics anymore. If you ask people in the field (of course exceptions excluded) many will indicate they are “not too comfortable” with statistics, or maybe let someone else do the statistics for them and/or if they do it themselves do not know exactly what they are doing except from knowing which buttons to push in SPSS. Recently, I found myself in the same position (still am actually), but I am slowly learning more about statistical methods whilst learning more about R due to the fact that when one has to program in R one has to specify everything instead of some commercial package having everything done for you.

Lastly, there is a wonderful community out there, which is ever ready to people in need with their code and show different ways of doing something. So, if you get stuck with some analysis just “google it” and/or put your problem on a forum, and you’ll be sure to get a swift response, at least this has been my experience. And I really, really, recommend the following book for all linguists interested in statistics and R.


This reader is the product of my own ongoing learning experience in R, I am not claiming that this is the best way of learning R. Probably when an experienced R user goes through this reader s/he may cry in agony at some of the code or statements I place here. If that is the case, please read back to the beginning of this preface where I state I am a beginner too and therefore I would welcome any suggestions and improvements. Thanks a lot for your assistance.

Have fun learning R, and don’t give up!

Rinus Verdonschot, Tokyo (July 2015)
Chapter 1 – Installation

1.1 Introduction

R is a free software environment used for statistical analyses. It features all the analyses SPSS, STATA or any other commercial software can do (and more) and it has powerful graphics capabilities. However, it requires to learn the R-language, which involves learning some programming skills, in other words, R is not too clickable and perhaps even ‘user-friendly’ as for instance SPSS (but see software like http://www.rcommander.com/) and one really needs to think about what kind of analysis one is doing and the best way to go about it. In this first chapter, you will learn how to install R, install R-studio, install a package, and load a package.

The way I have written this reader is that you should preferably read through it from A-Z and do the following

1) Normal text: try to understand what is written, if you don’t understand please ask the instructor (me). Also if there is code written you should preferably try it out (but you don’t have to if you don’t want to and you understand what is written).
2) When you see TASK you have to do the task (so, try the code out yourself)
3) When you see TIP this means extra information, you should try to understand it and if possible and you’re interested: try it out yourself!
4) Don’t just blindly type out all the code in this reader without thinking about it! Try to understand it, how to write it, why you need to put c() or brackets [] or comma’s somewhere. Change small parts of the code to see what happens. If you just copy everything (like a zombie), press enter and move on to the next part, you will learn nothing. So, try to understand what you are doing.

1.2. Requirements and installation.

We will now install R and the widely used R-Studio (which makes working with R easier).

R itself → Download from: http://www.r-project.org/

For instance, select a CRAN mirror close to you (I chose Institute of Statistical Mathematics, Tokyo from the Japan list), then click on download R for windows (if you have Windows), then on base and then on the file above (the one with the version number).

R-Studio → A powerful editor which makes use of R, download from: http://www.rstudio.com/
Check which OS you have for both R and R-Studio (MAC, Windows, Linux, but don’t be afraid: R-will work on all these platforms).

<table>
<thead>
<tr>
<th>TASK:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install R by clicking on it (go through all the steps, default installation)</td>
</tr>
<tr>
<td>Install R-Studio by clicking on it (go through all the steps, default installation)</td>
</tr>
</tbody>
</table>

### 1.3. Install packages

Packages are little programs (i.e. lines of programming code) which somebody made to enhance the R experience. So, for instance, although unlikely, imagine you have to constantly add 1 to a number all the time, such as 1+1 or 2+1 or 555+1 we could write a simple function to add 1 to a certain number in the following way now using the word “addone” (this is a word I chose, it may also be some other word).

```r
> addone <- function(p) {p=p+1;return(p)}
```

This simply means that the word addone now implies a function with one argument which I called p (an argument is something you want to manipulate which you put after the function name in accolades). What it does is to take that current value (p=p) and add one to it (+1) and then place the outcome on the screen, \( \rightarrow \) return(p).

Now if I type in `addone(1)` it will also give me the result 2.

```r
> 1+1
[1] 2
> addone(1)
[1] 2
> addone(2)
[1] 3
```

Relax, don’t be scared! You don’t need to understand how to write functions, this is just an example to demonstrate what a simple function could be. The important point is that other, much more clever people than me, have written codes too which are more complex and much better than this example and allow us to do a whole lot of things (new analyses, making nice graphs, restructuring data, etc.). So, in the previous example I created a new function called `addone` which now can be used in R. However, unfortunately when we close down R and start it up again, it is gone (¡_;¡). So, if the standard version of R does not exactly do what we like, we should download additional lines of R-code such as `addone()`. This is done by downloading so-called packages in which these lines of code are stored and whenever we need them we can add them to our running version of R. So, before we start using R, we should know how you can download/install and add packages to R.

So, for instance, if we want to use linear mixed effects modeling (Chapter 6), there is a very nice package written by Bates, Maelcher and Bolker called lme4 (http://cran.r-project.org/web/packages/lme4/), which contains a lot of code which lets us run a state-of-the-art linear mixed effects analysis.

Luckily, the installation of packages is quite easy. So, let’s install the `lme4` package first.
If you start R-studio, then in the right bottom you will see the following tabs.

![Tabs in R-studio]

**TASK:**
Click on the Packages tab in R-Studio and then press Install Packages.
Type in lme4 (auto-completion usually shows us already the correct name) and then press Install

![Install Packages dialog]

What you will see then is that in the *console* window (which is actually a running version of R) some lines in the mysterious R-language have appeared.

```
> install.packages("lme4")
Installing package into 'C:/Users/Rinus/Documents/R/win-library/3.0'
(as 'lib' is unspecified)
trying URL 'http://cran.rstudio.com/bin/windows/contrib/3.0/lme4_0.999999-2.zip'
Content type 'application/zip' length 1408796 bytes (1.3 Mb)
opened URL
downloaded 1.3 Mb

package 'lme4' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
C:\Users\Rinus\AppData\Local\Temp\RtmpsFcyZJ\downloaded_packages

INFO : Sometimes, packages need code stored in other packages, luckily, R is smart enough to automatically download those packages as well (they are called dependencies).
```

Now, it looks that R-has successfully downloaded and installed this package for us and placed it on our hard drive. It’s important to point out that you could have also installed the package by typing the following line by yourself directly in the console window (which is what you would have to do if you didn’t have R-Studio):

```
> install.packages("lme4")
```
This would then have led to the same effect. However, using R-Studio’s installation method is usually easier for people who have just started to use R.

TIP: you can install more than one package at the time by placing commas in the R-Studio window.

Now we know how to install R, R-Studio and packages, we should realize that although the packages are stored on our hard drive, we cannot run the R-codes contained in them until we add them to the running version of R. This is again very easy, because we can in the right bottom window of R-Studio just put a check mark next to the package name.

It will then be included as upon checking this box R will run the following code:

```r
> library("lme4")
Loading required package: Matrix
```

TIP: You can of course also just type it in yourself (in the console window) like:

```r
> library(lme4)
```

This just means: add to library (which contains the packages loaded in the running version of R) the package lme4.

TASK:
include the lme4 package into your running version of R via the console (so, just type it in)

TIP: If you want to exclude the package, just uncheck it in R-Studio or type into the console window the following command:

```r
> detach("package:lme4", unload=TRUE)
```

What have we learnt?

- Install R
- Install R-Studio
- Install packages using R-studio
- Add and detach packages from the running version of R.
Chapter 2 – Basic R Operations

2.1. Brief Introduction of R-Studio
Now we know how to install and add packages it’s time to have “some fun” with R and R-Studio. R-Studio is a program which makes certain tasks easier as we do not need to type all the R-code involved. Also, it provides a nice overview of our data, our R commands, the packages, graphical plots and what not. This is what it looks like.

The left top of the screen provides the R-script editor, here we can type in R-commands and transfer them to the running version of R (or more technical: “the console”). We can also type it directly into the console; however, as we will be doing analyses in certain steps later it’s more convenient to write all the steps we did in the analysis in this window. As R also provides output and feedback which may obscure what we did, it’s often clearer to see the analyses steps we entered without that feedback (however, this is a personal preference feel free to do whatever you want). We will deal with how to create graphical plots and charts later, for now we will just start with the basic expressions in R.

2.2. R-Expressions

**TASK:**

Ok, first of all, let’s give R some basic expressions so we can see whether it can calculate or not:
Do this in the console window of R-studio (so actually in R directly)

```
> 1 + 5
```

Now, R should respond with this (if it doesn’t you may need to get a new computer).

```
[1] 6
```
Do the same in the R-script editor window (if you do not have this window yet, you may want to open a new script file, by going to file -> new -> r-script).

However, sadly, if you press enter nothing happens. This is because in the script window we need to press control + enter to transfer what we want to do to running version of R. So, go to line nr 1 and press control + enter (it may be a different key combination for mac users such as command + enter but I do not know that for sure). You may also simply click “run” at the right top part of this window.

Now you see that the line in the R-script editor has been transferred to the running version of R and we get the same output.

Now, go to the R-script window again and type the following command on line 2

4 + 5

Then select both lines and press control + enter

You will see that running version of R-will now run both lines of code.

> 1 + 5
> 4 + 5

So, we can run each line individually, or run them in order by selecting a range.

However, running it as a calculator is not very efficient. Usually we would like to work with variables and data structures or other objects to do our fancy analyses. An object such as a variable or data structure is a location in memory which has a name (created by you) and stores a certain amount of a specific type of information. For instance, we can store numbers (or calculation results) in one of the following ways (you can either use the equal sign = or the arrow sign -> or <-).

> a <- 1 + 5
> 1 + 5 -> a
> a = 1 + 5

Each of these 3 lines means, do the calculation 1 + 5 and store that in the letter a
Also you’ll notice that this variable will now appear in the workspace.

You can see what is inside this variable by typing its name (or click on it in the workspace)

```
> a
[1] 6
```

Now you can also calculate with this variable like a + 1, a + a etcetera.

```
> a + a
[1] 12
```

**TIP:** in the R-language + means add, - means subtract, / means divide, * means multiply, ^ or ** means exponent [e.g. $4^2=16$], % means modulus [e.g. $17\%\%4=1$].

You can also add text to a variable.

```
myText <- "I love statistics."
```

and if you type myText it will give you

```
> myText
[1] "I love statistics"
```

You can for instance add additional text to myText by doing:

```
> myText <- paste(myText, "Don't you?")
```

```
Values
    a   6
myText  "I love statistics. Don't you?"
```

**TIP:** Watch out: R is case-sensitive! So MyText and myText (or mYtExT) are in fact different variables.
TIP: Watch out! R accepts names for variables which are reserved words such as: `mean = 10` this may however conflict with the function `mean()`. To see all conflicts do the command: `conflicts()`

```r
> conflicts()
[1] "body<~" kronecker" mean"
```

And if you have a conflict, you may want to remove it with `rm()`, like in: `rm(mean)`

```r
> rm(mean)
> conflicts()
[1] "body<~" kronecker"
```

R “logically”(lol?) also accepts `logical commands` such as:

```r
> 5 > 6
[1] FALSE
> 6 > 5
[1] TRUE
```

```r
> R > SPSS #This is a Joke
```

What would be the result ;-) )

If you would like to compare whether two values are equal (for instance a versus 6) you need the double `==` sign (single `=` will not work as it puts values into objects, so a single `=` does not mean compare, it means change something for an object, like `a = 5`).

```r
> a
[1] 6
> a == 6
[1] TRUE
```

TIP: to make things a bit complicated, *if you have not defined F or T as variables*, then in R capital T initially means TRUE and capital F initially mean FALSE (by the way 0 means FALSE and 1 means TRUE too in logical expressions).

```r
> F == T
[1] FALSE
```

(e.g. F or FALSE is not the same as T or TRUE)

```r
> T == TRUE
[1] TRUE
> T == 1
[1] TRUE
```

But...

```r
> T == 0
[1] FALSE
> T = 0
> T == 0
[1] TRUE
```
You can see the difference between single = and double ==. In the last example we assigned the value 0 to the T variable and now suddenly T == 0 is no longer FALSE it becomes TRUE. Just be careful with logical expressions. Perhaps you may consider to use < - or -> instead of = to assign values to variables (although I admittingly use = a lot myself).

