Workflow of statistical data analysis

Oliver Kirchkamp

© Oliver Kirchkamp

Workflow of empirical work may seem obvious. It is not. Small initial mistakes can lead to a lot of hard work afterwards. In this course we discuss some techniques that hopefully facilitate the organisation of your empirical work.

This handout provides a summary of the slides from the lecture. It is not supposed to replace a book.

Many examples in the text are based on the statistical software R. I urge you to try these examples on your own computer.

As an attachment of this PDF you find a file `wf.zip` with some raw data. You also find a file `wf.Rdata` with some R functions and some data already in R's internal format.

The drawing on the previous page is Albercht Dürer's "Der Hafen von Antwerpen" — an example for workflow in a medieval city.

Contents

© Oliver Kirchkamp

© Oliver Kirchkamp

1 Introduction

Literature: Surprisingly, there is not much literature about workflow of statistical data analysis:

General Literature

- J. Scott Long; The Workflow of Data Analysis Using Stata, Stata Press, 2009.

© Oliver Kirchkamp

- Hadley Wickham; Tidy Data; *Journal of Statistical Software*, 2014.
- Christopher Gandrud; Reproducible Research with R and RStudio, 2015.
- Garrett Grolemund, Hadley Wickham; R for Data Science, 2017.
- Andrew Gelman, Aki Vehtari, Daniel P. Simpson, C. Margossian, B. Carpenter, Y. Yao, Lauren Kennedy, J. Gabry, Paul-Christian Burkner, and Martin Modr'ak; Bayesian Workflow, 2020.

Literate Programming

- Friedrich Leisch; Sweave User Manual.
- Nicola Sartori; Sweave = R · LaTeX^2
- Yihui Xie; knitr - Elegant, flexible, and fast dynamic report generation with R.
- Max Kuhn; CRAN Task View: Reproducible Research.

Version control

- Scott Chacon, Ben Straub; Pro Git.
- Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato; Version Control with Subversion.

R

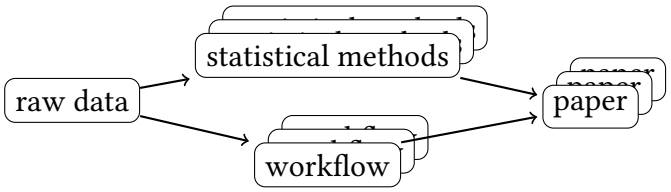
- Hadley Wickham; Advanced R.

- Introduction
  - Aims of statistical data analysis
  - Why worry about workflow?
- Keeping things together: literate programming.
- Documentation: version control (GIT)
- Organising work
  - Programming techniques
  - Repetition
  - Robustness
  - Data manipulation
- Working with data
  - Reading data
  - Cleaning data
  - Organising data
  - Descriptive statistics
  - Specific results

© Oliver Kirchkamp

1.1 Motivation

What is empirical research?



- We spend a lot of time explaining statistical methods to students.
- We do not tell students how to apply statistical methods, how to organise their data, how to organise their work…
- Why?
- Is "workflow" obvious? — I do not think so.

  Is the wrong workflow not costly? — On the contrary.
  - Mistakes in the statistical method can always be cured.
  - Mistakes in the workflow…

    (e.g. loss of data, loss of understanding the data, loss of methods applied)

    …can render the entire project invalid — no cure possible
- Isn't it sufficient to simply store and backup everything?
  - unfortunately not
    * statistical analysis tends to create a lot of data.

    → *storing everything* means *hiding everything* very well from us and from others.

Workflow and reproducibility Definition: Reproducible
Claerbout, Jon F, and Martin Karrenbach. 1992. 'Electronic Documents Give Reproducible Research a New Meaning'. In *SEG Technical Program Expanded Abstracts 1992*, 601–4. Society of Exploration Geophysicists:

- *Reproducible* ↔ with the same data and the same routines, we obtain the same result.

Definition 2: Reproducibility:
"…the data and code used to make a finding are *available* and they are *sufficient* for an independent researcher to recreate the finding." [emphasis added] Peng, R. D. (2011). Reproducible research in computational science. Science, 334:1226–1227.

Why do we want reproducibility?

© Oliver Kirchkamp

- It helps if we can reproduce our own work!
- Reproducability → structure → managing (multiple) projects is much easier when each project has a clear structure.
- Collaborators: It helps our coauthors if they can reproduce our work.
- Other scientists: If they can't reproduce our work it does not help them.

  The more of our work we sell to the world, the larger our impact.

  (Showing others what we did and how we did it is *good*!)

1.2 Is workflow obvious?

Silberzahn, R., E. L. Uhlmann, D. P. Martin, P. Anselmi, F. Aust, E. Awtrey, Š. Bahník, et al. 2018. 'Many Analysts, One Data Set: Making Transparent How Variations in Analytic Choices Affect Results'. *Advances in Methods and Practices in Psychological Science* 1 (3): 337–56:

- Data about 146028 dyads of soccer players and referees
- Research question: "Are soccer referees more likely to give red cards to dark-skin-toned players than to light-skin-toned players?"
- 29 teams of researchers.

↓ 29 different decisions on type of the model, treatment of dependent observations, control variables, etc.

- 29 different answers
- 11% *fewer* to 193% *more* yellow and red cards for dark-skin-toned players.

Innocent details of the statistical analysis → substantial influence on results.

→ Documentation

Errors in the data / errors introduced by the researcher? Albert J. Menkveld, Anna Dreber, Felix Holzmeister, Juergen Huber, Magnus Johannesson, Michael Kirchler, Sebastian Neusüss, Michael Razen, Utz Weitzel, et. al. 2021. "Non-Standard Errors". CESifo Working Papers 9453-2021.

- 164 research teams, 34 external peer evaluators (so the study can also measure the effect of feedback)

→ Error introduced by research teams is similar in magnitude to error in data.

© Oliver Kirchkamp

1.3 Consistency

Veldkamp, Coosje L. S., Michèle B. Nuijten, Linda Dominguez-Alvarez, Marcel A. L. M. van Assen, and Jelte M. Wicherts. 2014. 'Statistical Reporting Errors and Collaboration on Statistical Analyses in Psychological Science'. *PLoS ONE* 9 (12):

- 430 articles from six top psychology journals.
- "63% of the articles contained at least one p-value that was inconsistent with the reported test statistic."

Nuijten, Michèle B., Chris H. J. Hartgerink, Marcel A. L. M. van Assen, Sacha Epskamp, and Jelte M. Wicherts. 2015. 'The Prevalence of Statistical Reporting Errors in Psychology (1985–2013)'. *Behavior Research Methods* 48: 1205–26:

- 16695 articles with null-hypotheses significance tests (NHST) from 8 major psychology journals.
- "Across all journals and years 49.6% of the articles with NHST results contained at least one inconsistency."
- Hermans, Felienne, and Emerson Murphy-Hill. 2015. 'Enron's Spreadsheets and Related Emails: A Dataset and Analysis'. In *2015 Ieee/Acm 37th Ieee International Conference on Software Engineering*, 2:7–16:
  - 9120 Excel spreadsheet files containing formulae.
  - 24% of the analysed spreadsheets contain at least one Excel error.
- Powell, Stephen G., Kenneth R. Baker, and Barry Lawson. 2008. 'A Critical Review of the Literature on Spreadsheet Errors'. *Decision Support Systems* 46 (1): 128–38:
  - More on spreadsheet errors.

Chang, Andrew C., and Phillip Li. 2021. 'Is Economics Research Replicable? Sixty Published Papers from Thirteen Journals Say "Often Not" '. *Critical Finance Review* 10:

- Attempt to reproduce the analysis of publications from 13 economics journals. They use the data and code provided by the authors of the original papers.
- Only 33% of the papers could be reproduced without contacting the original authors.
- After contacting the authors, this percentage rose to 49%.

Wicherts, Jelte M., Denny Borsboom, Judith Kats, and Dylan Molenaar. 2006. 'The Poor Availability of Psychological Research Data for Reanalysis'. *American Psychologist* 61 (7): 726–28:

- Attempt to obtain data reported in 141 articles published by the American Psychological Association.
- For 73% of the articles the data could not be obtained.

© Oliver Kirchkamp

1.4 Replicability

Definition: Replicability
↔ Obtain similar results, based on *new* data.

Open Science Collaboration. 2015. 'Estimating the Reproducibility of Psychological Science'. *Science* 349 (6251):

- Attempt to replicate 100 studies taken from three psychology journals.
- Only 47% of effect sizes from the original study were in the 95% confidence interval of the replication effect size.

Camerer, Colin F., Anna Dreber, Eskil Forsell, Teck-Hua Ho, Jürgen Huber, Magnus Johannesson, Michael Kirchler, et al. 2018. 'Evaluating the Replicability of Social Science Experiments in Nature and Science Between 2010 and 2015.' *Nature Human Behaviour* 2 (9): 637–44:

- Attempt to replicate 18 studies from two economics journals.
- Only for 66.7% of their replications the effect size from the original study was in the the 95% confidence interval of the replication effect size.

John, Leslie K., George Loewenstein, and Drazen Prelec. 2012. 'Measuring the Prevalence of Questionable Research Practices with Incentives for Truth Telling'. *Psychological Science* 23 (5): 524–32:

- Researchers might engage in questionable research practices
  - Selective reporting
  - ⋮
  - Falsifying data

Munafò, M., Brian A. Nosek, D. Bishop, K. Button, C. Chambers, N. P. D. Sert, U. Simonsohn, E. Wagenmakers, J. Ware, and J. Ioannidis. 2017. 'A Manifesto for Reproducible Science'. *Nature Human Behaviour* 1:

- Scientists are easily misled to see structure in randomness.
- ⋮
- Data and testing
- ⋮

© Oliver Kirchkamp

1.5 Structure of a paper

- Describe the research question

  Which *economic model* do we use to structure this question?

  Which *statistical model* do we use for inference? (Estimation, hypothesis testing, classification…)

- Describe the economic method (experiment, field data,…)
- Describe the sample

  How many observations, means, distributions of main variables, key statistics

  Is there enough variance in the independent variables to test what we want to test?

- Statistical inference (estimate, test hypotheses, classify,…)

  possibly different variants of the model (increasing complexity)

- Discuss the model, robustness checks

1.6 Aims of statistical data analysis

- Limit work and time
- Get interesting results
- Reproducability
  - for us, to understand our data and our methods after we get back to work after a break
  - for our friends (coauthors), so that they can understand what we are doing
  - for our enemies — we should always (even years after) be able to *prove* our results *exactly*
- If statistical analysis was a straightforward procedure, then there would be no problem:
  - Store the raw data. All methods we apply are obvious and trivial.
- In the real world our methods are far from obvious:
  - We think quite a lot about details of our statistical analysis
- Assume we have another look at our paper (and our analysis) after a break of 6 months:
  - What does it mean if `sex==1` ?
  - For the variable `meanContribution`: was the mean taken with respect to all players and the same period, or with respect to the same player and all periods, or …
  - What is the difference between `payoff` and `payoff2`…

© Oliver Kirchkamp

  - Do the tables and figures in version 27 of the paper …
    * …refer to all periods of the experiment or only to the last 6 periods?
    * …do they include data from the two pilot experiments we ran?
    * …do they refer to the "cleaned" dataset, or to the "cleaned dataset in long form" (where we eliminated a few outliers)
    * Do all tables and figures and p-values and t-tests… actually refer to the *same* data? (or do some include outliers, some not,…)

Assume we take only 10 not completely obvious decisions between two alternatives during our analysis (which perhaps took us 1 week),…

> (coding of data, data to include, treatments to compare, lags to include, outliers to remove, interaction terms to include, types of model comparison, dealing with non-linearities, correlation structure of error terms,…)

…⟶ we will have to explore $2^{10} = 1024$ variants of our analysis (= 1024 weeks) to recover what we actually did.

Often we take more than 10 not completely obvious decisions.

⟶ we should follow a workflow that facilitates reproducibility.

What it the optimal workflow? The optimal workflow is different for each of us

Aims

- Exactness (allow clear replication)
- Efficiency
- We must like it (otherwise we don't do it)
- Whatever we do, we should do it in a systematic way
  - Follow a routine in our work (all projects should follow similar conventions)
  - Let the computer follow a routine (a mistake made in a routine will show up "routinely", a hand coded mistake is harder to detect).

    Use functions, try to make them as general as possbible.
  - Prepare for the unexpected! We should not assume that our data will always look the way it seems to look at the moment.

More on routines Example:

- Probability to make a mistake: 0.1
- Probability to discover (and fix) a mistake: 0.8

Now we solve two related problems, A and B:

© Oliver Kirchkamp

- Both problems are solved independently:
  - Probability of (undiscovered) mistake A: $0.1 \cdot 0.2$
  - Probability of (undiscovered) mistake B: $0.1 \cdot 0.2$
  - Probability of some undiscovered mistake: $1 - .98^2 \approx 0.04$
- Both problems are solved with the same routine (one function in our code):
  - Probability of some undiscovered mistake: $0.1 \cdot 0.2 \cdot 0.2 = 0.004$

Producing our results with the help of identical (and computerised) routines makes it much easier to discover mistakes.

1.7 Making the analysis reproducible

Here are again the steps in writing a paper:

1. organise raw data
2. descriptive analysis (figures, descriptive tables…)
3. develop methods for analysis
4. get results (run program code)
5. write paper (mix results with text and explanations)
6. interact with collaborators

- All these tasks require decisions.
- All these decisions should be documented.
- When is our documentation sufficient? — If a third person, without our help, can find out what we were doing in all the above steps. If we want to have another look at our data in one year's time we will be in the same position as an outsider today.
- We keep a log where we document the above steps for a given project on a daily basis (research log) (nobody wants to keep logs, so this must be easy)

1.8 Interaction with coauthors

- Clear division of labour
  - the "experimenter" decides how the experiment is actually run
  - the "empiricist" decides what statistics and graphs are produced
  - the "writer" decides how to present the text
  - help, do not interfere

© Oliver Kirchkamp

- In our communication: concentrate on the essentials:
  - exchange one file
  - make only essential changes to this file
  - clearly explain why these changes are necessary

2 Weaving and tangling

- Describe the research question.
  Which model do we use to structure this question?
  Which hypotheses do we want to test?
- Describe the method.
- Describe the sample.
  How many observations, means, distributions of main variables, key statistics?
  Is there enough variance in the independent variables to test what you want to test?
- Test hypotheses based on the model.
  Possibly different variants of the model (increasing complexity).
- Discuss model, robustness checks

2.1 How can we link paper and results?

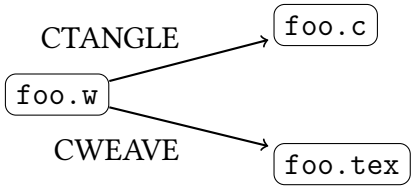
Lots of notes in the paper, e.g. the following:
In your LaTeX-file...:

```
%
% the following table was created by tableAvgProfits()
%                              from projectXYZ_160621.R
% \begin{table}
% ...
```

Better: Weave (Sweave, knitr)

2.2 A history of literate programming

Knuth, Donald E. 1992. *Literate Programming*. Vol. 26. CSLI Lecture Notes. CSLI Publications. Stanford University.