2.3. R-Functions

Remember the addone function we saw in Chapter One? Well, R has many built-in functions too which make our lives more comfortable. You may call a function by typing its name, followed by one or more arguments between the parentheses. Please see a comprehensive list of some important R-functions at: http://www.statmethods.net/management/functions.html

**TASK:**
Let’s add up some numbers with the sum function.

```r
> sum(1,1,2)
[1] 4
```

Let’s take the square root of a number and put it in a variable called b

```r
> b <- sqrt(16)
> b
[1] 4
```

With seq() you can generate a sequence of numbers. This function has 3 arguments: start, end and stepsize. So, seq(2,10,2) means generate a sequence from 2 to 10 in steps of 2

```r
> seq(1,10,1)
[1] 1 2 3 4 5 6 7 8 9 10
> seq(2,10,2)
[1] 2 4 6 8 10
```

Let’s get the mean of a series of numbers

```r
> mean(c(2,8,5))
[1] 5
```

We can’t do mean(2,8,5) directly because of the way the function is built up but we need to put it into a little vector (difficult name for a variable containing a sequence of something) instead which is easy by simply doing this: c(n1, n2, n3), the c() stands for concatenate, or “put together”.

However, if a certain variable was already a vector of numbers we can simply type in that variables name in mean.

```r
> a <- seq(2,10,2)
> mean(a)
[1] 6
```

**TIP:** you can see the function structure by typing help(mean) or ?mean in R, you will see the function’s structure appearing in the help portion of the right lower part of R-studio, also useful examples will be provided. You can also simply type in example(mean) to just see the examples.
I suggest you have a look at the functions described at http://www.statmethods.net/management/functions.html and try some of them out.

### 2.4. Working with vectors

R depends on vectors for many of its processes. So, it’s useful to learn how to create and manipulate them. You already saw in the last task how to create a vector using `c` (and using `seq`). Now let’s work a little bit more with it.

**TASK:**

Let’s first create a vector using a series of numbers for instance a series of five reaction times we might have gotten from a psycholinguistic experiment.

```
> RT = c(785, 621, 691, 663, 548)
> RT
[1] 785 621 691 663 548
```

Now, you can manipulate elements of this vector or the whole vector itself. For instance, if you would like to calculate the logarithm of these reaction times and store that in another vector you could simply do:

```
> logRT = log(RT)
> logRT
```

We can see now a new vector `logRT` is created and it holds the natural logarithm for each element of `RT`.

If we want to manipulate individual elements or ranges from the `RT` or `logRT` vectors this is also possible

```
> log(RT[3])
[1] 6.53814
> log(RT[3:5])
```

Gets the log from the 3rd element of `RT` or the log from the 3rd - 6th elements of `RT`.

Also one can do a logical expression, such as we only want see the elements greater than 675.

```
> RT[RT>675]
[1] 785 691
```

**TIP:** One can also exclude elements

```
> RT[c(1,5)]
[1] 785 548
```

```
RT[-c(1,5)]
[1] 621 691 663
```

This gives us the vector `RT` with only elements 1 and 5 and without them.
2.5. Working with Matrices

You can think of a matrix as a table (with rows and columns) which holds data of a certain type (numbers, strings of words, logical values, etc. you can see the type by using the `class()` command or looking in your workspace in R-Studio). The individual items in a matrix are called elements.

Let’s make our own matrix which holds data from 2 participants which each have 5 reaction times (RT) for a certain experiment.

**TASK:**

Make a vector called `subjects` containing 10 items, more specifically: 5 times 1 and 5 times 2. You can do this for instance like this:

```r
> subjects <- c(1,1,1,1,2,2,2,2,2,2)
```

but a better way is perhaps this:

```r
> subjects <- rep(1:2, each = 5)
> subjects
[1] 1 1 1 2 2 2 2 2 2 2
```

This will repeat `rep` function 5 times nr 1 and 5 times nr 2 and puts it in a vector

Now we will put some reaction times in a vector.

```r
RT = c(785, 621, 691, 663, 548, 867, 754, 674, 967, 673)
> RT
[1] 785 621 691 663 548 867 754 674 967 673
```

Now we will combine the vectors to form a 10x2 matrix

```r
> mat1 = cbind(subjects, RT)
```

`cbind()` means column bind and it will bind the two vectors together to form two columns.

You can also use row bind (`rbind`) to form a 2 x 10 matrix (so other way around)

```r
> mat2 = rbind(subjects,RT)
```

If there is no missing data you can even form a 5x2 matrix from the RT vector in which column 1 is for participant 1 and column 2 is for participant 2 (use the `ncol = x` argument where `x` is the amount of participants, or in other words the amount of columns you want).

```r
> mat3 <- matrix(RT, ncol=2)
> mat3
[,1] [,2]
[1,] 785  867
[2,] 621  754
[3,] 691  674
[4,] 663  967
[5,] 548  673
```
To access individual elements use the exact location, for instance, RT = 967 can be found in row 4 column 2:

So, in objectname[x, y] x means ROW and y means COLUMN

```r
> mat3[4,2]
[1] 967
```

We can change that value easily by doing:

```r
> mat3[4,2] <- 800
> mat3[4,2]
[1] 800
```

to access an entire row leave the column number blank

```r
> mat3[1,]
[1] 785  867
```

to access an entire column leave the row number blank

```r
> mat3[,1]
[1] 785 621 691 663 548
```

You can also use ranges, so for instance,

```r
> mat1[1:5,2]
[1] 785 621 691 663 548
```

Gives you row 1 to 5 from the second column.

### 2.6. Working with Data Frames

A data frame is a special type of matrix. The main difference between data frames and matrices is that a matrix needs all elements to be of the same type (e.g. all numbers), while data frames allow elements to have different types. Usually when you enter your research data into R it will be put into the form of a Data Frame, because you may have numeric variables (reaction times), categorical variables (conditions) and other variables. Each rows usually reflects a data point and columns usually depict the independent variables and covariates etc.

**TASK:**

Let’s make an imaginary data frame for 3 participants which are each tested on 5 items in 2 conditions of a certain experiment (so 3 x 5 x 2 = 30 rows in total). I will explain in the next chapter how you can load your own data into a data frame.

```r
> dat = data.frame(participant = rep(1:3, each = 10), condition = rep(1:2, each = 5, times = 3), RT = c(708, 685, 851, 705, 933, 826, 860, 674, 739, 682, 552, 568, 551, 611, 771, 661, 731, 801, 615, 785, 621, 691, 663, 548, 656, 672, 754, 555, 603, 583))
```

If you type dat (name of the data frame) you will see that there are 30 rows containing exactly what we specified. To see only the first five items we can use the `head` command (where n = x specifies the
amount of rows to be shown). So, to see the first five rows of the data frame “dat” we do head(dat, n = 5) (TIP: you can see last rows using the tail command)

```r
> head(dat, n = 5)
participant condition  RT
1           1         1 708
2           1         1 685
3           1         1 851
4           1         1 705
5           1         1 933
```

If you would like to see an overview (including the classes) of the different elements of the data frame use the str() command.

```r
> str(dat)
'data.frame': 30 obs. of 3 variables:
$ participant: int 1 1 1 1 1 1 1 1 1 1 ...
$ condition  : int 1 1 1 1 2 2 2 2 2 2 ...
$ RT         : num 708 685 851 705 933 826 860 674 739 682 ...
```

As you can see there is a dollar sign ($) in front of the different column names of the data frame. This is the way we can access the data by using `dataframenam$columnname` so, to access the RTs we do:

```r
> dat$RT
[1] 708 685 851 705 933 826 860 674 739 682 552 568 551 611 771 661 731 801 615 785 621
[22] 691 663 548 656 672 754 555 603 583
```

Also, we can use similar code as with matrices to access its elements.

So, let’s say we want to see only element 10-20 from the reaction times we do:

```r
> dat$RT[c(10:20)]
[1] 682 552 568 551 611 771 661 731 801 615 785
```

Let’s say we want to change the first reaction time from participant 2 condition 2, that will be element nr 16 so we simply access it by doing the following:

```r
> dat$RT[16]
[1] 661
> dat$RT[16] <- 500
> dat$RT[16]
[1] 500
```
TIP: Although a bit tedious (and not as efficient as excel), you can edit tables directly by using the \texttt{fix(name)} command. You can’t paste data from excel into this data editor however.

\begin{verbatim}
> fix(dat)
\end{verbatim}

will give you the editor as shown in the right panel

NOTE: many times this does not work on MAC. See: \url{http://stackoverflow.com/questions/26489928/cant-load-x11-in-r-after-os-x-yosemite-upgrade} for a possible solution (i.e., upgrade XQuartz).

Also, you could try to edit part of the data file like

\begin{verbatim}
> edit(dat$RT)
\end{verbatim}

TIP: if you need help with a specific command you can simply type

\begin{verbatim}
help(name of command) or ?(name of command)
\end{verbatim}

so, if you want to know what \texttt{str} exactly does you can type either

\begin{verbatim}
> help(str)
> ?str
\end{verbatim}

and you will see the info about \texttt{str} in the right bottom of R-studio:

\begin{verbatim}
str> require(stats); require(grDevices); require(graphics)
str> ## The following examples show some of 'str' capabilities
str> str(1:12)
int [1:12] 1 2 3 4 5 6 7 8 9 10 ...
\end{verbatim}

Also if you’re simply interested in examples how to use a certain command you can type:

\begin{verbatim}
> example(str)
\end{verbatim}
specific package you will also not get the help for commands contained in them. So, if you did not load the plyr package and if you try to access the help for join() (which is part of plyr package) you will get

```r
> help(join)
No documentation for join in specified packages and libraries:
```

If that happens, you may either not have loaded plyr or made a misspelling or something.. luckily there is a search function which allows you to search over all options called ?? just type ?? followed by a command name.

```r
> ??join
```

In this chapter you have learned to:

- calculate with R
- store numbers and text in variables
- use simple functions (sum, seq, mean)
- create and manipulate vectors, matrices and data frames
- get important information via help(), example(), ? and ??
Chapter 3 – Loading data into R and working with it.

3.1. Introduction
Most colleagues I have are using excel or another spreadsheet program and will copy their data via some kind of copy-paste action into SPSS. This is going to be a little bit different in R. As this is sometimes a big turn-off for people and stops them from using R as they want to have an easy way of inputting their data into a statistical program and also see an overview of their data in which they can click and manipulate. This chapter will explain how you can enter your data into R and how to manipulate it which will hopefully make everything a bit easier. In paragraph 3.3 there is also a tip how to use R-Studio to import CSV data without the need of using any R-Commands.

3.2. Loading data from comma-separated files into a data frame
The easiest way to get your data in an R-data frame is to already prepare your data in Excel or Libre Office or some other package. Naturally, R has the possibility to merge or bind data from various sources together (which we will talk about at the end of this chapter) but for most beginning users (including me) it’s easiest to have everything prepared in advanced.