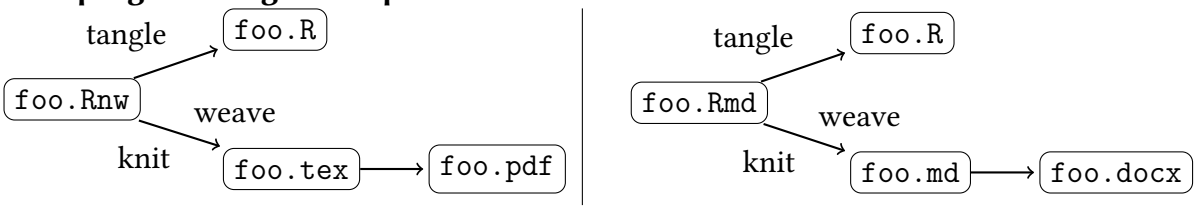


© Oliver Kirchkamp

What is "literate programming":

- meaningful and readable high-quality documentation
- details are usually not included in `#comments`
- supposed to be *read*
- facilitates feedback and reuse of code
- *reduces* the amount of text one must read to understand the code

Literate programming for empiricists:



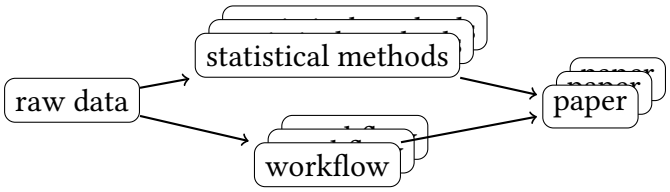
- tangle (Stangle, knit(..., tangle=TRUE)): `foo.Rnw` ⟶ `foo.R`
- weave (Sweave, knit): `foo.Rnw` ⟶ `foo.tex`
  (may contain parts of `foo.R`)

What does `Rnw` mean:

- `R` for the R project
- `nw` for noweb (*w*eb for *n*o particular language, or *N*orman Ramsey's *W*eb)

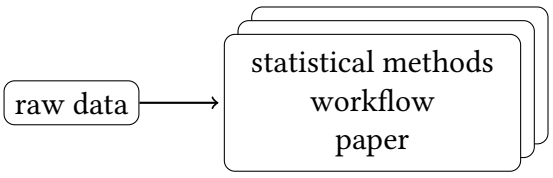
Nonliterate versus literate work

Nonliterate:



Remember: it is easy to confuse the different version of the analysis and their relation to the versions of the paper.

Literate:



With literatate programming in the analysis we avoid one relevant source of errors: Confusion about which parts of our work do belong together and which do not.

© Oliver Kirchkamp

Advantages of literate programming

- *Connection* of methods to paper (no more: 'which version of the methods were used for which figure, which table')
- The paper is *dynamic*
  - More raw data arrives: the new version of the paper writes itself
  - You organise and clean the data differently: the new version of the paper writes itself
  - You change a detail of the method which has implications for the rest of the paper: the new version of the paper writes itself

Don't write:

```
We ran 12 sessions with 120 participants.
```

instead:

```
numSession <- length(unique(sessionID))
numPart <- length(unique(partID))
⋮
We ran \Sexpr{numSession} sessions with \Sexpr{numPart} participants.
```

2.3 An Rnw document with LaTeX

Here is a brief `Rnw`-document:

```
\documentclass{article}
\begin{document}
    text that explains what you are doing and why it is
                                          interesting ...
```

```
<<someCalculations,results='asis',echo=FALSE>>=
library(Ecdat)
library(xtable)
library(lattice)
data(Caschool)
attach(Caschool)
est <- lm(testscr ~  avginc)
xtable(anova(est))
@
```

```
<<aFigure,echo=FALSE,fig.width=4,fig.height=3>>=
xyplot(testscr ~ avginc,xlab="average income",ylab="testscore",
    type=c("p","r","smooth"))
@
```

© Oliver Kirchkamp

```
the correlation between average income and testscore is
\Sexpr{round(cor(testscr,avginc),4)}.
more text …
\end{document}
```

To compile this Rnw-file, we can do the following:

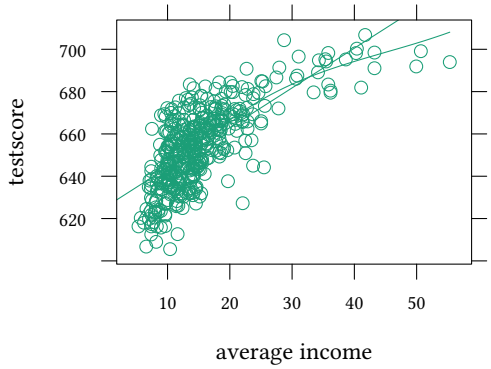
```
library(knitr)
knit("filename.Rnw")
system("pdflatex filename.tex")
```

…or use a front end like RStudio.

The result, after knitting:

text that explains what you are doing and why it is interesting …

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
avginc	1	77204.39	77204.39	430.83	0.0000
Residuals	418	74905.20	179.20		



the correlation between average income and testscores is 0.7124.
more text …

2.4 Why we use markup languages?

Visual representation

1. Introduction

…

1.1 Literature

…[12]

[12] We should add that this is more complicated…

- We describe how the text should look like.
- But we are no typographers.
- If the logic of our text changes, it is hard to adjust all visuals.

© Oliver Kirchkamp

Logical markup

```
\section{Introduction}
...
\subsection{Literature}
...
\footnote{We should add that this is more complicated...}
```

- We describe the logic.
- The visual appearance is done according to professional standards.
- If the logic of the text changes, adjustments are easy.

2.5 An Rmd document with Markdown

If we use Markdown to write our text (instead of LaTeX), the above example would look at follows:

````
text that explains what you are doing and why it is
                                       interesting ...
```{r someCalculations,results='asis',echo=FALSE}
library(Ecdat)
library(xtable)
library(lattice)
data(Caschool)
attach(Caschool)
est <- lm(testscr ~  avginc)
xtable(anova(est))
```
```{r aFigure,echo=FALSE,fig.width=4,fig.height=3}
xyplot(testscr ~ avginc,xlab="average income",ylab="testscore",
     type=c("p","r","smooth"))
```

the correlation between average income and testscore is
`r round(cor(testscr,avginc),4)`.

 more text...
```
````

2.6 Text chunks

What we saw:
````

© Oliver Kirchkamp

- The usual LaTeX(or Markdown) text
- Rnw chunks:

```
<<>>=
lm(testscr ~  avginc)
@
```

  Rmd chunks:

```
```{r}
lm(testscr ~ avginc)
```
```

  or “chunks” with parameters:

```
<<fig.height=2.5>>=
plot(est,which=1)
@
```

```
```{r fig.height=2.5}
plot(est,which=1)
```
```

  more generally

```
<<...parameters...>>=
...R-commands...
@
```

```
```{r ...parameters...}
...R-commands...
```
```

What are these parameters:

- `<<anyName,...>>=`

  not necessary, but identifies the chunk. Also helps *recycling* chunks, e.g. a figure.

```
<<anotherName,...>>=
<<anyName>>
@
```

- `<<...,eval=FALSE,...>>=`

  this chunk will not be evaluated (too time consuming...)

- `<<...,echo=FALSE,...>>=`

  the code of this chunk will not be shown

- `<<...,fig.width=3,fig.height=3,...>>=`

  All figures produced in this chunk will have this width and height.

- `<<...,results='asis',...>>=`

  The chunk produces LaTeX-output which should be inserted here ‘as is’.

Furthermore you can include small parts of output in the text:
`\Sexpr{...}`

© Oliver Kirchkamp

Elements of a knitr-document

```
\documentclass{article}
\begin{document}
<<>>=
opts_chunk[["set"]](dev='tikz', external=FALSE, fig.width=4.5,
                    fig.height=3, echo=TRUE, warning=TRUE,
                    error=TRUE, message=TRUE,
                    cache=TRUE, autodep=TRUE,
                    size="footnotesize")
@
\usepackage{tikz}
```

- `dev='tikz',external=FALSE` sets the format for plots

  (This requires package `tikzDevice`).

- `fig.width=4.5,fig.height=3` controls the the size for plots.

- `echo=TRUE, warning=TRUE, error=TRUE, message=TRUE` control what kind of output is shown.

- `cache=TRUE, autodep=TRUE` do calculate chunks only when they have changed.

- `size="footnotesize"` size of the output.

All these values can be overridden for specific knitr chunks.

Words of caution

There is still something that might break:
In case something in R changes in the future, better put somewhere in your document:

```
This document has been generated on \today, with
\Sexpr{version$version.string}, on
\Sexpr{version$platform}.
```

This document has been generated on December 16, 2021, with R version 4.1.0 (2021-05-18), on `x86_64-pc-linux-gnu`.
To reveal information about attached packages, use `sessionInfo()`:

```
cat(paste(sapply(sessionInfo()$otherPkgs,function(x)
    paste(x$Package,x$Version)),collapse=", "))
```

texreg 1.37.5, memisc 0.99.27.3, MASS 7.3-54, latticeExtra 0.6-29, lattice 0.20-44, xtable 1.8-4, car 3.0-11, carData 3.0-4, Ecdat 0.3-9, Ecfun 0.2-5, knitr 1.33 .

© Oliver Kirchkamp

The Psycho Package

Several packages translate results into readable text. One of these packages is psycho. psycho requires R $\geq$ 3.4.0.

```
install.packages("psycho")
library(psycho)
```

```
library(Ecdat)
data(Caschool)
est <- lm (testscr ~ str + elpct + avginc,data=Caschool)
```

The standard output from R looks a bit dull:

```
summary(est)
```

```

Call:
lm(formula = testscr ~ str + elpct + avginc, data = Caschool)

Residuals:
    Min      1Q  Median      3Q     Max
-42.800  -6.862   0.275   6.586  31.199

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 640.31550    5.77489 110.879   <2e-16 ***
str          -0.06878    0.27691  -0.248    0.804
elpct        -0.48827    0.02928 -16.674   <2e-16 ***
avginc        1.49452    0.07483  19.971   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.35 on 416 degrees of freedom
Multiple R-squared:  0.7072,Adjusted R-squared:  0.7051
F-statistic: 334.9 on 3 and 416 DF,  p-value: < 2.2e-16
```

```
analyze(est)
```

The overall model predicting testscr (formula = testscr ~ str + elpct + avginc) explains 70.72% of the variance of the endogen (adj. R2 = 70.51). The model's intercept is at 640.32 (SE = 5.77, 95% CI [628.96, 651.67]). Within this model:

- The effect of str is not significant (beta = -0.069, SE = 0.28, 95% CI [-0.61, 0.48], t = -0.25, p > .1) and can be considered as very small (std. beta = -0.0068, std. SE = 0.027).
- The effect of elpct is significant (beta = -0.49, SE = 0.029, 95% CI [-0.55, -0.43], t = -16.67, p < .001) and can be considered as small (std. beta = -0.47, std. SE = 0.028).
- The effect of avginc is significant (beta = 1.49, SE = 0.075, 95% CI [1.35, 1.64], t = 19.97, p < .001) and can be considered as medium (std. beta = 0.57, std. SE = 0.028).

psycho's support for LaTeX is limited (requires some tinkering).

© Oliver Kirchkamp

2.7 Advantages

- Accuracy (no more mistakes from copying and pasting)
- Reproducability (even years later, it is always clear how results were generated)
- Dynamic document (changes are immediately reflected everywhere, this speeds up the writing process)

2.8 Practical issues

What if some calculations take too much time Usually you will not be able (or willing) to *always* do the entire journey from your *raw data* to the *paper* in one single step.

```
<<fastOrSlow>>=
FAST=FALSE
@

<<eval=!FAST>>=
read.csv('rawData.csv')
expensiveData<-thisTakesALongTime()
save(expensiveData,file='expensive.Rdata')
@

<<>>=
load('expensive.Rdata')
...
@
```

Switch `FAST` to `TRUE` when you have more time and if you want to re-generate the data.

Alternatively: caching intermediate results
knitr can also *cache* intermediate results:

```
<<expensiveStep,cache=TRUE>>=
intermediateResults <- ....
@
```

The above chunk is executed only once (unless it changes), results are stored on disk and can be used lateron.

(`knitR` tries hard to understand how chunks depend on each other. Still, this automatic process might fail. You can use `dependson` or, to be safe, clear the cache. You can set the cache path (at the beginning of your paper) as follows:

```
opts_chunk[["set"]](fig.path='myFigures/paperX',
                    cache.path='myCache/paperX')
```

In particular when versions of R libraries change, the new version might find it hard to make sense of the old data.

To clear old results:

© Oliver Kirchkamp

```
unlink("myCache/paperX*")
unlink("myFigures/paperX*")
```

2.9 When R produces tables

2.9.1 Tables

You can save a lot of work if you harness R to create and format your tables for you. A versatile function is `xtable` (for Markdown look at `huxtable`):

```
set.seed(123)
(x <- matrix(rnorm(6),2,3))
```

```
           [,1]       [,2]      [,3]
[1,] -0.5604756 1.55870831 0.1292877
[2,] -0.2301775 0.07050839 1.7150650
```

```
<<results='asis'>>
```

```
library(xtable)
xtable(x)
```

	1	2	3
1	-0.56	1.56	0.13
2	-0.23	0.07	1.72

```
@
```

You can label rownames and columnames:

```
<<results='asis'>>
```

```
colnames(x)<-c("$\\alpha$","$\\beta$","$\\gamma$")
rownames(x)<-c("One","Two")
xtable(x)
```

	α	β	γ
One	-0.56	1.56	0.13
Two	-0.23	0.07	1.72

```
@
```

Note that `xtable` sanitizes all entries. Hence, what was meant to look like α is shown as `$\alpha$`.

© Oliver Kirchkamp

```
<<results='asis'>>
```

```
options(xtable.sanitize.colnames.function=function(x) x)
colnames(x)<-c("$\\alpha$","$\\beta$","$\\gamma$")
rownames(x)<-c("One","Two")
xtable(x)
```

	α	β	γ
One	-0.56	1.56	0.13
Two	-0.23	0.07	1.72

```
@
```

2.9.2 Regression results

```
library(Ecdat)
data(Caschool)
est1<-lm(testscr ~ str,data=Caschool)
xtable(summary(est1))
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	698.9330	9.4675	73.82	0.0000
str	-2.2798	0.4798	-4.75	0.0000

2.9.3 Mixed effects

If we use `lmer` to estimte models with mixed effects, we have a number of possibilities calculating p-values. The `lmerTest` packages uses Satterthwaite's degrees of freedom method.

```
library(lme4)
library(lmerTest)
fm1 <- lmer(Informed.liking ~ Product +
            (1|Consumer) , data=ham)
xtable(summary(fm1)[["coefficients"]])
```

	Estimate	Std. Error	df	t value	Pr(>|t|)
(Intercept)	5.81	0.19	320.68	30.11	0.00
Product2	-0.70	0.23	564.00	-3.03	0.00
Product3	0.28	0.23	564.00	1.22	0.22
Product4	0.12	0.23	564.00	0.50	0.61

© Oliver Kirchkamp

2.9.4 Comparison of several estimations

Several libraries format estimation results (e.g. `mtable`) in columns per estimation. Here we use `texreg` (for Markdown look at `huxtable`).

```
est1 <- lm(testscr ~ str,data=Caschool)
est2 <- lm(testscr ~ str + elpct,data=Caschool)
est3 <- lm(testscr ~ str + elpct + avginc,data=Caschool)
texreg(list(est1,est2,est3),table=FALSE)
```

	Model 1	Model 2	Model 3
(Intercept)	698.93***	686.03***	640.32***
	(9.47)	(7.41)	(5.77)
str	−2.28***	−1.10**	−0.07
	(0.48)	(0.38)	(0.28)
elpct		−0.65***	−0.49***
		(0.04)	(0.03)
avginc			1.49***
			(0.07)
R^2	0.05	0.43	0.71
Adj. R^2	0.05	0.42	0.71
Num. obs.	420	420	420

***p < 0.001; **p < 0.01; *p < 0.05

Nicer names for equations

```
texreg(list(small=est1,medium=est2,`model $\\gamma$`=est3),
       table=FALSE)
```

	small	medium	model γ
(Intercept)	698.93***	686.03***	640.32***
	(9.47)	(7.41)	(5.77)
str	−2.28***	−1.10**	−0.07
	(0.48)	(0.38)	(0.28)
elpct		−0.65***	−0.49***
		(0.04)	(0.03)
avginc			1.49***
			(0.07)
R^2	0.05	0.43	0.71
Adj. R^2	0.05	0.42	0.71
Num. obs.	420	420	420

***p < 0.001; **p < 0.01; *p < 0.05

© Oliver Kirchkamp

2.9.5 Comparing models with mixed effects

```
library(lme4)
library(lmerTest)
fm1 <- lmer(Informed.liking ~ Product + (1|Consumer) , data=ham)
fm2 <- lmer(Informed.liking ~ Product*Information + (1|Consumer),data=ham)
texreg(list(fm1,fm2),table=FALSE,single.row=TRUE)
```

	smaller model	larger model
(Intercept)	5.81 (0.19)***	5.73 (0.25)***
Product2	−0.70 (0.23)**	−0.83 (0.33)*
Product3	0.28 (0.23)	0.15 (0.33)
Product4	0.12 (0.23)	0.30 (0.33)
Information2		0.16 (0.33)
Product2:Information2		0.25 (0.46)
Product3:Information2		0.27 (0.46)
Product4:Information2		−0.36 (0.46)
AIC	2884.29	2889.97
BIC	2911.13	2934.71
Log Likelihood	−1436.14	−1434.99
Num. obs.	648	648
Num. groups: Consumer	81	81
Var: Consumer (Intercept)	0.83	0.83
Var: Residual	4.38	4.38

***p < 0.001; **p < 0.01; *p < 0.05

2.10 Alternatives to LaTeX

`knitr` can create other formats

- Markdown (`md`) → `html`, `docx`, `odt`...

Incremental assembly

- ReportRs: docx, odt
- pander: pandoc, HTML, PDF, docx, odt

(similar to Stata's `putdocx`)

2.10.1 Markdown

````
text that explains what you are doing and why it is
interesting ...

```r
````
````

© Oliver Kirchkamp

````
library(Ecdat)
library(xtable)
library(lattice)
data(Caschool)
attach(Caschool)
est <- lm(testscr ~ avginc)
kable(anova(est))
```

```r
 xyplot(testscr ~ avginc,xlab="average income",ylab="testscore",
     type=c("p","r","smooth"))
```
the correlation between average income and testscore is
`r round(cor(testscr,avginc),4)`
more text ...
````

Translate `Rmd` into `html`, `odt`, `docx`...

```
library(knitr)
knit("<filename.Rmd>")
pandoc("<filename.md>","docx")
```

2.10.2 Incremental assembly

```
library(ReporteRs)
myDoc <- docx()
myDoc <- addParagraph(myDoc, " ... ")
myTable <- FlexTable ( data = mtcars )
myDoc <- addFlexTable(myTable)
writeDoc (myDoc, file="<filename.doc>")
```

(similar to Stata's `putdocx`)

2.11 The magic of GNU make

In the same directory where I have my Rnw file, I also have a file that is called `Makefile`. Let us assume that the current version of my Rnw file is called `myProject_160601.Rnw`. Then here is my `Makefile`

```
PROJECT = myProject_160601

pdf: $(PROJECT).pdf

%.pdf: %.tex
     pdflatex $<
```
```

© Oliver Kirchkamp

```
%.tex: %.Rnw
	echo "library(knitr);knit(\"$<\");" | R --vanilla
```

Let us go through the individual lines of this Makefile.

```
PROJECT = myProject_160601
```

Here we define a variable. This is useful, since this most of the time the only line of the Makefile I ever have to change (instead of changing every occurence of the filename)

```
pdf: $(PROJECT).pdf
```

The part `pdf` before the colon is a target. Since it is the *first* target in the file it is also the *default* target. I.e. make will try to make it whenever I just say

```
make
```

Make will do the same when I call it explicitely

```
make pdf
```

The part after the colon tells make on which file(s) the target actually *depends* (the *prerequisites*). Here it is only one but there could be several. If all prerequisites exists, and if they are up-to-date (newer than all files they depends on), make will apply the rule. Otherwise, make will try to create the prerequisites (the `pdf` file in this case, with the help of other rules) and then apply this rule.

```
%.tex: %.Rnw
	echo "library(knitr);knit(\"$<\");" | R --vanilla
```

This is a rule that make can use to create `tex` files. So above we requested the `pdf` file `myProject_160601.pdf`, and now make knows that we require a file `myProject_160601.tex`. If this already exists and is up-to-date (i.e. newer than all files it depends on), make will apply this rule. Otherwise, make will first try to create the prerequisite (the single `tex` file in this case would be created with the help of other rules) and then apply this rule.

To create our `pdf` it is now sufficient to say (from the command line, not from R)

```
make
```

and make will do everything that is needed.

Note 1: In this context a simple shell script would work almost as well. However, `make` is very helpful when your pdf file depends on more than one `tex` or `Rnw` file.

Note 2: On BSD Systems GNU Make is called `gmake`, not `make`.

© Oliver Kirchkamp

A Makefile for a larger project

When I wrote this handout I split it into several Rnw files. This saves time. When I make changes to one part, only this part has to be compiled again. The files were all in the same directory. The directory also contained a "master"-tex file that would assemble the tex-files for each Rnw-file.

The following example shows how we assemble the output of several files to make one document:

```
PROJECT = myProject_160601
RPARTS        = $(wildcard $(PROJECT)_[1-9].Rnw)
TEXPARTS      = $(RPARTS:.Rnw=.tex)

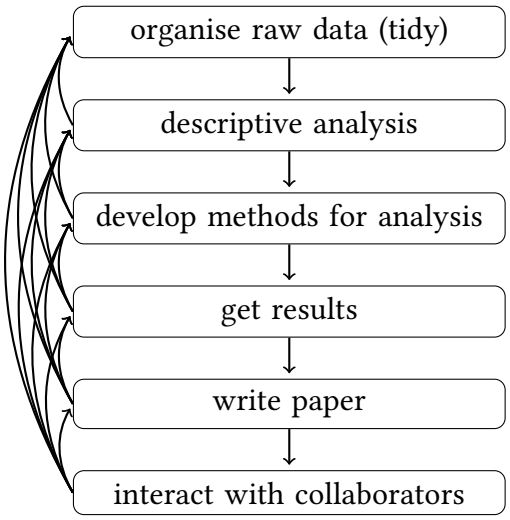
pdf: $(PROJECT).pdf

# our project depends on several files:
$(PROJECT).pdf: $(TEXPARTS) $(PROJECT).tex
        pdflatex $(PROJECT)

# only the tex files who belong to Rnw files
#                               should be knitted:
$(TEXPARTS)   : %.tex  : %.Rnw ; \
        echo "library(knitr);knit(\"$<\");" | R --vanilla
```

3 Version control

3.1 Non-linear workflow



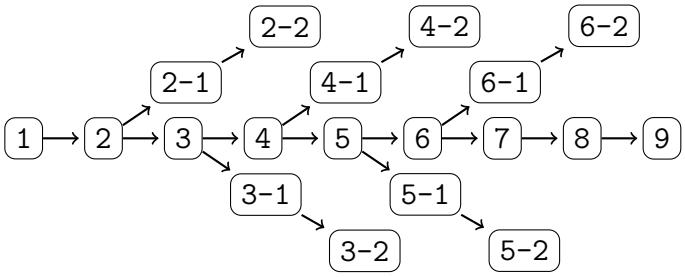
Solutions and restrictions: During our research we create a lot of intermediate results. How can we organise these results?

© Oliver Kirchkamp

- Store everything? — Not feasible! (Often our analysis will create a lot of data. Later we don't know what is relevant and what is not.)
- We want to be creative, take shortcuts, we want to explore things, play with different representations of a solution…
- During this phase we can not document everything.

3.2 Creativity and chaos

Progress in research is not linear:



→ ideally: version control, otherwise: creativity and chaos

CVS, SVN, Git, Mercurial, Bazaar, ⋮ } can store **text** efficiently (only differences between versions) → script for empirical part (R), markup language for text LaTeX, ⋮

Alternatively: Living two lives:

- creative (undocumented)
- permanent (documented)

(We must be aware whether we are in "creative" or in "permanent" mode).

Let our computer(s) reflect these two lives:

```
.../projectXYZ/
              /permanent/
                        /rawData
                        /cleanData
                        /R
                        /Paper
                        /Slides
              /creative/
                        /cleanData
                        /R
                        /Paper
                        /Slides
```

© Oliver Kirchkamp

You might need more directories for your work.

(In terms of version control, which we will cover later, "permament" could be a `trunk`, while "creative" could be a `branch`)

Rules

1. Anything that we give to other people (collaborators, journals,...) must come entirely from *permanent*
2. Never delete anything from *permanent*
3. Never change anything in *permanent*
4. We must be able trace back everything in *permanent* clearly to our raw data.

Since we give things to other people more than once (first draft, second draft,..., first revision, ..., second revision,...), we must be able to replicate each of these instances.

Consequences — permanent data has versions (Below we will discuss the advantages of a version control system (`git`, `svn`). Let us assume for a moment that we have to do everything manually.)

- We will accumulate versions in our *permanent* life (do not delete them, do not change them)

```
cleaned_data_180521.Rdata
cleaned_data_180522.Rdata
cleaned_data_180522b.Rdata
⋮
preparingData_180521.R
preparingData_180522.R
descriptives_180522.R
econometrics_180523.R
⋮
paper_180524.Rnw
paper_180525.Rnw
paper_180527.Rnw
⋮
```

PhDcomicFinal:

- Student without a name completes document `FINAL.doc`!
- Student gets feedback from Prof. Smith.
- Student without a name types `FINAL_rev.2.doc`!

© Oliver Kirchkamp

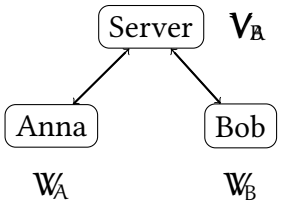
- …more feedback from Prof. Smith.
- Student without a name types…
- …Prof. Smith reading `FINAL_rev.8.comments5.CORRECTIONS.doc`…
- …`FINAL_rev.18.comments7.corrections9.MORE.30.doc`…
- ⋮

Permanent data has versions

- Nobody wants to see all these versions at the same time.
- Version control shows only the "relevant" version to us – still, all other versions are preserved.

3.3 Problem I – concurrent edits

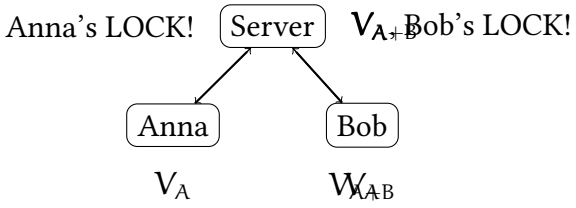
What happens if two authors, Anna and Bob, simultaneously want to work on the same file. Chances are that one is deleting the changes of the other. (This problem is similar to one author working on two different machines)



- Anna's work is lost — very inefficient (50% of the contribution is lost)

3.4 A "simple" solution: locking

Serialising the workflow might help. Anna could put a "lock" on a file while she wants to edit this file. Only when she is finished, the "unlocks" the file and Bob can continue.

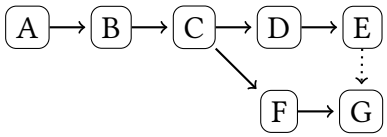


- Bob can only work with Anna's permission — very inefficient (50% of the time Anna and Bob are forced to wait)

© Oliver Kirchkamp

3.5 Problem II – nonlinear work

Even when Anna works on a problem on her own she can be in conflict with herself. Imagine the following: Anna successfully completed the steps A, B, and C on a paper and has now something readable that she could send around. Perhaps she actually has sent it around. Now she continues to work on some technical details D and E, but so far her work in incomplete – D and E are not ready for the public. Suddenly the need arises to go back to the last public version (C) and to add some work there (e.g. Anna decides to submit the paper to a conference, but wants to rewrite the introduction and the conclusion. It will take too much time to first finish the work on D and E, so she has to go back to C. Rewriting the introduction and conclusion are steps F and G. Once the paper (G) has been submitted, Anna wants to return to the technical bits D and E and merge them with F and G.



3.6 “Simple solutions”

- Keep different version of files with different names...