Ok, let’s say we have simply an excel sheet with three columns, one shows the participant number, one represents the experimental manipulation and the last one the control condition. How do we get this in a data frame?

Well, that’s very easy. First save it as a comma delimited file in Excel (try to ignore the Microsoft blablabla about only saving the active sheet and that some features are not available).

Then let’s load it. You’d best save the file in your current working directory for R, you can know which one this is by typing `getwd()`

```
> getwd()
[1] "C:/Users/Rinus/Documents"
```

However, you can change this by doing `setwd()` (Oh, if you do this, the path will be different of course than the one mentioned now)

```
> setwd("C:/Users/Rinus/Documents/R-Course/")
> getwd()
[1] "C:/Users/Rinus/Documents/R-Course"
```

For MAC/Linux its something like `setwd("/Users/Rinus/Documents/R-Course/")`
(So, no C: is necessary)
TASK:

The workshop makes use of some files prepared for you in advance. There should be a couple of USB sticks “going around” in the classroom but the files can also be downloaded from: [http://www.languageandcognition.nl/rinusverdonschot/Rworkshop.zip](http://www.languageandcognition.nl/rinusverdonschot/Rworkshop.zip)

First job is to create a directory on your computer and call it R-Course or something and put these files into that directory (unzip if you downloaded them from the internet).

Subsequently, you’ll need to make this directory the standard working directory of R. If you use `setwd()` as mentioned before if you close and open R it’ll change to the standard working directory again. So, for a permanent setting please go to TOOLS -> (GLOBAL) OPTIONS in R-Studio.

And on the general tab, go to default working directory, click browse and select the directory you just made, press APPLY and OK. Then close R-Studio and open it again, and your default working directory will be the one you just selected.

When not sure what files are in the current working directory one can do:

```r
> list.files()
[1] "Chapter3_2.csv" "Chapter3_3.por" "Chapter3_3_2.csv" "Chapter4_2.csv"
[5] "GenCon26pp.csv" "GenCon26ppV2.csv" "lexvar.csv"
"SemBlockingExportR.csv"
```

Now, it is time to load a data set (using `read.csv`)

```r
> dat = read.csv("Chapter3_2.csv")
```

TIP: you can also use `read.table` and specify the separator (in this case the comma, but you can also use tabs as in: `sep="\t"` if your file is tab delimited) such as in :

```r
> dat = read.table("Chapter3_2.csv", header = T, sep = ",")
```

And voila, the data is loaded in R now. However, don’t break out the champagne yet. Let’s check everything went well.

```r
> str(dat)
'data.frame': 20 obs. of  3 variables: 
$ PPN: int  1 2 3 4 5 6 7 8 9 10 ...
$ EXP: num  554 609 633 642 515 ...
$ CON: num  589 634 639 644 542 ...
```
It seems to have worked out. We will talk about how to prepare and analyze your data in one of the next chapters.

### 3.3. Loading data from a portable SPSS (.por) file into a R data frame

First, make sure that you save the SPSS file in the portable format (.por). This format is easy to read for R when using a special package which we need to install now. **TASK:** Make sure you install the *Hmisc* and the *foreign* packages (see Chapter 1 how to do that). **TIP:** how to import data from other programs can be found at [http://www.statmethods.net/input/importingdata.html](http://www.statmethods.net/input/importingdata.html)

If successful it should say:
- package ‘Hmisc’ successfully unpacked and MD5 sums checked
- package ‘foreign’ successfully unpacked and MD5 sums checked

Now load the packages into R by checking them in R-Studio or by using the following commands

```r
> library(Hmisc)
> library(foreign)
```

Now, using a similar command as the comma separated file we can do the following command.

```r
> dat2 <- spss.get("Chapter3_3.por", use.value.labels=TRUE)
> str(dat2)
'data.frame': 20 obs. of 3 variables:
$ PPN:Class 'labelled' atomic [1:20] 1 2 3 4 5 6 7 8 9 10 ... 
  ..- attr(*, "label")= Named chr " " 
  ..- attr(*, "names")= chr "PPN"
$ EXP:Class 'labelled' atomic [1:20] 554 609 633 642 515 ... 
  ..- attr(*, "label")= Named chr " " 
  ..- attr(*, "names")= chr "EXP"
$ CON:Class 'labelled' atomic [1:20] 589 634 639 644 542 ... 
  ..- attr(*, "label")= Named chr " " 
  ..- attr(*, "names")= chr "CON"
```

We can see that it's the same from the comma separated one, however, somehow it saves data with 4 digits behind the comma (a bit more precise) which may be actually my bad ^_^ as I copied it from SPSS to Excel to create the CSV file.
```r
> head(dat2, n = 3)
PPN      EXP      CON
1   1 553.5926 589.0000
2   2 608.8966 633.7333
3   3 632.7143 639.2143
```

TIP: If you change directories often between projects or if you want a small explorer type window to pop up to look for your files you can also use the `file.choose()` command instead of specifying the whole file name. For instance, if you use

```r
> dat = read.csv(file.choose(), header = TRUE)
```

or

```r
> dat2 <- spss.get(file.choose(), use.value.labels=TRUE)
```

Then a popup will appear allowing you to select the file (TIP: sometimes the pop-up appears behind the R-Studio screen so you may want to minimize the screen)

TIP: If you do not like using R-commands that much or you’re more of a graphical user interface type of person, you can also load files holding data to a data frame from within R-Studio by going to the workspace window and press import data set. This works only with text files (like csv) and not with SPSS files.

Then select the correct dataset (.csv) file, press next, and you will see the following screen:
Now you can give the data frame a name (in this case we’ll call it dat3) and select what type of file you have, which separator etc. and you will see the result appearing in the data frame window, if you’re satisfied you can press import and the file will be imported into the data frame.

**TIP:** how to merge two files together in R.

Suppose you have your experimental output. But for some reason you need to include some lexical information such as word frequency or word length into an analysis. However, this information is stored in another spreadsheet. With R you can easily merge them together using the (no surprise) `merge()` command.

**TASK:**
First load in the experimental data contained in chapter3_3_2.csv in a dataframe (call it dat)

```r
> dat = read.csv("Chapter3_3_2.csv")
> dat
```

<table>
<thead>
<tr>
<th>trialOrder</th>
<th>participant</th>
<th>condition</th>
<th>RT</th>
<th>itemNr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>708</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>685</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>705</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1933</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>826</td>
</tr>
</tbody>
</table>

Let’s say there are only 5 items which are named twice under each condition.
Now we would like to add the frequency and names of these items which are stored in another database called `lexvar.csv`
Load this database in another dataframe (call it lexvar or something).

```r
> lexvar = read.csv("lexvar.csv")
> lexvar
itemNr itemName itemFreq itemLen
1 1 boat 15857 4
2 2 house 18633 6
3 3 table 1625073 3
4 4 cat 86253 3
```

Now we merge the two lexical frames to include the additional information to the experimental dataframe.

```r
> datwithlex=merge(dat, lexvar, by="itemNr")
> head(datwithlex, n = 4)
itemNr trialOrder participant condition  RT  itemName itemFreq itemLen
1 1 1 1 1 1 1 1 1 1 708 boat 15857 4
2 1 10 2 1 2 682 boat 15857 4
3 1 5 2 2 1 771 boat 15857 4
4 1 6 2 2 2 661 boat 15857 4
```

You can see now that the three lexical variables (name, frequency and length) are added to the datwithlex dataframe.

You’ll also notice that the original ordering changed a bit (changed to ordering by itemNr because of the merge) if you want to change it back to the original order that’s possible (but not really necessary for the analysis).

```r
> datwithlex = datwithlex[with(datwithlex,order(participant,trialOrder)),]
```

Another, actually easier, way is to use the join function from the *plyr* package. It will simply keep the order of the first mentioned database. Remember from the first chapter we need to install that *plyr* package first. You should be able to do that now.

by Rinus G. Verdonschot, PhD – Waseda University (WIAS), Japan
Then, load the package in the running version of R (using library)

```r
> library(plyr)
> datwithlex = join(dat, lexvar)
Joining by: itemNr
> head(datwithlex, n = 4)
```

<table>
<thead>
<tr>
<th>trialOrder</th>
<th>participant</th>
<th>condition</th>
<th>RT</th>
<th>itemNr</th>
<th>itemName</th>
<th>itemFreq</th>
<th>itemLen</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1708</td>
<td>1</td>
<td>boat</td>
<td>15857</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1685</td>
<td>3</td>
<td>table</td>
<td>1625073</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1851</td>
<td>2</td>
<td>house</td>
<td>18633</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1705</td>
<td>4</td>
<td>cat</td>
<td>86253</td>
<td>3</td>
</tr>
</tbody>
</table>

Now, you see that the two csv files are merged together on the basis of the common column itemNr and the lexical variables have been added to the dataframe.

What have we learned?

- Know how to set the current and permanent working directories
- Loading data from csv and spss files into R using commands and R-studio
- Merging objects together in two different ways
Chapter 4 – Creating some basic graphs in R

4.1. Introduction
One of the nice features of R is its ability to create attractive graphs. Although anything seems to be possible with R, I will sadly keep to the most basic (and perhaps foulest) of graphs such as bar charts, scatterplots and pie-charts (as this is an introductory text).

4.2. Simple bar chart created from two averages.

<table>
<thead>
<tr>
<th>TASK:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let’s again use the dataset we had in Chapter 3.2. containing two simple columns</td>
</tr>
</tbody>
</table>

```r
> dat = read.csv("Chapter3_2.csv")
> head(dat, n = 4)
PPN  EXP  CON
1  1 553.59 589.00
2  2 608.90 633.73
3  3 632.71 639.21
4  4 641.63 643.83
```

Okay, let’s for now make a very simple two column bar chart from the averages from column EXP and CON.

```r
> barplot(c(mean(dat$EXP),mean(dat$CON)))
```

What we first did is to create a vector with two elements (mean EXP and mean CON from the dat dataframe) and put that in the `barplot()` command. This gives us the very ugly barplot as can be seen on the left side in the figure below. If we want to change the X and Y axis scale values we can type those into the command.

```r
> barplot(c(mean(dat$EXP),mean(dat$CON)), ylab="RT", ylim=c(500,700), xpd=F, main="Our RTs")
```

This narrows our Y axis to 500-700 range. We need to add `xpd=False` otherwise the bars will trespass the region specified in the Y-axis, you can try to do the command without this `xpd=F` command and you’ll see. Also, we gave the chart already a name (using `main`) and a label for the X-axis (using `ylab`).

```
1.  
2.  
3.  
```

Now, lastly, we want to add labels for the two columns (EXP and CON).

```r
> barplot(c(mean(dat$EXP),mean(dat$CON)), ylab="RT", ylim=c(500,700), xpd=F, main="Our RTs", axisnames = T, names.arg=c("EXP", "CON"))
```

What happens here is that we put `axisnames = T` (T means true, so -> there will be axisnames) and
names.arg = the arguments for the axis so in our case EXP and CON (but we can name it anything we want).

```r
> barplot(c(mean(dat$EXP),mean(dat$CON)), ylab="RT", ylim=c(500,700), xpd=F, main="Our RTs", axisnames = T, names.arg=c("EXP", "CON"))
```

So, let’s dissect this last and kinda long command a bit into its elements.