  ```
  FINAL.doc FINAL_rev.2.doc FINAL_rev.8.comments5.CORRECTIONS.doc FINAL_rev.18
  ...
  ```

- Filesystems with snapshots
  - ZFS, BtrFS,...
- Cloud storage with limited snapshots

3.7 Version control

(revision control, source control) Traditional:

- Editions of a book
- Revisions of a specification
- ⋮

Software:

- Concurrent Versions System (CVS)
- Subversion (SVN)
- Git

© Oliver Kirchkamp

- Mercurial
- Bazaar
- ⋮

In this course we will use Git.

- Free
- Distributed repository
- Supports many platforms, formats
- ⋮

3.8 Solution to problem II: nonlinear work

Before we create our first git-repository, we have to provide some basic information about ourselves:

```
git config --global user.name "Your Name Comes Here"
git config --global user.email you@yourdomain.example.com
```

Now we can create our first repository:

```
git init
```

We can check the current “status” as follows:

```
git status
```

git status

```
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```

now we create a file `test.Rnw`

git status

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       test.Rnw
nothing added to commit but untracked files present (use "git add" to track)
```

```
git add test.Rnw
```

© Oliver Kirchkamp

git status

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#        new file:   test.Rnw
```

```
git commit -a -m "first version of test.Rnw"
```

git status

```
# On branch master
nothing to commit, working directory clean
```

```
git log --oneline
```

git log --oneline

```
3ea6194 first version of test.Rnw
```

Note that git denotes versions with identifiers like “3ea6194” (and not A, B, C).
After some changes to `test.Rnw`...

git status

```
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#        modified:   test.Rnw
#
no changes added to commit (use "git add" and/or "git commit -a")
```

```
git commit -a -m "introduction and first results"
```

git status

```
# On branch master
nothing to commit, working directory clean
```

git log --oneline

```
74fd521 introduction and first results
3ea6194 first version of test.Rnw
```

More changes and...

```
git commit -a -m "draft conclusion"
```

more changes and...

```
git commit -a -m "improved regression results (do not fully work)"
```

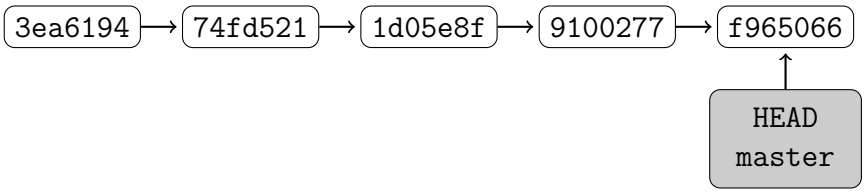
more changes and...

```
git commit -a -m "added funny model (does not fully work yet)"
```

© Oliver Kirchkamp

git log --oneline

```
f965066 added funnyModel model (does not fully work yet)
9100277 improved regression results (do not fully work)
1d05e8f draft conclusion
74fd521 introduction and first results
3ea6194 first version of test.Rnw
```



Assume we want to go back to `1d05e8f` but not forget what we did between `1d05e8f` and `f965066`.

Remember current state:

```
git branch funnyModel
```

Now that we have given the current branch a name we can revert to the old state:

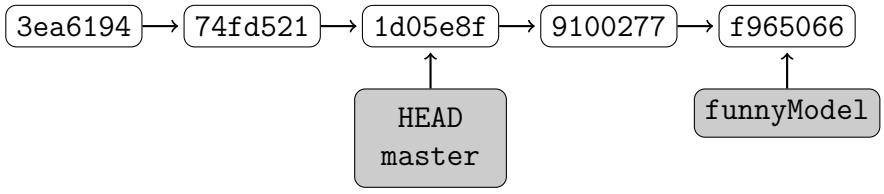
```
git reset 1d05e8f
```

```
Unstaged changes after reset:
M       test.Rnw
```

```
git checkout test.Rnw
```

git status

```
# On branch master
nothing to commit, working directory clean
```

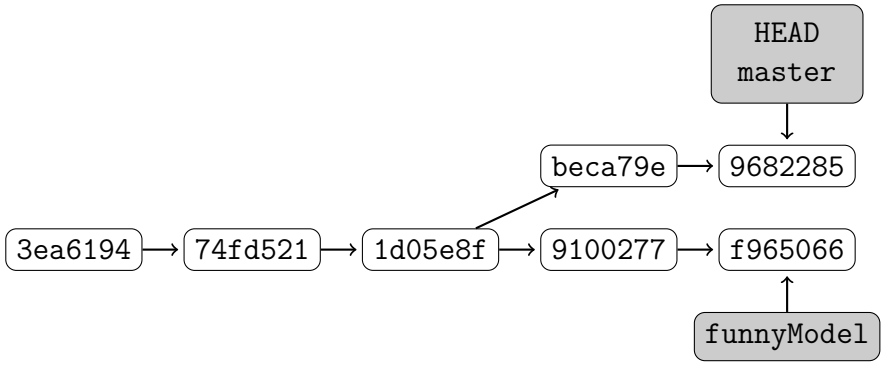


do more work...

```
git commit -a -m "rewrote introduction"
```

do even more work...

```
git commit -a -m "rewrote conclusion, added literature"
```



© Oliver Kirchkamp

eventuelly we want to join the two branches:

```
git merge funnyModel
```

now two things can happen: Either this...

```
Merge made by recursive.
 test.Rnw |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
```

or that...

```
Auto-merging test.Rnw
CONFLICT (content): Merge conflict in test.Rnw
Automatic merge failed; fix conflicts and then commit the result
```

We can fix this with `git mergetool`:

```
git mergetool
```

```
  Merging:
test.Rnw

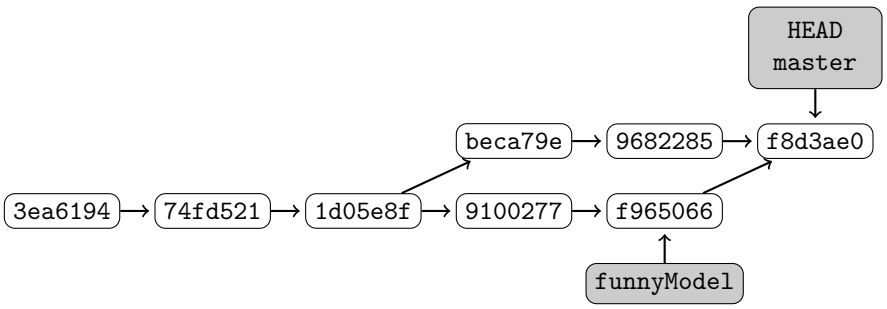
Normal merge conflict for 'test.Rnw':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (meld):
```

Now we can make detailed merge decisions in an editor.

```
git commit -m "merged funnyModel"
```

To make the previous part work...

- you need a mergetool installed (have a look at `meld`)
- you either tell `git` to use this tool (`git mergetool --tool=meld`)
- or you tell `git` once and for all that a specific tool is your favourite:
  `git config --global --add merge.tool meld`
- (you can do the same for the difftool:)
  `git config --global --add diff.tool meld`



© Oliver Kirchkamp

3.9 Solution to problem I: concurrent edits

Version control allows all authors to work on the file(s) simultaneously.

In this example we start with an empty repository. In a first step both Anna and Bob "checkout" the repository, i.e. they create a local copy of the repository on their computer.

Anna creates a file, adds it to version control and commits it to the repository. Bob then updates his copy and, thus, obtains Anna's changes.

- First step: create a "bare" repository on a "server"

```
git --bare init
```

- This repository can now be accessed from "clients", either on the same machine…

```
git clone /path/to/repository/
```

…or on a different machine via `ssh` (where `user` has access rights):

```
git clone ssh://user@my.server.org/path/to/repository
```

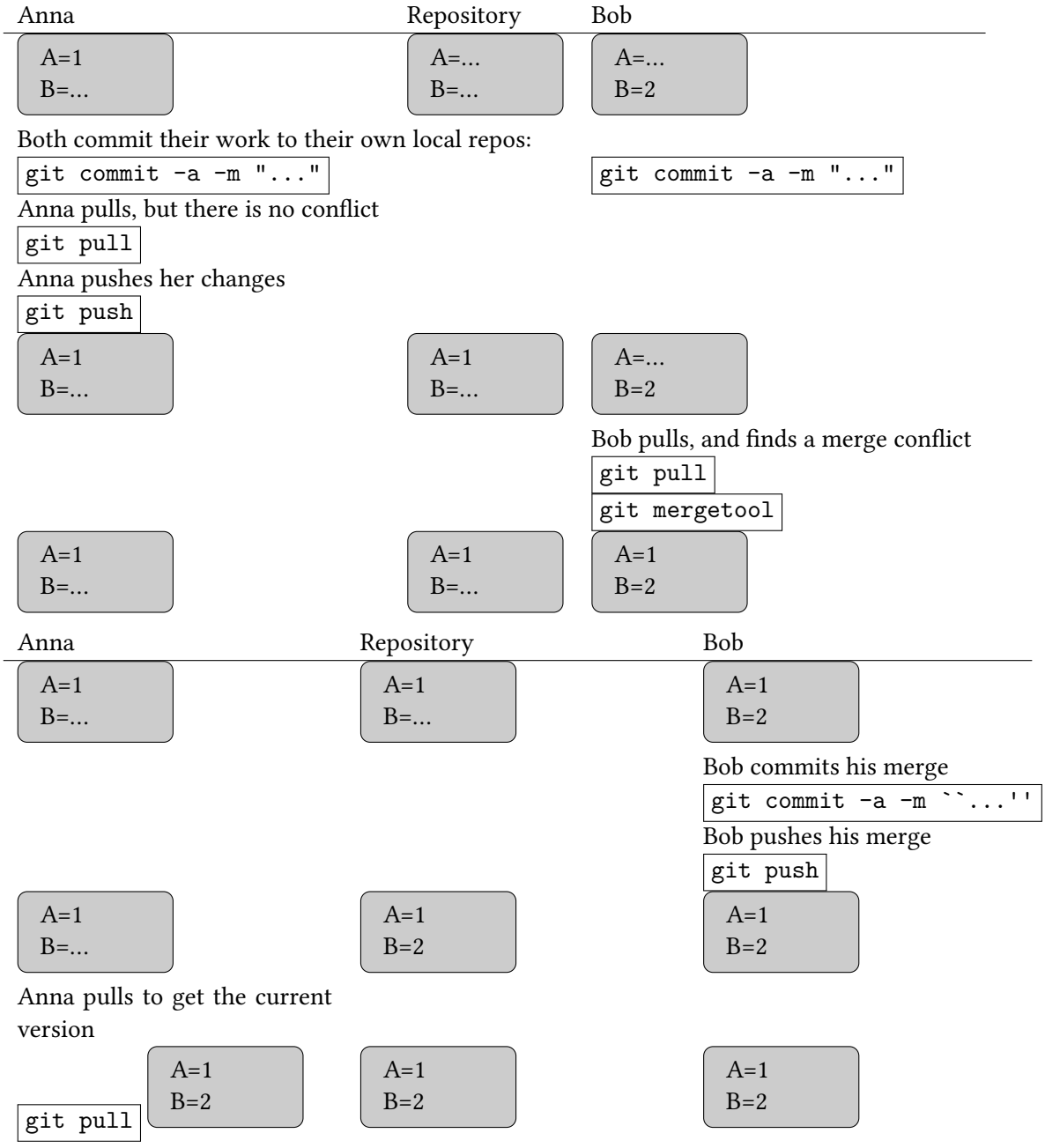
Anna	Repository	Bob
	empty	
`git clone ...`		`git clone ...`
creates a file `test.Rnw`: A=… B=…		
`git add test.Rnw`		
`git commit`		
uploads the file: `git push`		
A=… B=…	A=… B=…	
		`git pull`
A=… B=…	A=… B=…	A=… B=…

3.10 Edits without conflicts:

To make this more interesting we now assume that both work on the file. Anna works on the upper part (A), Bob works on the lower part (B). Both update and commit their changes.

© Oliver Kirchkamp

Since they both edit different parts of the file, the version control system can silently merge their changes.



3.11 Going back in time

Version control is not only helpful to avoid conflicts between several people, it also helps when we change our mind and want to have a look into the past. `git log` provides a list of the different revision of a file:

```
git log --oneline
```

© Oliver Kirchkamp

```
f965066 added funny model (does not fully work yet)
9100277 improved regression results (do not fully work)
1d05e8f draft conclusion
74fd521 introduction and first results
3ea6194 first version of test.Rnw
```

`git blame` allows you to inspect modifications in specific files. If we want to find out who introduced or removed "something specific" (and when), we would say...

```
git blame -L '/something specific/' test.Rnw
```

```
19eb9bac (w6kiol2 2016-06-17 ...) therefore important to study something specific which
dd0647f7 (w6kiol2 2016-06-21 ...) switched our focus to something else and continue with
```

There is a range of GUIs that allow you to browse the commit tree.
Try, e.g., `gitk`

3.12 git and subversion

- git: Local and Remote
- subversion: Remote only

  git can use subversion as a remote repository:

`git clone`	`git svn clone`	
`git pull`	`git svn rebase`	
`git commit`	`git commit`	←no need to change
`git push`	`git svn dcommit`	

- Conceptual differences:
  - subversion has only one repository (on the server), git has one or more local repositories plus one or more on different servers.
  - inconsistent uploads to a server:

    subversion will not complain if after a `push/commit` the state on the server is different from the state on any of the clients. git will not allow this (git forces you to pull first, merge, commit, and push then)

3.13 Limitations

3.13.1 General thoughts

git works well on text files (LaTeX, `Rnw`, `md`, `Rmd`, `R`,...).