1. barplot -- create a barplot
2. c() -- create a vector of something (this c will appear a couple of times more in this example)
3. mean(dat$EXP), mean(dat$CON) -- the 2 values in the vector (means of the columns)
4. ylab = “RT” -- name of the y-axis, we call it RT
5. ylim = c(500,700) -- c() once again creates a vector, this time two values, 500-700, and puts that in ylim (the limit of the y axis)
6. xpd = F -- (you can also write xpd = FALSE), this means the bars should not cross ylim/xlim borders
7. main = “Our RTs” -- this gives the whole graph a name
8. axisnames = T -- we are going to use a name for the x-axis (therefore True)
9. names.arg = c("EXP", "CON") -- we have two columns so we’ll specify a vector with two names, in this case EXP and CON and put that in names.arg which will pass it to axisnames.

TIP: Oh, if you want a nice box around the plot.. just do `box()`:

```r
> box()
```

TIP: if you have your data in another format such as a long format (as shown in Chapter4_2.csv file) you may try to use the following code:

```r
> dat2 = read.csv("Chapter4_2.csv", header = T)
> barplot(tapply(dat2$RT,list(dat2$Condition),mean), ylab="RT", ylim=c(500,700), xpd=F, main="Our RTs", axisnames = T, names.arg=c("EXP", "CON"))
```

So, now we know how to create a simple “excel-style” graph using the basic `barplot()` function (standard in R) which most people are familiar with.

Okay, let’s try to work a bit with a more realistic dataset now (will also appear in Chapter 6). Imagine the following data has been gathered from 24 students in Beijing when they were naming words and pictures in various blocks which either contained similar items such as: sheep, cat, rabbit, bear, horse (homogeneous blocks) or different items, such as: sheep, violin, plate, crossbow, nose (heterogeneous blocks).

**TASK:**

Let’s first load the dataset.

```r
> dat = read.csv("SemBlockingExportR.csv", header = T)
> str(dat)
'data.frame': 20000 obs. of 14 variables:
$ ppn : int 5 5 5 5 5 5 5 5 5 5 ...
$ classi : Factor w/ 5 levels "animals","body",...
```

by Rinus G. Verdonschot, PhD – Waseda University (WIAS), Japan
Let's take only the RTs which are accurate for now
This can be easily done by:

```R
> dat2 = dat[dat$HanziPicErr == 1,]
> nrow(dat)-nrow(dat2)
[1] 530
```

This means: create a new dataframe called dat2, and put in it all the data from ‘dat’ where dat$HanziPicErr equals 1 (remember == means “equals to” 1 in this case being a code which we put in the data set referring to “correct item”) so in essence everything else than 1 will not be put in the new dataframe called dat2.

nrow (number of row; so count number of rows from dat and dat2 and subtract them) means that we now know that 530 points have been excluded from dat2 compared to dat.

Let’s now for the sake of simplicity forget about the Chinese character naming and only concentrate on the Chinese picture naming part of the experiment. We do this by creating a new data frame including only PictureProc items and while we’re at it let’s get rid of some outliers for now by excluding RTs smaller than 200 and larger than 1500.

```R
> dat3 = dat2[dat2$Proc == "PictureProc",]
> dat4 = dat3[dat3$HanziPicRT > 200 & dat3$HanziPicRT < 1500,]
```

Ok, let’s see how the data looks for the homogeneous groups with a simple plot command.

```R
> plot(dat4$HanziPicRT[dat4$condition =="hetero"])
```

Which will give us the following, rather foul, plot (which looks like a ‘rainy day in data-land’)
So, this plot represents all data points in the picture naming condition for the heterogeneous groups. You can already see that most points are concentrated around 400-800 ms (thick black parts).

However, a kinder way to visualize this is to use a density plot which shows the frequency distribution more clearly and thereby makes it much easier to see what’s going on just by adding the word “density” to the plot command, like in:

```
plot(density(dat4$HanziPicRT[dat4$condition =="hetero"]))
```

Now, adding even more to the fun we can supplement the homogeneous distribution to this plot using the `lines()` command.

```
lines(density(dat4$HanziPicRT[dat4$condition =="homo"], col = "red")
```

This plot already hints that the homogeneous distribution is shifted to the right a small bit compared to the heterogeneous distribution, indicating perhaps an effect of group.

Another way to investigate this is to create a distribution curve by ordering the RTs from low to high (using the `sort` command), and placing the two conditions on the same plot (for instance with a separate color, such as: col = “red”, to make it visually more easy to inspect).

```
> plot(sort(dat4$HanziPicRT[dat4$condition =="hetero"]), type = "l")
> lines(sort(dat4$HanziPicRT[dat4$condition =="homo"]), col = "red")
```
We get a plot like this, now, the cool thing is to realize (hopefully I’m not mistaken) is that the distance between these two lines is actually the effect between the two conditions (heterogeneous in black and homogeneous in red). In this case there seems to be an effect most of the range of the stimuli. Sometimes for instance, you only get effects when people get really slow (slower or right side of the distribution) in that case the lines would be overlapping except for the last bit where they split, but not in this case.

Also, we can also use points instead of lines, and put a nice title, name the axes and everything.. We first need to start a plot by using the `plot` command, but then we can use subsequent commands such as `lines()` or `points()`, to add stuff to the plot.

**TASK:**

Imagine you have ran a picture naming experiment between mono- and bilinguals. There were 3 types of distractors placed on the pictures (semantically related, semantically unrelated and control distractors). The averaged reaction times are placed in two vectors.

```r
> monolinguals = c(655, 596, 612)
> bilinguals = c(639, 570, 582)
```

Now, the nice thing is that with R you can build up a plot from scratch. So, you can get exactly what you wanted. First of all we will plot the monolingual values, using both lines and dots (type = 0), in blue (col = “blue”) with the circle symbol (pch = 21), the limits of the y-axis will be from 500 to 700ms (ylim = c(500,700)), and we will in the beginning not present the axes nor the annotation (axes=F, ann=F) because we will create them ourselves.

```r
> plot(monomlinguals, type="o", col = "blue", pch=21, ylim = c(500,700), axes = F, ann=F)
```
So, what we end up with is only a blue line with dots denoting the monolingual averaged RTs for the three conditions.

We will now add the values of the bilinguals, in red (col="red") also using lines+dots (type="o") and with another line type namely dotted lines (lty=4) and a square to denote the points (pch=22).

```r
> lines(bilinguals, type="o", lty=4, pch = 22, col = "red")
```

So, now we have two lines for both mono- and bilinguals. But no axis or any other information.

```r
> axis(1, at=1:3, lab=c("SEM+","SEM-", "CON"))
> axis(2, las=1, at=c(500,600,700))
```

These commands adds the X axis (axis number 1) at the three points (at=1:3) and name them SEM+, SEM- and CON. Subsequently, the second axis (y-axis) will be constructed, and horizontally (las=1) not vertically it will show 500, 600 and 700 ms as axis values.

```r
> title(main="myExperiment", xlab = "Condition", ylab="RT")
> box()
```

This will put a box around the plot and adds a title and axis labels.

Lastly, we add a legend to indicate what the lines indicate.

```r
> legend(2.35, 705, c("monolinguals","bilinguals"), col=c("blue","red"), pch=21:22, lty=c(1,4))
```

The first values (2.35 and 705) indicate the position of the legend in the graph. So, it starts at x = 2.35 position and y = 705 position. This means right top. If you want to do it left bottom for instance you may start the command as:

```r
legend(1, 580, c("monolinguals","bilinguals"), col=c("blue","red"), pch=21:22, lty=c(1,4))
```

Subsequently, we name the parameters of the legend (monolinguals, bilinguals) denote the colors which we used (blue and red) and the icons which we used for the points (21,22) and the line types (lty 1 and 4).

We should then end up with the following figure constructed by us from scratch. Feel free to experiment and change a lot of stuff depending on your personal wishes.
Another package which allows us to plot very nice graphics is called ggplot. We will not discuss it here but it should be mentioned as it is a very powerful package.

First install `ggplot()` (`ggplot2` is the actual package name) by using the packages tab in R-studio or simply typing:

```r
> install.packages("ggplot2")
```

And loading it into R by checking it in the packages tab or typing:

```r
> library(ggplot2)
```

`ggplot()` is also normally used to incrementally construct a plot. As such it uses the `+` operator to add layers (esthetics) to an existing `ggplot` object. This can be beneficial in the sense that the code is explicit about which layers should be added and the specific order in which they should be added.

Some nice information you can find about ggplot is here.

[http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2))

**TASK:** Lastly, I'll show you to construct a very simple pie chart

Imagine you’re in a in a one hour meeting and we record how much time everybody actually speaks in a vector. I-Fan (35m), Brendan (15m), Mehdi (5m) and Rinus (5m).

```r
> SpeakingTime = c(35, 15, 5, 5)
```

You can make a nice pie graph now showing you the proportion of speaking time.

```r
> pie(SpeakingTime, main="SpeakingTime", col=rainbow(4), labels=c("I-Fan", "Brendan", "Mehdi", "Rinus"))
```
If you're really brave though you may also use the percentages which you calculate and put in a vector. (round, <something>, <number> means round <something> off using this <number> amount of digits behind the comma).

```r
> lengthspeaking = round((SpeakingTime/60)*100,1)
```

and then create the pie (using lengthspeaking for the labels) and using a legend (position -.6 and 0.55 so in middle of pie) to denote who belongs to the color).

```r
> pie(SpeakingTime, main="SpeakingTime", col=c("red","green","purple","blue"), lengthspeaking)
> legend(-.6,0.55, c("I-Fan", "Brendan", "Mehdi", "Rinus"), fill = c("red","green","purple", "blue"))
```

- In this chapter we learned to construct basic graphs such as bar charts, plots and pie charts.
Chapter 5 – Basic Analyses in R compared with SPSS

In this chapter we will compare a couple of basic analyses to do in R and compare the results with SPSS output which most people are familiar with.

5.1. T-Tests

**Independent t-test.** In this test we will compare 2 groups of cases with each other. For instance, Japanese people and Dutch people doing the same mathematics test. The question is whether Japanese or Dutch score similarly or different on this test.

**TASK:**

10 Japanese and 10 Dutch students took the math test (Score Range can be between 1-10)

```
mathscore = data.frame(Score = c(10, 8, 8, 7, 8, 8, 8, 8, 6, 6, 7, 6, 5, 6, 8, 7), Nationality = c(rep("Japanese", times = 10), rep("Dutch", times = 10)))
```

**TIP:** If you do not already understand, please try again to really understand what the above command does.

You may of course also type the data out in excel and then import it into R.

If you do an independent samples t-test in SPSS on this data you’ll get the following result:

**T-Test**

```
Group Statistics

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>Std Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>10</td>
<td>7.600</td>
<td>1.549</td>
<td>0.4893</td>
</tr>
<tr>
<td>Score</td>
<td>20</td>
<td>6.400</td>
<td>1.349</td>
<td>0.4266</td>
</tr>
</tbody>
</table>

Independent Samples Test

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>Sig</th>
<th>t</th>
<th>df</th>
<th>Sig (2-tailed)</th>
<th>Mean Difference</th>
<th>Std Error Difference</th>
<th>%5% Confidence Interval of the Difference Lower</th>
<th>Upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equal variances assumed</td>
<td>0.00</td>
<td>1.000</td>
<td>2.155</td>
<td>18</td>
<td>0.05</td>
<td>1.4000</td>
<td>0.64979</td>
<td>0.3485</td>
<td>2.78015</td>
</tr>
<tr>
<td>Equal variances not assumed</td>
<td>0.00</td>
<td>1.000</td>
<td>17.669</td>
<td>0.05</td>
<td>1.4000</td>
<td>0.64979</td>
<td>0.0391</td>
<td>2.78959</td>
<td></td>
</tr>
</tbody>
</table>
```

So, there is a significant difference between groups t(18) = 2.155, p < .05 with Japanese scoring higher.
than Dutch on the test (;)}. If you do the same analysis on this data in R you'll get the same results.

```r
> t.test(mathscore$Score ~ mathscore$Nationality)
Welch Two Sample t-test

data:  mathscore$Score by mathscore$Nationality
t = -2.1546, df = 17.669, p-value = 0.04526
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-2.76698502 -0.03301498
sample estimates:
mean in group Dutch mean in group Japanese
6.4                    7.8
```

Also you can get the results assuming equal variances

```r
> t.test(mathscore$Score ~ mathscore$Nationality, var.equal=T)
Two Sample t-test

data:  mathscore$Score by mathscore$Nationality
t = -2.1546, df = 18, p-value = 0.04499
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-2.76515034 -0.03484966
sample estimates:
mean in group Dutch mean in group Japanese
6.4                    7.8
```

TIP: This of course comes from reviewing Levene’s test (which can be found in the lawstat package) which you can also get with R using

```r
> install.packages("lawstat")
> library("lawstat")
> levene.test(mathscore$Score, mathscore$Nationality)

modified robust Brown-Forsythe Levene-type test based on the absolute deviations from the median

data:  mathscore$Score
Test Statistic = 0, p-value = 1
```

In this case you can see that assuming equal variances is not a problem as Levene’s test is not significant.

**Paired t-test.** In this test measurements on the same or similar variable are compared over time or over different circumstances. For instance, let’s say Dutch students would take a similar math test twice would they score better the second time?

**TASK:** Ok, well first create some fake second test scores for the Dutch population.
You could of course also type it out in excel and then import it into R (for practice).

```
mathscore2 = data.frame(First = c(9, 5, 6, 7, 6, 5, 6, 5, 8, 7), Second = c(8, 6, 8, 6, 7, 6, 5, 9, 8, 8))
```

So, with SPSS we would get the following results.

### T-Test

#### Paired Samples Statistics

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>N</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>First</td>
<td>5.4000</td>
<td>10</td>
<td>1.34990</td>
<td>.47667</td>
</tr>
<tr>
<td>Second</td>
<td>7.1000</td>
<td>10</td>
<td>1.28689</td>
<td>.40669</td>
</tr>
</tbody>
</table>

#### Paired Samples Correlations

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Correlation</th>
<th>Sig</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1 First &amp; Second</td>
<td>10</td>
<td>0.264</td>
<td>0.209</td>
</tr>
</tbody>
</table>

#### Paired Samples Test

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
<th>95% Confidence Interval of the Difference</th>
<th>t</th>
<th>df</th>
<th>Sig (2-tailed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>First - Second</td>
<td>-0.7000</td>
<td>1.56702</td>
<td>.49564</td>
<td>-1.82099</td>
<td>-1.413</td>
<td>9</td>
<td>.191</td>
</tr>
</tbody>
</table>

And with R we get of course exactly the same (the paired = T option means that it is now a paired sample t-test in contrast to an independent samples test).

```r
> t.test(mathscore2$First, mathscore2$Second, paired = T)
```

```
Paired t-test

data:  mathscore2$First and mathscore2$Second
t = -1.4126, df = 9, p-value = 0.1914
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:  
-1.8209795 0.4209795
sample estimates:
mean of the differences
-0.7
```
5.2. Analysis of Variance (ANOVA)

Okay, we have so far been working with two groups. How about when we have to compare 3 or more means. Then we bring out the good ol’ ANOVA of course.

Let’s say that we want to know whether there is a significant difference between three groups who scored on that math test, for instance: Japanese, Chinese and Korean students.

**TASK:** Ok let’s take 10 students of Japanese, Chinese and Korean nationality and compare their scores.

<table>
<thead>
<tr>
<th>Score</th>
<th>Nationality</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Japanese</td>
</tr>
<tr>
<td>8</td>
<td>Japanese</td>
</tr>
<tr>
<td>8</td>
<td>Japanese</td>
</tr>
<tr>
<td>7</td>
<td>Japanese</td>
</tr>
<tr>
<td>8</td>
<td>Japanese</td>
</tr>
<tr>
<td>8</td>
<td>Japanese</td>
</tr>
<tr>
<td>10</td>
<td>Japanese</td>
</tr>
<tr>
<td>5</td>
<td>Japanese</td>
</tr>
<tr>
<td>6</td>
<td>Japanese</td>
</tr>
<tr>
<td>5</td>
<td>Chinese</td>
</tr>
<tr>
<td>7</td>
<td>Chinese</td>
</tr>
<tr>
<td>8</td>
<td>Chinese</td>
</tr>
<tr>
<td>8</td>
<td>Chinese</td>
</tr>
<tr>
<td>6</td>
<td>Chinese</td>
</tr>
<tr>
<td>8</td>
<td>Chinese</td>
</tr>
<tr>
<td>10</td>
<td>Chinese</td>
</tr>
<tr>
<td>9</td>
<td>Chinese</td>
</tr>
<tr>
<td>7</td>
<td>Chinese</td>
</tr>
<tr>
<td>5</td>
<td>Chinese</td>
</tr>
<tr>
<td>10</td>
<td>Korean</td>
</tr>
<tr>
<td>7</td>
<td>Korean</td>
</tr>
<tr>
<td>9</td>
<td>Korean</td>
</tr>
<tr>
<td>10</td>
<td>Korean</td>
</tr>
<tr>
<td>7</td>
<td>Korean</td>
</tr>
<tr>
<td>9</td>
<td>Korean</td>
</tr>
<tr>
<td>6</td>
<td>Korean</td>
</tr>
<tr>
<td>9</td>
<td>Korean</td>
</tr>
<tr>
<td>7</td>
<td>Korean</td>
</tr>
<tr>
<td>7</td>
<td>Korean</td>
</tr>
</tbody>
</table>

> mathscore = data.frame(Score = c(10, 8, 8, 7, 8, 10, 5, 6, 5, 7, 8, 8, 6, 8, 10, 9, 7, 5, 10, 7, 9, 10, 7, 9, 6, 9, 7, 9), Nationality = c(rep("Japanese", times = 10), rep("Chinese", times = 10), rep("Korean", times = 10)))

Okay, SPSS gives us the following output.
So, the effect is not significant, and also using options we can get descriptives in SPSS.

How about R? The ANOVA command is `aov()` and we will put the result of the analysis in another data frame. You can call that whatever you like, but I use `mathscore.aov = aov` (do analysis of variance) with `Score ~ (depends on) Nationality, from the dataset mathscore). Or in other words, the command will look like this:

```r
dataframenname.aov = aov(dependent variable ~ independent variable, data = dataframenname)
```

```r
> mathscore.aov = aov(Score~Nationality, data = mathscore)
> summary(mathscore.aov)
Df Sum Sq Mean Sq F value Pr(>F)
Nationality 2 1.633 0.817 1.863 0.154
Residuals 27 64.60 2.393
> print(model.tables(mathscore.aov, "means"), digits=3)
Tables of means
Grand mean
7.733333
Nationality
Chinese Japanese Korean
7.3 7.8 8.1
```

So, again we see essentially the same results using R which again gives us a reason to abandon commercial packages and go for free open source! ;-)

5.3. Repeated measures ANOVA (2x2)

In a repeated measures design, the same participant takes part in all conditions in an experiment. In the experiment we will use for this analysis (experiment was conducted in Dutch) participants had to name pictures with distractor words being superimposed on them. In the design, two things were manipulated, namely the semantic relationship and the gender of the distractor words.
To put it more clearly, in this experiment participants named the same picture four times “de hond / the dog” using a determiner noun phrase, but there was a different distractor word on it all the time either semantically related (kikker/frog and konijn/rabbit) or not (doos/box and vat/barrel) and additionally had same or different gender (common vs. neuter). This amounts to a 2x2 repeated measures design.

**TASK:**

A 2x2 repeated measures ANOVA is not always that easy to run in R, so let’s examine two methods to do that, a more “difficult” one when you have data in the aggregated SPSS format and an easier one when you can make columns for the experimental factors (conditions).

**Method 1: using Anova()**

We need to prepare our data a bit in advance (like we also need to do that with SPSS), so that every line is a participant and the columns hold the RTs for the various conditions.

Let’s first of all load the data into a data frame.

```r
dat = read.csv(file="GenCon26pp.csv", header = T)
> head(dat, n = 5)
   ppm SpGp SmGp SpGm SmGm
1  1  889.06 823.89 832.86 830.29
2  2  765.05 738.44 785.53 757.87
3  3  867.87 866.23 941.90 938.13
4  4  699.38 694.52 720.20 674.54
5  5  855.74 820.45 813.21 741.84
```

We can see that the data have already been averaged/aggregated into the four respective conditions (S = semantics, G = gender, p = related, m = unrelated).

Now we will use the Anova() command from the car package so we should install and load the car package first (so, go to: packages, install packages from the right bottom pane in R-studio to install the car package). Then add it to the running version of R using the library command.

```r
> library(car)
```

Now, we need to set up a matrix of factor codes for the repeated measures variables to use in our model. We do this by defining a factor for semantics (factorS) and gender (factorG) and put all our combinations in it. We then combine them into a data frame called modelFactors.
```
> factors = c("Sp", "Sm", "Sp", "Sm")
> factorG = c("Gp", "Gp", "Gm", "Gm")
> modelFactors = data.frame(cbind(factors, factorG))
> modelFactors
  factors factorG
  1     Sp     Gp
  2     Sm     Gp
  3     Sp     Gm
  4     Sm     Gm
```

So this data frame specifies the factors in the model which we will use in our Repeated Measures ANOVA. This is actually similar to specifying the model in SPSS like in the following picture:

![SPSS Repeated Measures ANOVA](image)

Yay!! This looks familiar...懐かしい!

Let’s also practice attaching our data (attaching actually just means we do not need to type the name of the particular object, e.g. like our data frame dat$, before a column name all the time anymore).

```
> attach(dat)
```

Now we will use ‘`lm()` (a function to fit linear models) to calculate the multivariate model we want. TIP: ‘`cbind`’ combines the columns into a vector of all the data columns of the repeated measures variables.

```
> datmodel = lm(cbind(SpGp, SmGp, SpGm, SmGm)~1)
> datmodel
Call:
lm(formula = cbind(SpGp, SmGp, SpGm, SmGm) ~ 1)
Coefficients:
 (Intercept)    SpGp    SmGp    SpGm    SmGm
            779.4   766.0   796.4   774.5
```

Now we attach the modelFactors (specifying the design) we made earlier too and run the Anova() on the model.

```
> attach(modelFactors)
> GenConRepMes=Anova(datmodel,idata=modelFactors,idesign=~ factorS*factorG, type="III")
> summary(GenConRepMes,multivariate=F)
```

```
Univariate Type III Repeated-Measures ANOVA Assuming Sphericity

            SS   num Df Error SS   den Df   F   Pr(>F)
(Intercept) 63126359   1   625976   25 2521.194 < 2.2e-16 ***
factors     8112     1   17972   25   11.284   0.002508 **
factorG     4229     1   22731   25    4.650   0.040855 *
factorS:factorG 472     1   13914   25    0.848    0.365892
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```
So, what happened here is that we ran the Anova() command (found in the car package) with the model object we called ‘datmodel’ (which holds our actual RT data) and the factors for the model (idata holds our dataframe specifying the specific factors defining our model). Idesign specifies the formula for the ANOVA based on the idata factors and Type = “III” indicates the specific way of calculating the “sums of squares” for the model (you can choose between II and III, but as SPSS uses III we use this too so we can compare the outcomes from SPSS with this model to see whether they match).

TIP: this may seem very difficult, however, you can look at the help documentation for Anova() using