Git can not understand changes in binary files (Pictures, `PDF`, `Rdata`, files created by office software...).

© Oliver Kirchkamp

- If a binary file is based on a text file (e.g. a graph is created from an `R` file), then the text file should be stored and should be under version control. The binary file can always be recreated from the text file.
- We should organise our work such that (if possible) only text files define the work.
- If binary files are unavoidable, they should not change frequently.

3.13.2 Interaction with Office software

If a coauthor insists on using office software (which stores files as binaries)...

- Convert office file into text file (e.g. with `pandoc`) and version control the text file.

```
pandoc officeDocument.docx -o paper.tex
git add paper.tex
git commit -a "added tex version of office document"
git push
⋮
git pull
pandoc paper.tex -o newVersionOfOfficeDocument.docx
```

→ conversion will lose parts of the paper (formulae).

Process does not work too well.
→ perhaps using a visual editor for markdown could be a compromise.

3.14 Steps to set up a subversion repository at the URZ at the FSU Jena

If you need to set up a subversion repository here at the FSU, tell me about it and tell me the ⟨*urz-login*⟩s of the people who plan to use it. Technically, setting up a new repository means the following:

- ssh to subversion.rz.uni-jena.de
- `svnadmin create /data/svn/ewf/`⟨*repository*⟩
- `chmod -R g+w /data/svn/ewf/`⟨*repository*⟩
- set access rights for all involved ⟨*urz-login*⟩s in `/svn/access-ewf`
- then, at the local machine in a directory that actually contains only the files you want to add: `svn --username` ⟨*urz-login*⟩ `import . https://subversion.rz.uni-jena.de/sv "Initial import"`

  (this "imports" data into the repository)
- then, at all client machines,

  `svn --username` ⟨*urz-login*⟩ `checkout https://subversion.rz.uni-jena.de/svn/ew`

© Oliver Kirchkamp

3.15 Setting up a subversion repository on your own computer

- On your own computer: `svnadmin create` ⟨*path*⟩/⟨*repository*⟩

  (⟨*path*⟩is a complete path, e.g. `/home/user/Documents/` or `/C:MyDocuments/`)

- then, in a directory that actually contains only the files you want to add:

  `svn import . file://`⟨*path*⟩/⟨*repository*⟩ `-m "Initial import"`

- then, wherever you actually want to work on your own computer:

  `svn checkout file://`⟨*path*⟩/⟨*repository*⟩

- if you have `ssh` access to your computer you can also say from other machines:

  `svn checkout svn+ssh://`⟨*yourComputer*⟩/⟨*path*⟩/⟨*repository*⟩

3.16 Usual workflow with git

While setting up a repository looks a bit complicated, using it is quite simple:

- `git pull` check whether the others did something
- editing
  - `git add` add a file to version control
  - `git mv` move a file under version control
  - `git rm` delete a file under version control
- `git commit` commit own work to local repository
- `git pull` check whether the others did something
- `git mergetool` merge their changes
- `git commit` commit merge
- `git push` upload everything to the server

© Oliver Kirchkamp

3.17 Exercise

3.17.1 SVN

Create (in ⟨*path*⟩) four directories `A`, `B`, `C`.

From `A` create a repository: `svnadmin create ../R`

In `A` create a file `test.txt` with some text:

```
A=...
B=...
```

Initial import. In `A` say:

```
svn import . file://⟨path⟩/R -m "My first initial import"
```

in `B`:	in `C`:
`svn checkout file://⟨path⟩/R`	`svn checkout file://⟨path⟩/R`
in `B/R`:	in `C/R`:
Simultaneous changes to `test.txt`	
`A=1` `B=...`	`A=...` `B=2`
Commit changes	
`svn commit`	`svn commit`
Update	
`svn update`	`svn update`

© Oliver Kirchkamp

3.17.2 Git

Create (in ⟨*path*⟩) four directories `A`, `B`, `C`.

In `A` create a repository:
```
git init
```

In `A` create a file `test.txt` with some text:
```
A=...
B=...
```

In A: stage and commit
```
git add test.txt
git commit -am "first commit"
```

In `R`: create a remote repository
```
git init --bare ../R
```

In A: make R a remote of A
```
git remote add origin ../R
git push --set-upstream origin master
```

In A: push work from A to R
```
git push
```

In B: checkout from R to B:
```
git clone ../R
```

In C: checkout from R to C:
```
git clone ../R
```

in `B/R`:	in `C/R`:
Simultaneous changes to `test.txt`	
`A=1` `B=...`	`A=2` `B=...`
Commit changes	

`git commit -am "change at B"`
`git pull`
`git push`
	`git commit -am "change at C"`
	`git pull`
	`git mergetool`
	`git commit -am "merge..."`
	`git push`

4 Organising work

4.1 Scripting

Most of the practical work in data analysis and statistics can be seen as a sequence of commands to a statistical software.

How can we run these commands?

Execute commands in command window
(or with mouse and dialog boxes)

- clumsy
- hard to repeat actions

© Oliver Kirchkamp

- hard to replicate what we did and why we did it (logs don't really help).
- hard to find mistakes (structure of the mistake is easy to overlook).

Write file (`.Rnw`, `.R`, `.do`)
execute single lines (or small regions) from the file while editing the file.

- great way to creatively develop code line by line.
  Not reproducible since the file changes permanently.
- one window with the file, another window with mainly the R output

Write source file (`.Rnw`, `.R`, `.do`)
open it in an editor and then always execute the entire file (while editing the file).

- great way to creatively develop larger parts of code

Steps of analysis are in…

Source files
Source "public" files (`.R` or `.do`) from a "master file"

```
source("read_data_180715.R")
source("clean_data_180715.R")
source("create_figures_180715.R")
```

This is the first step to reproducible research. When our script seems to do what it is supposed to do, we make it "public", give it a unique name, and never change it again.

Functions
From a master file, first source a file which defines functions. Then call these functions.

```
source("functions_XYZ_180715.R")
read_data()
clean_data()
create_figures()
```

Better:

- Functions (which belong together) are kept together in one file.
- Functions can be more flexible and more transparent.

Advantages of using source files (with or without functions):

- We keep a record of our work.

© Oliver Kirchkamp

- We can work incrementally, fix mistakes and introduce small changes (if we refer to a public file, we should work on a copy of this file with a new name).
- We can use the editor of our choice (Emacs is a nice editor)

Advantage of using functions:

- functions can take parameters.
- several functions go in one file (still do not harm each other).

  Systematic changes are easier with only one file (things that belong together stay together).

Regardless whether we divide our work into source files or into functions: This division allows us to save time. Some of these steps take a lot of time. Once they work, we do not have to do them over and over again.

4.2 Robustness

How can we make our work "robust"? Remember:

- The structure of the data may change over time.
  - New variables might come with new treatments of our experiment.
  - New treatments might require that we code variables differently.
- Commands may not only run on our computer.
- Commands are not always sourced in the same context.
- Our random number generator may start from different seeds.

4.2.1 Robustness towards different computers

We better use relative pathnames.

Assume that on my computer the script is stored in

```
/home/oliver/projectXYX/R
```

next to it we have

```
/home/oliver/projectXYX/data/munich/1998/test.Rdata
```

From the script I might call either (absolute path)

```
load(file="/home/oliver/projectXYX/data/munich/1998/test.Rdata")
```

or (relative path)

© Oliver Kirchkamp

```
load(file="../data/munich/1998/test.Rdata")
```

The latter assumes that there is a file
`../data/munich/1998/test.Rdata`
next to the script. But it does *not* assume that everything is in
`/home/oliver/projectXYZ`

Hence, the latter works even if my coauthor has stored everything as

```
C:/users/eva/PhD/projectXYX/R
C:/users/eva/PhD/projectXYX/data/munich/1998/test.Rdata
```

If a lot happens in `../data/munich/1998/` anyway, use the `setwd` command

```
setwd("../data/munich/1998/")
...
load(file="test.Rdata")
```

(and remember to make the `setwd` relative, i.e. avoid the following:

```
setwd("/home/oliver/projectXYZ/data/munich/1998/")
...
```

).

4.2.2 Robustness against changes of directories

Although the following function might change the working directory, `on.exit()` remembers to revert the original state.

```
abc <- function() {
  oldDir <- setwd("...new directory...")
  on.exit(setwd(oldDir))
  do.something(...)
  do.something.else(...)
}
```

4.2.3 Robustness against changes in context

Assume we have the following two files

```
# script1.R
load("someData.Rdata")
# now two variables, x and y are defined
source("script2.R")
```

The content of `script2.R` might be this:

© Oliver Kirchkamp

```
# script2.R
est <- lm ( y ~ x)
```

In this example `script2.R` *assumes* that variables y and x are defined. As long as `script2.R` is called in this context, everything is fine.

Changing `script1.R` might have unexpected side effects since we transport variables from one script to the other. The call

```
source("script2.R")
```

does not reveal how y and x are used by the script.

4.2.4 Verify assumptions

Often we assume a condition, but we can not be really sure:

- Does an estimation really converge?
- Does a subset of the data really contain (sufficiently many) observations?
- Does a file really exist?
- Do the explanatory variables really have the necessary properties?
- ⋮

```
if (...) stop("...informative error message...")
```

If we don't stop with an informative error,

- R will stop with an obscure error, or
- we will get wrong results (and we might not notice).

4.3 Functions

4.3.1 Functions increase robustness

```
# script1.R
source("script2.R")
load("someData.Rdata")
myFunction(y=a,x=b)
```

```
# script2.R
# defines myFunction
myFunction <- function(y,x) {
  est <<- lm ( y ~ x)
}
```

© Oliver Kirchkamp

Now `script2.R` only defines a function. The function has arguments, hence, when we use it in `script1.R` we realise which variable goes where.

Note that the function takes *arguments*. This is more elegant (and less risky) than passing parameters as global variables:

```
# script1.R
source("script2.R")
load("someData.Rdata")
y <- a
x <- b
myFunction()
```

with a function without parameters:

```
# script2.R
# defines myFunction
myFunction <- function() {
  est <<- lm ( y ~ x)
}
```

It will still work, but later it will be less clear to us that the assignments before the function call are essential for the function.

Side effects:

```
myFunction <- function(y,x) {
  est <<- lm ( y ~ x)
}
```

This function has a *side effect*. It changes a variable `est` outside the function. Often it is less confusing to define functions with `return values` and no side effects.

No side effects:

```
myFunction <- function(y,x) {
  lm ( y ~ x)
}
```

When we call this function later as

```
est <- myFunction(y,x)
```

it is clear where the result of the function goes.

Recap

- Functions which use *global variables*: risky

  → better: Functions which take parameters.

© Oliver Kirchkamp

- Functions with *side effects*: risky

  → better: Functions which return values.

Note: If we use *scripts* instead of *functions*:

→ Scripts must use *global variables* and can only produce *side effects*.

→ Scripts are more likely to lead to *mistakes* than functions.

→ Replace scripts by *functions* (with arguments) whenever possible.

4.4 Calculations that take a lot of time

If a sequence of functions takes a lot of time to run, let it generate intermediate data.
Our master-R-file could look like this:

```
set.seed(123)
...
source("projectXYZ_init_180715.R")
getAndCleanData() # takes a lot of time
save(cleanedData,file="cleanedData180715.Rdata")

load("cleanedData180715.Rdata")
doBootstrap() # takes a lot of time
save(bsData,file="bsData180715.Rdata")

load("cleanedData180715.Rdata")
load("bsData180715.Rdata")
doFigures()
...
```

When we develop code, we could work on parts of the code, without having to do expensive calculations to prepare our work.

4.5 Nested functions

If our functions become long and complicated, we can divide them into small chuncs. Define outside:

```
...
doAnalysis <- function () {
  firstStep()
  secondStep()
  thirdStep()
  ...
}

firstStep <- function() {
  ...
}
```

© Oliver Kirchkamp

```
secondStep <- function() {
  ...
}
...
```

Define inside:
Actually, if we need some functions only within a specific other function then we can define them *within* this function:

```
...
doAnalysis <- function () {
  firstStep <- function() {
  ...
  }
  secondStep <- function() {
  ...
  }
  firstStep()
  secondStep()
  thirdStep()
  ...
}
```

- Advantage of inside: Functions are *only visible* from within `doAnalysis` and can do *no harm elsewhere* (where we, perhaps, defined functions with the same name that do different things).

Nesting of functions has three advantages:

- It structures our work.
- It facilitates debugging.
- It facilitates communication with coauthors. (we can say: "...there is a problem in `thirdStep` in `doAnalysis`...")

4.6 Reproducible randomness

```
set.seed(123)
```

Random numbers affect our results:

- Simulation
- MCMC samples
- Bootstrapping
- Approximate permutation tests
- Selection of training and confirmation samples
- ...

© Oliver Kirchkamp

4.7 Exploit structure

- If there is a systematic structure in our problem, then we can exploit it
- If we make mistakes, we make them systematically!

```
N <- 100
profit88 <- rnorm(N)
profit89 <- rnorm(N)
profit98 <- rnorm(N)
myData <- data.frame(profit88,profit89,profit98)
```

Compare

```
t.test(profit88,data=myData)$p.value
t.test(profit89,data=myData)$p.value
t.test(profit98,data=myData)$p.value
```

with

```
library(dplyr)
myData %>%
    summarise(across(starts_with("profit"),
                     function(x) t.test(x)$p.value))
```

The first looks simpler.
The second is more robust against

- a change in the dataset (instead of `myData` we now use `myDataClean`)
- a change in the names of the variables (`profit` becomes `Profit_`)
- adding another `profit`-variable (`profit2016`...)
- typos (use `profit88` twice, instead of `profit88` and `profit89` once each.

4.8 Human readable scripts

- Weaving and knitting → Section 2.
- Comments at the beginning of each file

```
# scriptExample180715.R
#
# the purpose of this script is to illustrate the use of
#                                                 comments
#
# first version: 180715
# this version:  180715
# last change by: Oliver
# requires:      test180715.Rdata, someFunctions180715.R
# provides:      ...
#
set.seed(123)
```

© Oliver Kirchkamp

- Comments at the beginning of each function

```
#
# exampleFun transforms two vectors into an example
# side effects: ...
# returns: ...
#
exampleFun <- function(x,y) {
  ...
}
```

- Comment non-obvious steps

```
#
# to detect outliers we use lrt-method.
# We have tried depth.trim and depth.pond
# but they produce implausible results...
outl <- foutliers(data,method="lrt")
```

- Document your thoughts in your comments

```
  ...
# 18/07/21: although I thought that age should not affect
#           profits, it does here! I also checked
#           xyz-specification and it still does.
#           Perhaps age is a proxy for income.
#           Unfortunately we do not have data on
#           income here.
...
```

- Formatting
  Compare

```
lm ( s1 ~ trust + ineq + sex + age + latitude )
lm ( otherinvestment ~ trust + ineq + sex + age + latitude )
```

  with

```
lm ( s1              ~ trust + ineq + sex + age + latitude )
lm ( otherinvestment ~ trust + ineq + sex + age + latitude )
```

Insert linebreaks Compare

```
lm ( otherinvestment ~ trust + ineq + sex + age + latitude, data=trustExp, subset=sex=="female"
```

with

© Oliver Kirchkamp

```
lm ( otherinvestment ~ trust + ineq + sex + age + latitude,
                       data=trustExp,
                       subset=sex=="female" )
```

- Variables names

  short but not too short

```
lm ( otherinvestment ~ trustworthiness + inequalityaversion + sexOfProposer + ageOfProposer + latit
lm ( otherinvestment ~ trust + ineq + sex + age + latitude)
lm ( oi ~ t + i + s + a + l1 + l2)
lm ( R100234 ~ R100412 + R100017 + R100178 + R100671 + R100229 )
```

We will say more about variable names in section 7.4.

- Abbreviations in scripts

  R (and other languages too) allows you to refer to parameters in functions with names:

```
qnorm(p=.01,lower.tail=FALSE)

[1] 2.326348
```

  To save space, we can abbreviate these names:

```
qnorm(p=.01,low=FALSE)

[1] 2.326348
```

5 Some programming techniques

5.1 Debugging functions

```
library(Ecdat)
data(Kakadu)
str(Kakadu)

'data.frame': 1827 obs. of  22 variables:
 $ lower       : num  0 0 0 0 0 0 0 0 0 0 ...
 $ upper       : num  2 2 2 2 2 2 2 2 2 2 ...
 $ answer      : Ord.factor w/ 3 levels "nn"<"ny"<"yy": 1 1 1 1 1 1 1 1 1 1 ...
 $ recparks    : num  3 5 4 1 2 3 1 5 5 2 ...
 $ jobs        : num  1 5 4 2 4 3 1 3 3 3 ...
 $ lowrisk     : num  5 3 5 4 5 3 5 5 5 3 ...
 $ wildlife    : num  5 5 3 5 3 4 5 5 5 4 ...
 $ future      : num  1 5 5 3 1 5 3 5 4 4 ...
 $ aboriginal  : num  1 1 1 4 3 2 1 2 2 4 ...
```

© Oliver Kirchkamp

```
 $ finben      : num  1 5 5 3 4 3 3 3 3 2 ...
 $ mineparks   : num  4 1 3 3 1 4 1 2 1 2 ...
 $ moreparks   : num  5 5 2 5 1 3 1 3 1 3 ...
 $ gov         : num  1 2 2 1 1 1 1 2 1 1 ...
 $ envcon      : Factor w/ 2 levels "no","yes": 2 1 1 2 1 2 1 1 1 1 ...
 $ vparks      : Factor w/ 2 levels "no","yes": 2 2 2 1 2 2 1 2 2 1 ...
 $ tvenv       : num  1 3 2 1 3 1 3 1 1 3 ...
 $ conservation: Factor w/ 2 levels "no","yes": 1 1 1 2 1 1 1 1 1 1 ...
 $ sex         : Factor w/ 2 levels "female","male": 2 1 2 1 2 2 2 1 2 1 ...
 $ age         : num  27 32 32 70 32 47 42 70 32 47 ...
 $ schooling   : num  3 4 4 6 5 6 5 3 5 2 ...
 $ income      : num  25 9 25 25 35 27 25 25 35 25 ...
 $ major       : Factor w/ 2 levels "no","yes": 1 1 2 1 2 1 2 1 2 1 ...
```

general strategies: debug the function with a simple example

```
sqMean <- function (x) {
   z <- mean(x)
   z^2
}
sqMean(Kakadu$lower)

[1] 2361.471
```

Is this correct? Take a (simpler) subsample of the data:

```
(xx <- sample(Kakadu$lower,3))

[1]  20 100  50

sqMean(xx)

[1] 3211.111
```

Assume that we still do not trust the function. `debug` allows us to debug a function. `ls` allows us to list the variables in the current environment.

```
debug(sqMean)
sqMean(xx)

debugging in: sqMean(xx)
debug at <text>#1: {
    z <- mean(x)
    z^2
}
debug at <text>#2: z <- mean(x)
debug at <text>#3: z^2
exiting from: sqMean(xx)
[1] 3211.111

undebug(sqMean)
```

© Oliver Kirchkamp

If the function returns with an error, it helps to set

```
options(error=recover)
```

In the following function we refer to the variable xxx which is not defined. The function will, hence, fail. With `options(error=recover)` we can inspect the function at the time of the failure.

```
sqMean <- function (x) {
   z <- mean(xxx)
   z^2
}
sqMean(xx)
```

```
Error in mean(xxx) (from #2) : object 'xxx' not found
Enter a frame number, or 0 to exit

1: sqMean(xx)
2: #2: mean(xxx)

Selection: 1
Called from: top level
Browse[1]> xxx
Error during wrapup: object 'xxx' not found
Browse[1]> x
 [1]  20   0   0 250 100  50  20  50  50 100
Browse[1]> Q
```

5.2 Models and lists of variables

To make the analysis more consistent.

Whenever things repeat, we define them in variables at the top of the paper:

```
models <- list(a="income",
               b="income + age + sex",
               c="income + age + sex + conservation + vparks")
```

(We use here character strings to represent parts of formulas. Alternatively, we could also store objects of class `formula`. However, manipulating these objects is not always to obvious. To keep things simple, we will use character strings here.) Later in the paper we compare the different models:

```
mylm <- function (m) lm(paste("as.integer(answer) ~ ",m),data=Kakadu)
lmList<-lapply(models,mylm)
class(lmList)<-c("list","by")
mtable(lmList)
```

© Oliver Kirchkamp

	a	b	c
(Intercept)	2.122***	2.765***	2.648***
	(0.035)	(0.065)	(0.076)
income	0.003*	0.003*	0.002
	(0.001)	(0.001)	(0.001)
age		−0.013***	−0.012***
		(0.001)	(0.001)
sex: male/female		−0.196***	−0.190***
		(0.043)	(0.043)
conservation: yes/no			0.215**
			(0.083)
vparks: yes/no			0.120*
			(0.047)
R-squared	0.003	0.073	0.080
N	1827	1827	1827

Significance: $***\equiv p<0.001$; $**\equiv p<0.01$; $*\equiv p<0.05$

Now we use the same explanatory variables to explain a different dependent variable:

```
mylogit <-function(m) glm(paste("answer=='yy' ~ ",m),
                          data=Kakadu,family=binomial(link=logit))
logitList <- lapply(models,mylogit)
class(logitList)<-c("list","by")
mtable(logitList)
```

	a	b	c
(Intercept)	−0.121	1.100***	0.796***
	(0.078)	(0.155)	(0.181)
income	0.008**	0.009**	0.008*
	(0.003)	(0.003)	(0.003)
age		−0.025***	−0.023***
		(0.003)	(0.003)
sex: male/female		−0.343***	−0.332**
		(0.102)	(0.102)
conservation: yes/no			0.345
			(0.202)
vparks: yes/no			0.334**
			(0.110)
Log-likelihood	−1261.282	−1216.749	−1210.164
N	1827	1827	1827

Significance: $***\equiv p<0.001$; $**\equiv p<0.01$; $*\equiv p<0.05$

Similarly, we might define at the beginning of the paper...

© Oliver Kirchkamp

- lists of random effects
- lists of variables to group by
- themes for plots

5.3 Return values of functions

Most functions do not only return a number (or a vector) but rather complex objects. In R `str()` helps us to learn more about the structure of these objects. (In Stata similar return values are provided by `return`, `ereturn`, and `sreturn`)

```
lm1 <- mylm (models[[1]])
str(lm1)

List of 12
 $ coefficients : Named num [1:2] 2.12202 0.00278
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "income"
 $ residuals    : Named num [1:1827] -1.19 -1.15 -1.19 -1.19 -1.22 ...
  ..- attr(*, "names")= chr [1:1827] "1" "2" "3" "4" ...
 $ effects      : Named num [1:1827] -93.28 1.95 -1.17 -1.17 -1.21 ...
  ..- attr(*, "names")= chr [1:1827] "(Intercept)" "income" "" "" ...
 $ rank         : int 2
 $ fitted.values: Named num [1:1827] 2.19 2.15 2.19 2.19 2.22 ...
  ..- attr(*, "names")= chr [1:1827] "1" "2" "3" "4" ...
 $ assign       : int [1:2] 0 1
 $ qr           :List of 5
  ..$ qr   : num [1:1827, 1:2] -42.7434 0.0234 0.0234 0.0234 0.0234 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:1827] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "(Intercept)" "income"
  .. ..- attr(*, "assign")= int [1:2] 0 1
  ..$ qraux: num [1:2] 1.02 1.02
  ..$ pivot: int [1:2] 1 2
  ..$ tol  : num 0.0000001
  ..$ rank : int 2
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 1825
 $ xlevels      : Named list()
 $ call         : language lm(formula = paste("as.integer(answer) ~ ", m), data = Kakadu)
 $ terms        :Classes 'terms', 'formula'  language as.integer(answer) ~ income
  .. ..- attr(*, "variables")= language list(as.integer(answer), income)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:2] "as.integer(answer)" "income"
  .. .. .. ..$ : chr "income"
  .. ..- attr(*, "term.labels")= chr "income"
  .. ..- attr(*, "order")= int 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: 0x55a107c28d50>
  .. ..- attr(*, "predvars")= language list(as.integer(answer), income)
```

© Oliver Kirchkamp

```
  .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. ..- attr(*, "names")= chr [1:2] "as.integer(answer)" "income"
 $ model        :'data.frame': 1827 obs. of  2 variables:
  ..$ as.integer(answer): int [1:1827] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ income            : num [1:1827] 25 9 25 25 35 27 25 25 35 25 ...
  ..- attr(*, "terms")=Classes 'terms', 'formula'  language as.integer(answer) ~ income
  .. .. ..- attr(*, "variables")= language list(as.integer(answer), income)
  .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "as.integer(answer)" "income"
  .. .. .. .. ..$ : chr "income"
  .. .. ..- attr(*, "term.labels")= chr "income"
  .. .. ..- attr(*, "order")= int 1
  .. .. ..- attr(*, "intercept")= int 1
  .. .. ..- attr(*, "response")= int 1
  .. .. ..- attr(*, ".Environment")=<environment: 0x55a107c28d50>
  .. .. ..- attr(*, "predvars")= language list(as.integer(answer), income)
  .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. .. ..- attr(*, "names")= chr [1:2] "as.integer(answer)" "income"
 - attr(*, "class")= chr "lm"
```

There are at least two ways to extract data from these objects:

- Extractor functions

```
coef(lm1)

(Intercept)      income
2.122018102 0.002781938

vcov(lm1)

               (Intercept)          income
(Intercept)  0.00121806075 -0.000035685812
income      -0.00003568581  0.000001647787

hccm(lm1)

               (Intercept)          income
(Intercept)  0.00123366056 -0.000036812592
income      -0.00003681259  0.000001719666
```

```
logLik(lm1)

'log Lik.' -2402.751 (df=3)

extractAIC(lm1)

[1]    2.0000 -375.2986
```

© Oliver Kirchkamp

```
effects(lm1)
fitted.values(lm1)
residuals(lm1)
```

(the equivalent in Stata are postestimation commands)

- Whatever is a list item can also be accessed directly:

```
lm1$coefficients
lm1$residuals
lm1$fitted.values
lm1$residuals
```

Note: Some interesting values are not provided by the `lm`-object itself. These can often be accessed as part of the `summary`-object.

```
slm1 <- summary(lm1)
slm1$r.squared
slm1$adj.r.squared
slm1$fstatistic
```

5.4 Repeating things

Looping The simplest way to repeat a command is a loop:

```
for (i in 1:10) print(i)

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

If the command is a sequence of expressions, we have to enclose it in braces.

```
for (i in 1:10) {
  x <- runif(i)
  print(mean(x))
}

[1] 0.3565607
[1] 0.9663778
[1] 0.5063639
[1] 0.4378409
```

© Oliver Kirchkamp

```
[1] 0.487012
[1] 0.5853594
[1] 0.3502112
[1] 0.499148
[1] 0.5078825
[1] 0.4557163
```

Avoiding loops In R loops should be avoided. It is more efficient (faster) to apply a function to a vector.

```
sapply(1:10,print)

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

Or, the more complex example:

```
sapply(1:10,function(i) {
  x <- runif(i)
  mean(x)
})

 [1] 0.6538133 0.4623162 0.8092458 0.4935831 0.6997635 0.4856793 0.6413399
 [8] 0.5610393 0.5781580 0.4712342
```

Note that `sapply` already returns a vector which is in many cases what we want anyway.

In the above examples we applied a function to a vector. Sometimes we want to apply functions to a matrix.