```
> ?Anova
```

and see what exactly you need to do to make this work for yourself.

So, what we can see that there is a main effect of semantics F(1,25) = 11.3, p < .01, and Gender F(1,25) = 4.7, p < .05. Which is exactly what we also get in SPSS if we run its Repeated Measures on this data:

<table>
<thead>
<tr>
<th>Effect</th>
<th>Value</th>
<th>F</th>
<th>Hypothesis df</th>
<th>Error df</th>
<th>Sig.</th>
<th>Partial Eta Squared</th>
</tr>
</thead>
<tbody>
<tr>
<td>sem</td>
<td>Pillai's Trace</td>
<td>.311</td>
<td>11.268&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.03</td>
</tr>
<tr>
<td></td>
<td>Wilks' Lambda</td>
<td>.689</td>
<td>11.268&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.03</td>
</tr>
<tr>
<td></td>
<td>Hotelling's Trace</td>
<td>.451</td>
<td>11.268&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.03</td>
</tr>
<tr>
<td></td>
<td>Roy's Largest Root</td>
<td>.451</td>
<td>11.268&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.03</td>
</tr>
<tr>
<td>gen</td>
<td>Pillai's Trace</td>
<td>.157</td>
<td>4.651&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.041</td>
</tr>
<tr>
<td></td>
<td>Wilks' Lambda</td>
<td>.843</td>
<td>4.651&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.041</td>
</tr>
<tr>
<td></td>
<td>Hotelling's Trace</td>
<td>.196</td>
<td>4.651&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.041</td>
</tr>
<tr>
<td></td>
<td>Roy's Largest Root</td>
<td>.196</td>
<td>4.651&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.041</td>
</tr>
<tr>
<td>sem * gen</td>
<td>Pillai's Trace</td>
<td>.033</td>
<td>.849&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.366</td>
</tr>
<tr>
<td></td>
<td>Wilks' Lambda</td>
<td>.034</td>
<td>.849&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.366</td>
</tr>
<tr>
<td></td>
<td>Hotelling's Trace</td>
<td>.034</td>
<td>.849&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.366</td>
</tr>
<tr>
<td></td>
<td>Roy's Largest Root</td>
<td>.034</td>
<td>.849&lt;sup&gt;b&lt;/sup&gt;</td>
<td>1.000</td>
<td>25.000</td>
<td>.366</td>
</tr>
</tbody>
</table>

<sup>a</sup> Design: Intercept  
<sup>b</sup> Within Subjects Design: sem * gen + sem + gen
<sup>c</sup> Exact statistic

Method 2. ezANOVA. (much easier!!! Yay?)

In this method we will use the ezANOVA() function from the ez package. So, make sure you have installed the ez package before doing this.

```
> dat = read.csv(file="GenCon26ppV2.csv", header = T)
```

In this file the set-up is already prepared. With one column specifying the semantics and another column specifying the gender factors, a participant column and the averaged RTs. So, this file looks a bit different from the other file.

```
> head(dat, n = 4)
pptn semantic gender RT
1  1  Sp  Gp 889.06
2  2  Sp  Gp 765.05
3  3  Sp  Gp 867.87
```
Now, it's actually very easy (yay, we like easy!) to run a 2x2 Repeated Measures analysis with `ezANOVA()`.

```r
> library(ez)
> ezANOVA(data = dat, dv = RT, wid = ppn, within = .(semantic, gender),
detailed = 1)
```

Warning: Converting "ppn" to factor for ANOVA.

```
$ANOVA
      Effect DFn DFd        SSn       SSd            F            p
     p<.05          ges
1  (Intercept)   1  25  63126359.359 625975.51 2521.1193819 1.262924e-26
  * 0.9893335736
2  semantic     1  25     8112.298  17971.68   11.2848323 2.508336e-03
  * 0.0117790701
3   gender     1  25     4228.665  22731.29    4.6507092 4.085471e-02
  * 0.0061748492
4 semantic:gender   1  25     472.008 13913.67    0.8481015 3.658919e-01
  0.0006930449
```

In the `ezANOVA()` command we specified a data frame to hold our data (data=dat) and a dependent variable (dv= RT) also we specified the case identifier (wid = ppn) and the two columns which specify our model factors, within = .(semantic, gender) don’t forget the dot ;-) and we want a detailed output (detailed = 1).

What have we learned?

- Run several t-tests
- Run an one-way ANOVA
- Run 2x2 Repeated measures ANOVA using two different techniques
Chapter 6 - Linear mixed effects modeling (LME)

6.1. A bit of background on F1/F2/F’/minF’ and LME.

In the early 70s there was a Stanford psychologist named Herbert Clarke who in an influential paper (Clarke, 1973) put forward that when doing only a by-subjects analysis (which was evidently normal at that time) one may overlook the fact that findings one obtained may actually be due to the specific selection of stimuli (or conditions) for the experiment. Therefore, results may not necessarily extrapolate to language in general and conclusions should be drawn carefully. Or as he put it: “those 20 words you’ve chosen may not necessarily represent all the words in the whole language you’re generalizing to”. He, luckily, proposed a solution to calculate the so-called quasi-F ratio (or F’) which in turn depends on calculating F-ratio’s for participant’s (F1) as well as and items (F2). In addition he proposed (Clarke, 1973, p.347) that if one can obtain F1 and F2 one may also calculate min F’ which is as follows: \( \text{min } F'(i,j) = \frac{F_1F_2}{F_1+F_2} \). The point of this min F’ (or F’) calculation is to generalize both over subjects and items at the same time.

However, scientists are human beings (really?) and need to feed their families too. So, one problem was that F’ or min F’ are more conservative measures than those popular before. Which means it’s harder to get a significant value and hence less papers published. Therefore, some researchers started to argue that doing only F1 and F2 was sufficient and we didn’t need the over-conservative F’ or min F’.

![Graph](image)

In this graph by Raaijmakers et al. (1999) on the left you can see the proportion of papers reporting min F’ compared to reporting F1/F2 only. So, in about 20 years the min F’ statistic was completely wiped out of psycholinguistic literature in favor of only reporting F1/F2.

Luckily, there are other techniques at our disposal such as linear mixed effects modeling (LME). For a wonderful book which deals with every aspect of R and lme I gladly refer to Harald Baayen’s (2008) book “Analyzing Linguistic Data: A Practical Introduction to Statistics using R” which should be on the shelf of anybody working with language and statistics. Anyway, to go on, what’s so cool about LME is that, like Clark suggested, now we can use both participants and items as so-called random effects. Random means that we don’t know what to expect from these factors. Every subject is different and will produce different RTs for instance, but the same holds for words, every word may show different reactions to it too. In addition to random effects, LME also includes fixed effects which are factors you are actually controlling in your experiments, such as condition, frequency etc.

6.2. Analyzing a semantic blocking experiment using what we know + LME

Okay, we will analyze a fictional dataset collected in China (this is the same dataset as used in paragraph 4.2). Twenty-four participants underwent a semantic blocking task in which they had to read
aloud (in Chinese) picture names and Chinese characters (called Hanzi) like the stimuli depicted in the two panels below (which give an overview of all the stimuli used in the experiment). In this experiment pictures or words were presented sequentially in separate picture naming and Chinese character naming blocks. In these blocks stimuli were presented one-by-one either in a homogeneous fashion (e.g. sheep, cat, rabbit, bear, horse) or a heterogeneous fashion (sheep, violin, swords, spoon, tongue).

What is usually found in this paradigm is that homogeneous groups of stimuli take longer to name due to the buildup of semantic activity/interference (called the semantic blocking effect) compared to heterogeneous blocks (see Kroll & Stewart, 1994).

As this is the last analysis of this reader I will try to combine as much as we have learned from previous chapters in doing it. Therefore, we will now analyze this dataset using both linear mixed effects (LME) and F1/F2/min F’ and include some transformations and create some graphs and do some basic “math” operations. This may end up in a not so “typical” analysis for this particular data but the purpose is to learn and practice as much as we can regarding R, so I guess it’s okay.

**TASK:**

First thing we will do is to load the *lme4, lmerTest* and the *lattice* packages in the running version of R as we will be needing them later (make sure to also *install* them if you haven’t done so already).

```r
> library(lme4)
Loading required package: Matrix
Attaching package: lme4.
> library(lmerTest)
> library(lattice)
```

And load the dataset into a dataframe

```r
> dat = read.csv(file="SemBlockingExportR.csv", header = T)
```

R automatically puts values as factors if they contain text, however if values contain numbers it sees it as a integer or floating numbers but not a factor, you can see the layout of the data frame using `str()`,

by Rinus G. Verdonschot, PhD – Waseda University (WIAS), Japan
therefore we need to convert some of the columns into factors (participants and errors); Errors were encoded by the experiment leader for each stimulus with a number ranging from 1-3 (1 = correct, 2 = Word Error, 3 = Voice Key Error).

> dat$ppn = factor(dat$ppn)
> dat$PicErr = factor(dat$PicErr)
> dat$HanziErr = factor(dat$HanziErr)
> dat$HanziPicErr = factor(dat$HanziPicErr)

Now these values are converted into factors it’s time to look at our data. For this particular paradigm it has been suggested under certain circumstances to remove the first value of each new block (Damian et al. 2001, p.B80) as that value may have a diverging reaction time as the context changes. Although we may choose to do or not to do that for this particular dataset, I will do it anyway for learning purposes.

To do that one must appreciate that in the design each separate mini-block consist of 20 trials (4 repetitions of each 5 items in a hetero- or homogeneous group) and we want to throw each first one of this 20 trial series away, and keep the remaining nineteen. So, we just add a vector (length 20 x amount of mini-blocks in total) code 1 for first and 2 for the remaining 19 and repeat that for all our data.

> dat$FirstOut = rep(c(1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2), times = 1000)

Now we just reuse the “dat” dataframe which only includes the 2’s from dat$FirstOut, thereby effectively throwing out the 1’s. This amounts to excluding 5% (1000 trials). Don’t worry about whether or not we should actually do this for this particular set, this is just to show one how you may do it if the need arises.

> dat=dat[dat$FirstOut == 2,]

Now I want to see whether there were any “weird” participants for which you can immediately see whether or not they show some strange patterns. For instance, due to getting tired/bored or perhaps malfunction of equipment.

>qqmath(~HanziPicRT|ppn, dat = dat)

This command qqmath() draws quantile-quantile plots per participant so the variable which holds RTs for both Picture and Hanzi naming called: HanziPicRT and use | (means ‘for each’) per participant (ppn) and pls don’t forget the tilde sign (~) before ~HanziPicRT.

What you’ll (hopefully) end up with is the following plot.
And you can immediately see that there is something odd about the data from participant five (s/he may be on drugs?). There are a lot of RT = 0 and a lot of RT > 1500 for this person. Although some may try to salvage his/her data I will throw the data for this person away.

```r
> dat2 = dat[dat$ppn!=5,]
```

This means, create new a new object (dat2) and use data from dat where ppn is not (!=) number 5, which actually means everything except 5. This command effectively copies dat into dat2 without including participant 5.

For educational purposes in learning more about R, we will now first check the picture naming section of this experiment by separating it out of the overall list. Let’s make a new object holding only the data from the picture naming part.