Applying a function along one dimension of a matrix In the following example we apply a function along the second dimension of the dataset `Kakadu`.

```
apply(Kakadu,2,function(x) mean(as.integer(x)))

      lower       upper      answer    recparks        jobs     lowrisk
  48.594964  536.714286          NA    3.688560    2.592228    2.790367
   wildlife      future  aboriginal      finben   mineparks   moreparks
    4.739464    4.466886    3.569787    2.915709    3.643678    3.864806
        gov      envcon      vparks       tvenv conservation         sex
    1.083196          NA          NA    1.785441          NA          NA
        age   schooling      income       major
   42.968254    3.683634   21.656814          NA
```

© Oliver Kirchkamp

```
xtable(cbind(mean=apply(Kakadu,2,function(x)
    mean(as.integer(x)))))
```

	mean
lower	48.59
upper	536.71
answer	
recparks	3.69
jobs	2.59
lowrisk	2.79
wildlife	4.74
future	4.47
aboriginal	3.57
finben	2.92
mineparks	3.64
moreparks	3.86
gov	1.08
envcon	
vparks	
tvenv	1.79
conservation	
sex	
age	42.97
schooling	3.68
income	21.66
major	

Rectangular and ragged arrays Rectangular array:

wide

	a	b	c
A	1	2	3
B	4	5	6

long

hor	vert	x
a	A	1
b	A	2
c	A	3
a	B	4
b	B	5
c	B	6

Ragged array:

wide

	a	b	c
A		2	3
B	4	5	

long

hor	vert	x
b	A	2
c	A	3
a	B	4
b	B	5

© Oliver Kirchkamp

Applying a function to each element of a ragged array In R ragged arrays can be represented as datasets grouped by one or more factors. These variables describe which records belong together (e.g. to the same person, year, firm,...)

In the following example we use the dataset `Fatality`. This dataset contains for each state of the United States and for each year in 1982 to 1988 in `mrall` the traffic fatality rate (deaths per 10000).

```
data(Fatality)
head(Fatality)

  state year   mrall  beertax  mlda jaild comserd   vmiles unrate   perinc
1     1 1982 2.12836 1.539379 19.00    no      no 7.233887   14.4 10544.15
2     1 1983 2.34848 1.788991 19.00    no      no 7.836348   13.7 10732.80
3     1 1984 2.33643 1.714286 19.00    no      no 8.262990   11.1 11108.79
4     1 1985 2.19348 1.652542 19.67    no      no 8.726917    8.9 11332.63
 [ reached 'max' / getOption("max.print") -- omitted 2 rows ]
```

```
by(Fatality,list(Fatality$year),function(x) mean(x$mrall))

: 1982
[1] 2.089106
------------------------------------------------------------
: 1983
[1] 2.007846
------------------------------------------------------------
: 1984
[1] 2.017122
------------------------------------------------------------
: 1985
[1] 1.973671
------------------------------------------------------------
: 1986
[1] 2.065071
------------------------------------------------------------
: 1987
[1] 2.060696
------------------------------------------------------------
: 1988
[1] 2.069594
```

```
by(Fatality,list(Fatality$state),function(x) mean(x$mrall))

: 1
[1] 2.412627
------------------------------------------------------------
: 4
[1] 2.7059
------------------------------------------------------------
: 5
```

© Oliver Kirchkamp

```
[1] 2.435336
------------------------------------------------------------
: 6
[1] 1.904977
------------------------------------------------------------
: 8
[1] 1.866981
------------------------------------------------------------
: 9
[1] 1.463509
------------------------------------------------------------
: 10
[1] 2.068231
------------------------------------------------------------
: 12
[1] 2.477799
------------------------------------------------------------
: 13
[1] 2.401569
------------------------------------------------------------
: 16
[1] 2.571667
------------------------------------------------------------
: 17
[1] 1.405084
------------------------------------------------------------
: 18
[1] 1.834221
------------------------------------------------------------
: 19
[1] 1.679544
------------------------------------------------------------
: 20
[1] 1.969664
------------------------------------------------------------
: 21
[1] 2.133043
------------------------------------------------------------
: 22
[1] 2.120829
------------------------------------------------------------
: 23
[1] 1.87013
------------------------------------------------------------
: 24
[1] 1.629377
------------------------------------------------------------
: 25
[1] 1.199393
------------------------------------------------------------
: 26
[1] 1.672087
```

© Oliver Kirchkamp

```
------------------------------------------------------------
: 27
[1] 1.370441
------------------------------------------------------------
: 28
[1] 2.761846
------------------------------------------------------------
: 29
[1] 1.977451
------------------------------------------------------------
: 30
[1] 2.903021
------------------------------------------------------------
: 31
[1] 1.685413
------------------------------------------------------------
: 32
[1] 2.74526
------------------------------------------------------------
: 33
[1] 1.798824
------------------------------------------------------------
: 34
[1] 1.319227
------------------------------------------------------------
: 35
[1] 3.653197
------------------------------------------------------------
: 36
[1] 1.207581
------------------------------------------------------------
: 37
[1] 2.34471
------------------------------------------------------------
: 38
[1] 1.601454
------------------------------------------------------------
: 39
[1] 1.550474
------------------------------------------------------------
: 40
[1] 2.33993
------------------------------------------------------------
: 41
[1] 2.177147
------------------------------------------------------------
: 42
[1] 1.541673
------------------------------------------------------------
: 44
[1] 1.110077
------------------------------------------------------------
```

© Oliver Kirchkamp

```
: 45
[1] 2.821669
------------------------------------------------------------
: 46
[1] 2.04929
------------------------------------------------------------
: 47
[1] 2.403066
------------------------------------------------------------
: 48
[1] 2.27587
------------------------------------------------------------
: 49
[1] 1.835836
------------------------------------------------------------
: 50
[1] 2.092991
------------------------------------------------------------
: 51
[1] 1.740946
------------------------------------------------------------
: 53
[1] 1.677211
------------------------------------------------------------
: 54
[1] 2.300624
------------------------------------------------------------
: 55
[1] 1.616567
------------------------------------------------------------
: 56
[1] 3.217534
```

by does not return a vector but an object of class by. If we actually need a vector we have to use c and sapply.

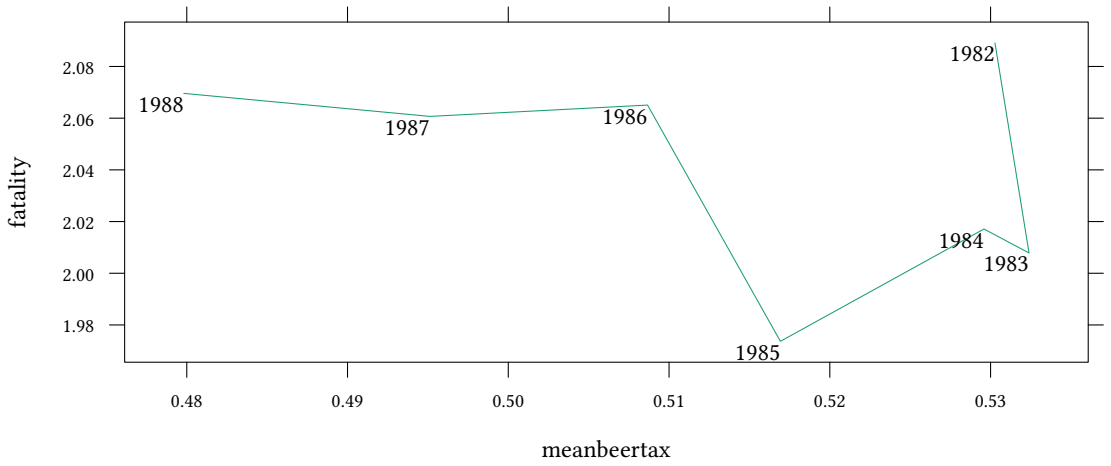
In the following example we let by actually return two values.

```
byObj <- by(Fatality,list(Fatality$year),
            function(x) c(year=median(x$year),
                          fatality=mean(x$mrall),
                          meanbeertax=mean(x$beertax)))
sapply(byObj,c)
```

```
                  1982        1983         1984         1985         1986
year         1982.0000000 1983.000000 1984.0000000 1985.0000000 1986.0000000
fatality        2.0891059    2.007846    2.0171225    1.9736708    2.0650710
meanbeertax     0.5302734    0.532393    0.5295902    0.5169272    0.5086639
                  1987         1988
year         1987.0000000 1988.0000000
fatality        2.0606956    2.0695941
meanbeertax     0.4951288    0.4798154
```

© Oliver Kirchkamp

```
xx<-data.frame(t(sapply(byObj,c)))
xyplot(fatality ~ meanbeertax,type="l",data=xx)+
    layer(with(xx,panel.text(label=year,y=fatality,x=meanbeertax,adj=c(1,1))))
```



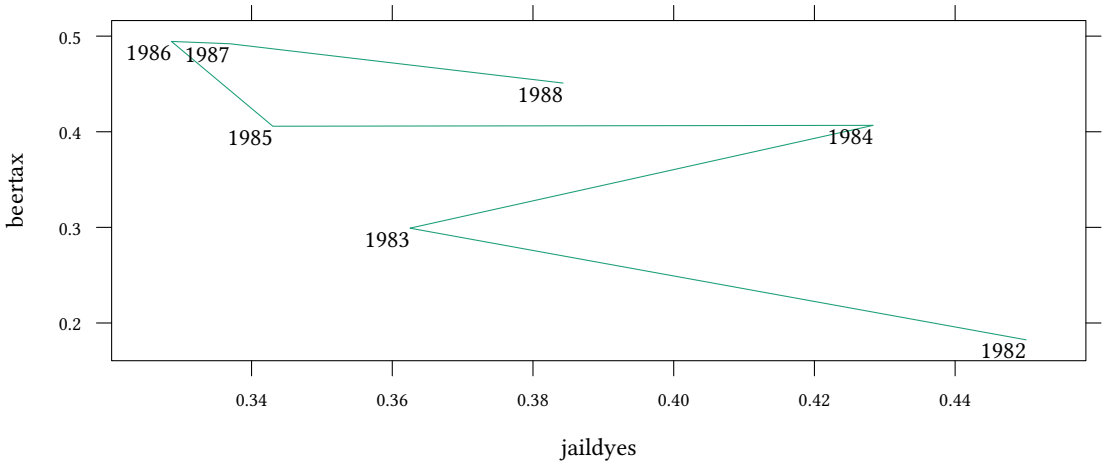
We can do more complicated things in by. In the following example we estimate a regression. To get only the coefficients from the regression (and not fitted values, residuals, etc.) we use the extractor function coef.

```
byObj <- by(Fatality,list(Fatality$year),function(x)
            lm(mrall ~ beertax + jaild, data=x))
sapply(byObj,coef)

                1982      1983      1984      1985      1986      1987
(Intercept) 1.9079924 1.7503870 1.6768093 1.6567128 1.7108657 1.7188081
beertax     0.1824028 0.2991742 0.4066922 0.4057889 0.4944595 0.4920275
jaildyes    0.4500807 0.3625151 0.4283417 0.3430232 0.3286131 0.3369277
                1988
(Intercept) 1.7411593
beertax     0.4509099
jaildyes    0.3842788
```

```
xx<-data.frame(t(sapply(byObj,coef)))
xyplot(beertax ~ jaildyes,type="l",data=xx)+
    layer(with(xx,panel.text(label=rownames(xx),y=beertax,x=jaildyes,adj=c(1,1))))
```

© Oliver Kirchkamp



by is very complex. It offers the entire subset of the *dataframe*, as defined by the index variable, to the function.

Sometimes we want simply to apply a function of only a *vector* along a ragged array.

```
with(Fatality,aggregate(mrall~year,FUN=mean))

  year    mrall
1 1982 2.089106
2 1983 2.007846
3 1984 2.017122
4 1985 1.973671
5 1986 2.065071
6 1987 2.060696
7 1988 2.069594
```

Again, the function (which was `mean` in the previous example) can be defined by us:

```
with(Fatality,aggregate(mrall~year,FUN=function(x) sd(x)/mean(x)))

  year     mrall
1 1982 0.3196449
2 1983 0.3017002
3 1984 0.2721300
4 1985 0.2726437
5 1986 0.2709500
6 1987 0.2738153
7 1988 0.2518286
```

© Oliver Kirchkamp

6 Data manipulation

6.1 Subsetting data

There are several ways to access only a part of a dataset:

- Many functions take an option `...,subset=...`

```
lm(mrall ~ beertax + jaild, data=Fatality, subset = year == 1982)


Call:
lm(formula = mrall ~ beertax + jaild, data = Fatality, subset = year ==
    1982)

Coefficients:
(Intercept)      beertax     jaildyes
     1.9080       0.1824       0.4501
```

- The `subset()` function

```
subset(Fatality, year == 1982 )

   state year   mrall   beertax mlda jaild comserd   vmiles unrate   perinc
1      1 1982 2.12836 1.5393795   19    no      no 7.233887   14.4 10544.15
8      4 1982 2.49914 0.2147971   19   yes     yes 6.810157    9.9 12309.07
15     5 1982 2.38405 0.6503580   21    no      no 7.208500    9.8 10267.30
22     6 1982 1.86194 0.1073986   21    no      no 6.858677    9.9 15797.14
 [ reached 'max' / getOption("max.print") -- omitted 44 rows ]

with(subset(Fatality, year == 1982 ), lm(mrall ~ beertax + jaild))


Call:
lm(formula = mrall ~ beertax + jaild)

Coefficients:
(Intercept)      beertax     jaildyes
     1.9080       0.1824       0.4501
```

- The first index of the dataset

```
Fatality[ Fatality$year==1982 , ]

   state year   mrall   beertax mlda jaild comserd   vmiles unrate   perinc
1      1 1982 2.12836 1.5393795   19    no      no 7.233887   14.4 10544.15
8      4 1982 2.49914 0.2147971   19   yes     yes 6.810157    9.9 12309.07
15     5 1982 2.38405 0.6503580   21    no      no 7.208500    9.8 10267.30
22     6 1982 1.86194 0.1073986   21    no      no 6.858677    9.9 15797.14
 [ reached 'max' / getOption("max.print") -- omitted 44 rows ]
```

© Oliver Kirchkamp

```
with(Fatality[ Fatality$year==1982 , ],lm(mrall ~ beertax + jaild))


Call:
lm(formula = mrall ~ beertax + jaild)

Coefficients:
(Intercept)      beertax     jaildyes
     1.9080       0.1824       0.4501
```

6.2 Merging data

- Appending two datasets

```
library(dplyr)
bind_rows(x,y)
```

(In Stata this is done by append)

- Matching two datasets (inner join)

```
merge(x,y)
```

(In Stata this is done by merge)

- Joining two datasets (left join)

```
merge(x,y,all.x=TRUE)
```

(In Stata this is done by joinby)

Dataset A

Name	Grade
Eva	2.0
Mary	1.0
Mike	3.0

Dataset B

Name	eMail
Eva	eva@...
Eva	eva2@...
Susan	susan@...
Mike	mike@...