```r
> datP = dat2[dat2$Proc=="PictureProc",]
> datP2 = datP[datP$PicErr == 1,]
```

So, this makes a new object (datP) and uses data from dat2 but only where the procedure equals PictureProc (thereby not including “HanziProc”). The next command makes another object (datP2) which only holds the correctly named items from datP (PicErr == 1)

So, as you can see there are not too many “wild” participants like participant 5 anymore (although participant 3 may be his cousin). In order to trim the data for outliers we can set boundaries usually for instance 2.5 SD from a participant’s mean per condition or some hard boundaries like < 200 and > 1500. As this is a beginner’s manual I’ll do the second for now (although I prefer the first).
```r
> datP3 = datP2[datP2$PicRT > 200 & datP2$PicRT < 1500,]
```

This command makes a new object (datP3) which uses data from the data frame “datP2” we’ve just created but includes only that data which is larger than 200 and shorter than 1500ms.

We can take a peek at the average reaction times for the hetero- and homogeneous groups in the picture naming task using the following command.

```r
> with(datP3, tapply(PicRT, list(condition), mean))
  hetero homo
  578.0021 602.9754
```

Now we can do the same thing for the Chinese character part (hanzi)

```r
> datK = dat2[dat2$Proc == "HanziProc",]
> datK2 = datK[datK$HanziErr == 1,]
> qqqmath(~HanziRT|ppn, dat = datK2)
```

You can see that the Chinese characters are named faster compared to pictures (which is normal), so for now we’ll apply another trimming criterion (> 200 and < 1000).

```r
> datK3 = datK2[datK2$HanziRT > 200 & datK2$HanziRT < 1000,]
> with(datK3, tapply(HanziRT, list(condition), mean))
  hetero homo
  479.4268 486.0660
```

Just to make it clear, we apply outlier trimming like this just to learn how to do it using R. You have to consider what is best for your own data when you do your own data analyses. Many researchers do not like to trim too much and use as much data as they can.

We will now combine the picture naming and hanzi naming data together with `rbind()` (row bind)

```r
> datA = rbind(datP3, datK3)
```

Although we already had a column containing the RTs (HanziPicRT) which I made in excel just by adding
the HanziRT and PicRT columns together in excel let’s do that in R again. In excel, you can do that by simply adding the HanziRT and PicRT columns together.. but if you do that in R you’ll hit a little snag.

```r
> data$RT = data$HanziRT + data$PicRT
> data$RT
[1] NA NA NA NA NA NA NA
```

Because the data comes from E-prime it created two RT columns (one for PictRT and one for HanziRT) and either column has NA values for the parts that specific procedure wasn’t run (so no RTs in PictRT column when HanziProc was run and vice versa). So, there are some cells which do not contain data for HanziRT and PicRT (those are the cells denoted as NA). So, let’s get rid of the NA’s before adding the two columns together, and add the two columns together after we put the NA’s to 0.

```r
> data$HanziRT[is.na(data$HanziRT)] <- 0
> data$PicRT[is.na(data$PicRT)] <- 0
> data$RT = data$HanziRT + data$PicRT
```

What this does is for data$HanziRT or data$PicRT the cells containing NA (is.na) put 0 in them and combine these two HanziRT and PicRT to form a new column called RT containing all RT data. This column should be equal to the HanziPicRT column, which it is: (yay!)

```r
> sum(data$RT - data$HanziPicRT)
[1] 0
```

Okay, let’s practice some more and plot the means into a nice bar chart we can use to impress our boss (and get a nice raise ^_^). We know from Chapter 4 how to do this (hopefully).

```r
> myMeans = with(data,tapply(RT,list(condition, Proc),mean))
> barplot(myMeans, beside=T, main = "Averaged RT per condition per task", ylab = "mean RT", ylim=c(400,700), xpd=F)
> legend(1.1, 670, c("Hanzi", "Pictures"), fill=c("black", "grey"))
> box()
```

So, first we create a matrix called “myMeans” which holds the means of the RTs, then use tapply() for both hetero- and homogeneous conditions (condition) for Picture and Character naming procedures (proc).

Then we use the barplot() command to plot that matrix (beside =T means plot the conditions next to each other instead of on top of each other), we give it a title (main=) and a y-label (ylab) and put some limits on the y-axis (400, 700) and make sure the bars do not cross the boundaries (with xpd = F).

Lastly, we place a legend (x, y coordinates) containing two labels (hanzi, pictures) and fill it with black and grey color, and of course lastly, like its Christmas, put everything in a box using box().

If everything went well we should see a nice chart such as
Now, we will plot the RT distributions for both Character naming and picture naming to put in a paper to show how the distributions we’ll do our LME on look like.

```r
> plot(density(sort(datA$RT[datA$Proc == "HanziProc"])), type = "l", col = "blue", main = "Density Plot", xlab = "RT", ylab = "frequency")
> lines(density(sort(datA$RT[datA$Proc == "PictureProc"])), type = "l", col="red", lty=2)
> legend(850,0.005,c("Hanzi", "Pictures"), col = c("blue", "red"), lty = c(1,2))
```

I already explained this procedure in chapter 4.2 so please refer to that if you’re unsure how to construct these commands (also you can use `?plot`, `?lines` and `?legend` of course in R to find out more about those commands).

In any case, if everything went fine, you should have ended up with a plot like this. In this plot we can see the distributions for the naming latencies for both character naming (Hanzi) and picture naming.

Ok, time to run the LME analysis. This is not too difficult as we only have two factors with two levels.

```r
> datA.lmer = lmer(RT~condition*Proc+(1|ppn)+(1|item), dat = datA)
```

So what this does is to create a new object called `datA.lmer` the linear mixed effects analysis we’ll run is denoted by the function `lmer()` the dependent variable is RT which depends (denoted by `~`) on two fixed or independent variables, namely: condition and Proc. The asterisk (*) between means that the interaction between them is also taken into account. If you’re not interested in the interaction just use the + sign. Lastly, we add the two random factors participant (1|ppn) and items (1|item). You’ll have to get used to the somewhat awkward notation of (1|<something>) to denote random factors. Now after we run this we’ll look at the results.
As we can see from this model is that 17203 data points (nr of obs) were used in the analysis. There was a main effect of condition (homogeneous blocks were named slower than heterogeneous blocks) and also of procedure (pictures were named slower than characters) and also an interaction between the two. Also, sometimes you want to see whether you should run a model with or without the interaction. You can see how much a model explains by looking at the Akaike Information Criterion (AIC) which is informative of how well a model fits a certain set of data (the higher the number the more informative).

But wait, where are my P-values?? (this is usually the first, and often only, thing people look at).... omg.. my P-valueeeeeeees???!!!!!

It turns out that in the current version of lme() it's not clear for some complicated reasons (such as the determination of the appropriate degrees of freedom) to calculate p-values. To alleviate this, for earlier versions of R combined with lme, Prof. Harald Baayen has written the languageR package in which makes use of so-called Markov Chain Monte Carlo sampling (MCMC sampling) to estimate p-values for our design, using the pvals.fnc() function. But, if you try this now, the following appears!!

> pvals.fnc(dataA.lmer)
Error: could not find function "pvals.fnc"

OMG, no more monte carlo sampling.. luckily, another scholar named Alexandra Kuznetsova and her colleagues from the magic country of Denmark have written an alternative package using the Satterthwaite (Satter-what?!?) approximation...


So, we can still get our p-values.. simply by doing summary()..(make sure lmerTest is loaded)

> summary(dataA.lmer)
Linear mixed model fit by REML t-tests use Satterthwaite approximations to degrees of freedom [merModLmerTest]
Formula: RT ~ condition * Proc + (1 | ppn) + (1 | item)
Data: datA

REML criterion at convergence: 202070.6

Scaled residuals:

Min  1Q  Median  3Q  Max
-4.6522 -0.5750 -0.1086 0.4137 9.8551

Random effects:

Groups   Name        Variance Std.Dev.
item     (Intercept) 1011     31.80
ppn      (Intercept) 3185     56.44
Residual             7298     85.43
Number of obs: 17203, groups: item, 25; ppn, 23

Fixed effects:

Estimate Std. Error        df t value Pr(>|t|)
(Intercept)                     478.953     13.440    34.000  35.637  < 2e-16 
***
conditionhomo                     7.732      1.839 17165.000   4.204 2.64e-05  
***
ProcPictureProc                  98.900      1.839 17164.000  53.786 < 2e-16 
***
conditionhomo:ProcPictureProc    17.470      2.605 17164.000   6.705 2.07e-11
***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:

                   (Intr) conditionhomo ProcPictureProc
conditionhomo -0.068
ProcPictureProc -0.068  0.497
conditionhomo:ProcPictureProc  0.048 -0.705 -0.706

> anova(data.lmer)

So, for this analysis, one might report something like this for the condition factor (hetero- vs. homogeneous groups) \( \text{“} t(17203) = 4.204, SE=1.839, p < .001 \text{”} \). However, there are many other factors to consider. As there is an interaction, we might want to re-think how we set-up the analysis. Additionally, there are papers out there particularly the paper by Dale Barr:

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, 68, 255–278. ([http://idiom.ucsd.edu/~rlevy/papers/barr-etal-2013-jml.pdf](http://idiom.ucsd.edu/~rlevy/papers/barr-etal-2013-jml.pdf)) which states that its best to keep the maximal random effects structure (if the design allows), that means random slopes and random intercepts such as in the following example:

> data_MAX.lmer = lmer(RT~condition*Proc+(1+condition|ppn)+(1+condition|item), dat = datA)

(1+condition|ppn) means random slopes for the participants instead of (1|ppn)

You can check the outcome again by:

> summary(data_MAX.lmer)

Linear mixed model fit by REML t-tests use Satterthwaite approximations to degrees of freedom [merModLmerTest]
Formula: RT ~ condition * Proc + (1 + condition | ppn) + (1 + condition | item)

REML criterion at convergence: 202022.7

Scaled residuals:

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>1Q</th>
<th>Median</th>
<th>3Q</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-4.7246</td>
<td>-0.5701</td>
<td>-0.1104</td>
<td>0.4133</td>
<td>9.8737</td>
</tr>
</tbody>
</table>

Random effects:

<table>
<thead>
<tr>
<th>Groups</th>
<th>Name</th>
<th>Variance</th>
<th>Std.Dev.</th>
<th>Corr</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>(Intercept)</td>
<td>1046.25</td>
<td>32.346</td>
<td></td>
</tr>
<tr>
<td></td>
<td>conditionhomo</td>
<td>121.65</td>
<td>11.029</td>
<td>-0.19</td>
</tr>
<tr>
<td>ppn</td>
<td>(Intercept)</td>
<td>3008.44</td>
<td>54.849</td>
<td></td>
</tr>
<tr>
<td></td>
<td>conditionhomo</td>
<td>46.54</td>
<td>6.822</td>
<td>0.44</td>
</tr>
<tr>
<td>Residual</td>
<td></td>
<td>7258.00</td>
<td>85.194</td>
<td></td>
</tr>
</tbody>
</table>

Number of obs: 17203, groups: item, 25; ppn, 23

Fixed effects:

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>Std. Error</th>
<th>df</th>
<th>t value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>478.719</td>
<td>13.203</td>
<td>35.000</td>
<td>36.257</td>
</tr>
<tr>
<td>&lt; 2e-16 ***</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conditionhomo</td>
<td>7.923</td>
<td>3.203</td>
<td>43.000</td>
<td>2.474</td>
</tr>
<tr>
<td>0.0174 *</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProcPictureProc</td>
<td>98.902</td>
<td>1.834</td>
<td>17124.000</td>
<td>53.935</td>
</tr>
<tr>
<td>&lt; 2e-16 ***</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conditionhomo:ProcPictureProc</td>
<td>17.460</td>
<td>2.598</td>
<td>17125.000</td>
<td>6.720</td>
</tr>
<tr>
<td>1.88e-11 ***</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signif. codes:</td>
<td>0 '<em><strong>' 0.001 '</strong>' 0.01 '</em>' 0.05 '.' 0.1 ' ' 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Correlation of Fixed Effects:

<table>
<thead>
<tr>
<th></th>
<th>(Intr)</th>
<th>conditionhomo</th>
<th>ProcPictureProc</th>
</tr>
</thead>
<tbody>
<tr>
<td>conditionhomo</td>
<td>0.065</td>
<td>-0.069</td>
<td>0.285</td>
</tr>
<tr>
<td>ProcPictureProc</td>
<td>-0.069</td>
<td>0.285</td>
<td>-0.404</td>
</tr>
<tr>
<td>conditionhomo:ProcPictureProc</td>
<td>0.049</td>
<td>-0.404</td>
<td>-0.706</td>
</tr>
</tbody>
</table>

As this is an introductory reader I refer to specialist papers such as the ones mentioned above to find out what is best for your own data. This reader is just there to make you a bit familiar with R commands..

TIP: you can also do anova() using lmertest which gives you the same information (from the class) but with additional F statistics.

> anova(dataA_MAX.lmer)

However, lets forget about LME for now, to make things extra fun and see whether we get comparable results from F1/F2 analysis let’s do that too (it’s good for practicing R!!).

> attach(dataA)
> AggDatAsubj = aggregate(dataA$RT, by=list(ppn, Proc, condition), FUN = mean)
> detach(dataA)
> library(ez)
We first attach the datA dataframe we also used for the LME analysis then we aggregate the reaction times for each participant using the \texttt{aggregate()} function, for picture or character naming and for condition (hetero- vs. homogeneous groups) separately so they are in the correct format for ezANOVA (see also Method 2 of Paragraph 5.3.). The \texttt{ezANOVA()} function is contained in the “ez” package which we need to load, using \texttt{library(ez)}.

\begin{verbatim}
> ezANOVA(data=AggDatAsubj, dv=(x), wid=(Group.1), within=(Group.2, Group.3), type=3, detailed=T)
Warning: You have removed one or more Ss from the analysis. Refactoring "Group.1" for ANOVA.

\textbf{$\text{ANOVA}$}

\begin{tabular}{cccccc}
\hline
\textbf{Effect} & \textbf{DFn} & \textbf{DFd} & \textbf{SSn} & \textbf{SSd} & \textbf{F} & \textbf{p} \\
\hline
(Intercept) & 1 & 22 & 26486465.674 & 280639.259 & 2076.33902 & 2.843869e-23 \\
* & 0.988216999 & & & & & \\
Group.2 & 1 & 22 & 265340.422 & 31600.977 & 184.72496 & 3.510953e-12 \\
* & 0.456576876 & & & & & \\
Group.3 & 1 & 22 & 5700.799 & 1842.333 & 68.07540 & 3.522948e-08 \\
* & 0.017731214 & & & & & \\
Group.2:Group.3 & 1 & 22 & 1797.790 & 1728.696 & 22.87932 & 8.916677e-05 \\
* & 0.005660386 & & & & & \\
\hline
\end{tabular}

\textbf{So, we now have ran the ezANOVA for the participants analysis and show indeed that results are comparable to the LME analysis so the condition (Group.2) and Proc (Group.3) and the condition x Proc interaction are all significant. What is cool is the fact that ezANOVA detects that we’ve removed participant 5 in the very beginning of this chapter and gives us this warning in red.}

Ok let’s do the same for items (now using item instead of ppn in the aggregate command).

\begin{verbatim}
> attach(datA)
> AggDatAitem = aggregate(datA$RT, by=list(item, Proc, condition), FUN = mean)
> detach(datA)
> ezANOVA(data=AggDatAitem, dv=(x), wid=(Group.1), within=(Group.2, Group.3), type=3, detailed=T)
\end{verbatim}

\textbf{$\text{ANOVA}$}

\begin{tabular}{cccccc}
\hline
\textbf{Effect} & \textbf{DFn} & \textbf{DFd} & \textbf{SSn} & \textbf{SSd} & \textbf{F} & \textbf{p} \\
\hline
(Intercept) & 1 & 24 & 28789721.888 & 97026.061 & 7121.31688 & 3.328612e-31 \\
* & 0.99589820 & & & & & \\
Group.2 & 1 & 24 & 290248.462 & 14902.479 & 467.43654 & 3.035680e-17 \\
* & 0.70995850 & & & & & \\
Group.3 & 1 & 24 & 7035.120 & 4089.548 & 41.28644 & 1.208435e-06 \\
* & 0.05600711 & & & & & \\
Group.2:Group.3 & 1 & 24 & 1797.790 & 1728.696 & 22.87932 & 8.916677e-05 \\
* & 0.005660386 & & & & & \\
\hline
\end{tabular}

\textbf{So, we have now both the F1 and F2 statistics. Usually this is where people stop, however, not us!}

We can now actually calculate min $F’$ ourselves.

Let’s say for the factor condition (hetero- vs. homogeneous groups) we have the F value 68.1 for F1 and 41.3 for F2, the min $F’$ value is according to Clark (1973) described by the formula $\rightarrow$
(15) \( \min F'(i,j) = \frac{F_1 F_2}{(F_1 + F_2)} \) (taken from Clark, 1973; p.347).

Therefore our \( \min F' \) is:

\[
> \text{datAminF} = 68.1 * 41.3 / (68.1 + 41.3)
\]

Degrees of freedom for the numerator is the same as the one for \( F_1 \) so 1 in our case, however the degrees of freedom for the denominator is a bit tricky..

\[
(16) j = \left( \frac{F_1 + F_2}{n_2} + \frac{F_2^2}{n_1} \right)
\]

(taken from Clark, 1973; p.347)

Let's do it anyway.. but you don't really need to know how to do it. this is just for practicing how to do a bit of math and operators in R (like in Chapter 2). Oh, btw. \texttt{round()} is the round off function.

\[
> \text{round}((68.1 + 41.3)^2 / ((68.1^2)/24 + (41.3^2)/22), 0)
\]

Now again, we have no p-value, but luckily they can be looked up via the \texttt{pf()} command, you only need a \( F \) value and the numerator and denominator for the df's (which we all calculated now).

\[
> 1-pf(25.7, 1, 44)
\]

So, \( \min F'(1,44) = 25.7, p < .001 \)

Very significant even when generalizing via \( \min F' \);-)

Lastly, let's do some pairwise comparisons using paired t-tests (to practice what we learned in Chapter 5).

\[
> \text{t.test(\texttt{AggDatAsubj$x[AggDatAsubj$Group.3=="homo" & AggDatAsubj$Group.2=="HanziProc"], AggDatAsubj$x[AggDatAsubj$Group.3=="hetero" & AggDatAsubj$Group.2=="HanziProc"], paired = T})}
\]

Paired t-test

data:  \texttt{AggDatAsubj$x[AggDatAsubj$Group.3 == "homo" & AggDatAsubj$Group.2 == "HanziProc"] and AggDatAsubj$x[AggDatAsubj$Group.3 == "hetero" & AggDatAsubj$Group.2 == "HanziProc"]}
t = 2.5356, df = 22, p-value = 0.01884 alternative hypothesis: true difference in means is not equal to 0 95 percent confidence interval:
1.256999 12.548026 sample estimates:
mean of the differences
6.902512
```r
> t.test(AggDatAitem$x[AggDatAitem$Group.3=="homo" & AggDatAitem$Group.2=="HanziProc"],
AggDatAitem$x[AggDatAitem$Group.3=="hetero" & AggDatAitem$Group.2=="HanziProc"], paired = T)

Paired t-test
data:  AggDatAitem$x[AggDatAitem$Group.3 == "homo" & AggDatAitem$Group.2 == "hetero" & AggDatAitem$Group.2 == "HanziProc"] and  "HanziProc"
t = 4.291, df = 24, p-value = 0.000252 alternative hypothesis: true difference in means is not equal to 0 95 percent confidence interval:
3.974286 11.340359 sample estimates:
mean of the differences 7.657323
```

So, it seems that the 7ms effect in hanzi naming condition between hetero- and homogeneous groups for both the t1 and t2 analyses are significant.

```r
> t.test(AggDatAsubj$x[AggDatAsubj$Group.3=="homo" & AggDatAsubj$Group.2=="PictureProc"],
AggDatAsubj$x[AggDatAitem$Group.3=="hetero" & AggDatAsubj$Group.2=="PictureProc"], paired = T)

Paired t-test
data:  AggDatAsubj$x[AggDatAsubj$Group.3 == "homo" & AggDatAsubj$Group.2 == "hetero" & AggDatAsubj$Group.2 == "PictureProc"] and  "PictureProc"
t = 9.4948, df = 22, p-value = 3.074e-09 alternative hypothesis: true difference in means is not equal to 0 95 percent confidence interval:
19.21486 29.95450 sample estimates:
mean of the differences 24.58468
```

(it may give you a warning message about length)

```r
> t.test(AggDatAitem$x[AggDatAitem$Group.3=="homo" & AggDatAitem$Group.2=="PictureProc"],
AggDatAitem$x[AggDatAitem$Group.3=="hetero" & AggDatAitem$Group.2=="PictureProc"], paired = T)

Paired t-test
data:  AggDatAitem$x[AggDatAitem$Group.3 == "homo" & AggDatAitem$Group.2 == "hetero" & AggDatAitem$Group.2 == "PictureProc"] and  "PictureProc"
t = 5.9443, df = 24, p-value = 3.908e-06 alternative hypothesis: true difference in means is not equal to 0 95 percent confidence interval:
16.90275 34.88310 sample estimates:
mean of the differences 25.89293
```

So, again it seems that the 25ms effect in picture naming condition between hetero- and homogeneous...
groups for both the $t_1$ and $t_2$ analyses are significant.

Now, we practiced most of the things we learned in Chapters 1-5 plus the LME analysis described in this Chapter. Hopefully, this last chapter will give a good idea of how to do some elementary stuff in R. Of course doing a LME analysis requires more thinking and analyses, such as: do we want random slopes? or what to do with the interaction etc. I recommend to read Chapter seven called “mixed models” of Baayen’s (2008) book for a complete overview of everything possible using LME and linguistic data.

In this chapter we learned

- How to load data
- How to preprocess data
- How to run a LME and ImerTest/Satterthwaite-approximation via `summary()`
- How to run a F1/F2 analysis using `ezANOVA()`
- How to manually calculate min $F'$
- How to run pairwise comparisons (paired t-tests)
Appendix - Index

Anova(), 39
aov, 37
attach, 39
barplot, 26
barplot(), 44
box(), 26
cbind, 14
class, 14
conflicts, 10
console, 6
Data Frames, 15
edit, 15
ezANOVA(), 39
file.choose(), 22
fixed effects, 43
getwd(), 19
help, 17
lawstat, 34
library, 7
linear mixed effects modeling, 5
lines, 27
lm(), 39
lmer(), 44
logical commands, 11
Matrices, 14
mean, 12
merge, 23
nrow, 27
packages, 5
Packages, 5
paired = T, 35
pf(), 44
plyr, 23
pvals.fnc(), 44
qqmath(), 44
random effects, 43
rbind, 14
rbind(), 44
read.csv, 20
read.table, 20
rm, 10
round(), 44
seq, 12
setwd(), 19
str, 15
sum, 12
tail, 15
tapply(), 44
vectors, 13
by Rinus G. Verdonschot, PhD – Waseda University (WIAS), Japan