Inner join: `merge(A,B)`

Name	Grade	eMail
Eva	2.0	eva@...
Eva	2.0	eva2@...
Mike	3.0	mike@...

Left join: `merge(A,B,all.x=TRUE)`

Name	Grade	eMail
Eva	2.0	eva@...
Eva	2.0	eva2@...
Mary	1.0	NA
Mike	3.0	mike@...

© Oliver Kirchkamp

Appending In the following example we first split the data from an experiment into two parts. Merge helps us to append them to each other.

```
load("data/180716_060x.Rdata")
experiment1 <- subset(trustGS$subjects,Date=="180716_0601")
experiment2 <- subset(trustGS$subjects,Date=="180716_0602")
dim(experiment1)

[1] 108  14

dim(experiment2)

[1] 108  14

library(dplyr)
dim(bind_rows(experiment1,experiment2))

[1] 216  14
```

Joining A frequent application for a join are tables in z-Tree that have something to do with each other. E.g. the globals and the subjects tables both provide information about each period. Common variables in these tables are `Date`, `Treatment`, and `Period`.

By merging globals with subjects, `merge` looks up for each record in the subjects table the matching record in the globals table and adds the variables which are not already present in subjects.

```
head(trustGS$global)

         Date Treatment Period NumPeriods RepeatTreatment
1 180716_0601         1      1          6               0
2 180716_0601         1      2          6               0
3 180716_0601         1      3          6               0
4 180716_0601         1      4          6               0
5 180716_0601         1      5          6               0
6 180716_0601         1      6          6               0

head(trustGS$subject)

         Date Treatment Period Subject Pos Group Offer Receive   Return GetBack
1 180716_0601         1      1       1   2     1     0   1.530 0.585990       0
2 180716_0601         1      1       2   2     4     0   1.674 1.131624       0
  country siblings sex age
1       6        1   1  27
2      15        3   1  19
 [ reached 'max' / getOption("max.print") -- omitted 4 rows ]
```

In the following example we simply get two more variables in the dataset (`NumPeriods` and `RepeatTreatment`). With more variables in globals we would, of course, also get more variables in the merged dataset.

© Oliver Kirchkamp

```
dim(trustGS$global)

[1] 24  5

dim(trustGS$subject)

[1] 432  14

dim(merge(trustGS$global,trustGS$subject))

[1] 432  16
```

Joining aggregates A common application for a join is a comparison of our individual data with aggregated data. Let us come back to the Fatalities example. We want to compare the traffic fatility rate `mrall` for each state with the average values for each year.

```
head(Fatality)

  state year   mrall  beertax  mlda jaild comserd   vmiles unrate   perinc
1     1 1982 2.12836 1.539379 19.00    no      no 7.233887   14.4 10544.15
2     1 1983 2.34848 1.788991 19.00    no      no 7.836348   13.7 10732.80
3     1 1984 2.33643 1.714286 19.00    no      no 8.262990   11.1 11108.79
4     1 1985 2.19348 1.652542 19.67    no      no 8.726917    8.9 11332.63
 [ reached 'max' / getOption("max.print") -- omitted 2 rows ]

aggregate(cbind(avgMrall=mrall) ~ year,data=Fatality,FUN=mean)

  year avgMrall
1 1982 2.089106
2 1983 2.007846
3 1984 2.017122
4 1985 1.973671
5 1986 2.065071
6 1987 2.060696
7 1988 2.069594
```

```
merge(Fatality,aggregate(cbind(avgMrall=mrall) ~ year,data=Fatality,FUN=mean))

  year state   mrall   beertax mlda jaild comserd   vmiles unrate   perinc
1 1982     1 2.12836 1.5393795   19    no      no 7.233887   14.4 10544.15
2 1982    30 3.15528 0.3464475   19   yes      no 8.284474    8.6 12033.41
3 1982    10 2.03333 0.1730310   20    no      no 7.651654    8.5 14263.72
  avgMrall
1 2.089106
2 2.089106
3 2.089106
 [ reached 'max' / getOption("max.print") -- omitted 333 rows ]
```

`merge` has joined the two datasets, the large Fatality one, and the small aggregated one, on the variable `year`.

© Oliver Kirchkamp

6.3 Reshaping data

Sometimes we have different observations of the same (or similar) variable in the same row (e.g. `profit.1` and `profit.2`), sometimes we have them stacked in one column (e.g. as `profit`). We call the first format *wide*, the second *long*.

For the *long* case we need a variable that distinguishes the different instances of this variable (`profit.1` and `profit.2`) from each other. In R such a variable is called `timevar` (Stata calls them `j`).

We also need one or more variables that tells us, which observations actually belonged to one row in the *wide* format. In R we call these variables `idvar` (Stata call these variables `i`).

Let us look at a part of our trust dataset

```
trustLong <- trustGS$subjects[,c("Date","Period","Subject","Pos",
                                 "Group","Offer")]
trustLong[1:4,]
```

```
         Date Period Subject Pos Group Offer
1 180716_0601      1       1   2     1 0.000
2 180716_0601      1       2   2     4 0.000
3 180716_0601      1       3   1     5 0.495
4 180716_0601      1       4   2     2 0.000
```

```
trustWide <- reshape(trustLong,v.names=c("Offer","Subject"),
                     idvar=c("Date","Period","Group"),timevar="Pos",
                     direction="wide")
trustWide[1:4,]
```

```
         Date Period Group Offer.2 Subject.2   Offer.1 Subject.1
1 180716_0601      1     1       0         1 0.5100000        13
2 180716_0601      1     4       0         2 0.5580000         5
3 180716_0601      1     5       0         7 0.4950000         3
4 180716_0601      1     2       0         4 0.8422333         8
```

```
reshape(trustWide,direction="long")[1:4,]
```

```
                          Date Period Group Pos Offer Subject
180716_0601.1.1.2 180716_0601      1     1   2     0       1
180716_0601.1.4.2 180716_0601      1     4   2     0       2
180716_0601.1.5.2 180716_0601      1     5   2     0       7
180716_0601.1.2.2 180716_0601      1     2   2     0       4
```

↑ Reshaping back returns more or less the orignal data. The ordering has changed and rows have got names now.

```
library(reshape2)
recast( trustLong, Date + Period + Group ~ Pos, measure.var=c("Offer"))
```

© Oliver Kirchkamp

```
         Date Period Group         1 2
1 180716_0601      1     1 0.5100000 0
2 180716_0601      1     2 0.8422333 0
3 180716_0601      1     3 0.7510000 0
4 180716_0601      1     4 0.5580000 0
5 180716_0601      1     5 0.4950000 0
6 180716_0601      1     6 0.6910000 0
7 180716_0601      1     7 0.5430000 0
8 180716_0601      1     8 0.3660000 0
 [ reached 'max' / getOption("max.print") -- omitted 208 rows ]
```

Reshaping with reshape2 `recast` does not give us `Subject`, though.

6.4 More on functions

6.4.1 Functional programming

Consider the following dataframe:

```
wide <- reshape(Indometh, v.names = "conc", idvar = "Subject",
                timevar = "time", direction = "wide")
wide

   Subject conc.0.25 conc.0.5 conc.0.75 conc.1 conc.1.25 conc.2 conc.3 conc.4
1        1      1.50     0.94      0.78   0.48      0.37   0.19   0.12   0.11
12       2      2.03     1.63      0.71   0.70      0.64   0.36   0.32   0.20
23       3      2.72     1.49      1.16   0.80      0.80   0.39   0.22   0.12
   conc.5 conc.6 conc.8
1    0.08   0.07   0.05
12   0.25   0.12   0.08
23   0.11   0.08   0.08
 [ reached 'max' / getOption("max.print") -- omitted 3 rows ]
```

Now assume that you consider all values of `conc>1` invalid and you want to replace them with `NA`

```
within(wide,{
    conc.0.25[conc.0.25>1]<-NA
    conc.0.5[conc.0.5>1]<-NA
    conc.0.75[conc.0.75>1]<-NA
    ...
}
```

This is clumsy and error prone. Instead:

```
varnames <- grep("conc",names(wide))
cbind(wide[-varnames],data.frame(lapply(wide[,varnames],
   function(x) {x[x>1]<-NA;x})))
```

© Oliver Kirchkamp

```
   Subject conc.0.25 conc.0.5 conc.0.75 conc.1 conc.1.25 conc.2 conc.3 conc.4
1        1        NA     0.94      0.78   0.48      0.37   0.19   0.12   0.11
12       2        NA       NA      0.71   0.70      0.64   0.36   0.32   0.20
23       3        NA       NA        NA   0.80      0.80   0.39   0.22   0.12
   conc.5 conc.6 conc.8
1    0.08   0.07   0.05
12   0.25   0.12   0.08
23   0.11   0.08   0.08
 [ reached 'max' / getOption("max.print") -- omitted 3 rows ]
```

6.4.2 Closures

```
power <- function(exponent)
    function(x) x^exponent
power(2)

function(x) x^exponent
<environment: 0x55a10a64cd48>

square <- power(2)
sqroot <- power(1/2)
sqroot(16)

[1] 4

square(16)

[1] 256

as.list(environment(sqroot))

$exponent
[1] 0.5

as.list(environment(square))

$exponent
[1] 2
```

Functions keep the environment under which they are created. (So here they remember the exponent).

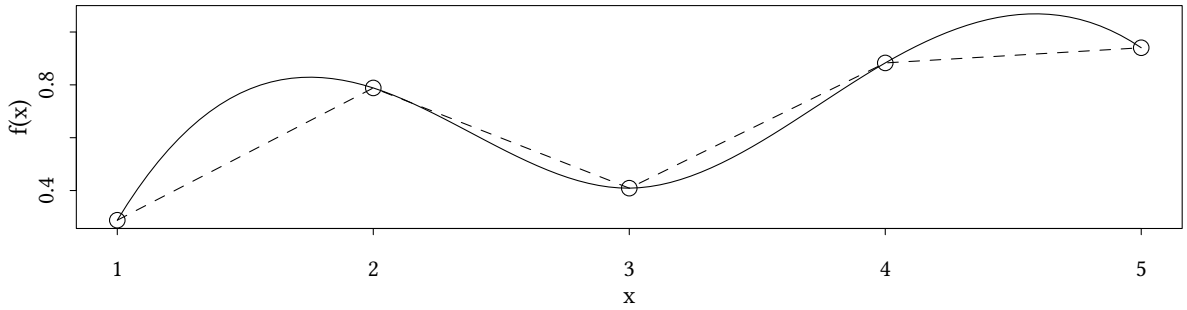
Here is an application of closures:

```
set.seed(123)
x<-1:5
y<-runif(5)
f<-splinefun(x,y)
f2<-approxfun(x,y)
curve(f,from=1,to=5)
```

© Oliver Kirchkamp

```
curve(f2,add=TRUE,lty=2)
points(x,y)
f(2.5)

[1] 0.5585953
```



6.4.3 Chaining functions

Sometimes you want to apply functions of functions:

```
x <- 1:10
var(x)

[1] 9.166667

sqrt(var(x))

[1] 3.02765
```

So far this is trivial. Here is a more complicated example:
A deeply nested function can be hard to understand:

```
library(plyr)
ddply(subset(mtcars,!is.na(am) & !is.na(cyl)),.(am,cyl),
         summarize,disp=mean(disp),hp=mean(hp))

  am cyl     disp        hp
1  0   4 135.8667  84.66667
2  0   6 204.5500 115.25000
3  0   8 357.6167 194.16667
4  1   4  93.6125  81.87500
5  1   6 155.0000 131.66667
6  1   8 326.0000 299.50000
```

We could store intermediate results in a variable (xx)

```
xx<-subset(mtcars,!is.na(am) & !is.na(cyl))
ddply(xx,.(am,cyl),summarize,disp=mean(disp),hp=mean(hp))
```

© Oliver Kirchkamp

```
  am cyl     disp        hp
1  0   4 135.8667  84.66667
2  0   6 204.5500 115.25000
3  0   8 357.6167 194.16667
4  1   4  93.6125  81.87500
5  1   6 155.0000 131.66667
6  1   8 326.0000 299.50000
```

We could combine all this into a single chain of functions:
The %>% operator from from dplyr allows us to chain functions more transparently (we have to unload plyr first, since both define functions with identical names).

```
library(dplyr)
mtcars %>%
  filter(!is.na(am), !is.na(cyl))  %>%
  group_by(am,cyl) %>%
  summarise(disp=mean(disp),hp=mean(hp))

      disp       hp
1 230.7219 146.6875
```

7 Preparing Data

- read data
- check structure (names, dimension, labels)
- check values
- create new data:
  - recode variables
  - rename variables
  - label variables
  - eliminate outliers
  - reshape data

7.1 Preserve raw data

- If our raw data is software generated (e.g. from our experiments): We better keep *all* programs in the working directory).
- If our raw data includes data from a questionnaire:
  - We need a codebook
    - variable name — question number — text of the questions

© Oliver Kirchkamp

* branching in the questionnaire
* levels (value labels) used for factors
* missing data, how was it coded?
* cleaned data, how was it cleaned? (if we have no access to the raw data)

7.2 Reading data

7.2.1 Reading z-Tree Output

The function
`zTreeTables(...vector of filenames...[,vector of tables])`
reads zTree `.xls` files and returns a list of tables. Here we use `list.files` to find all files that match the typical z-Tree pattern. If we ever get more experiments our command will find them and use them.

```
library(foreign)
library(readstata13)
```

```
library("zTree")
```

```
setwd("data/rawdata/Trust")
files <- list.files(pattern = "[0-9]{6}_[0-9]{4}.xls$",recursive=TRUE)
files

[1] "180716_0601.xls" "180716_0602.xls" "180716_0603.xls" "180716_0604.xls"
```

```
trustGS <- zTreeTables(files)

reading  180716_0601.xls ...
Skipping:
Doing: globals
Doing: subjects
*** 180716_0602.xls is file 2 / 4 ***
reading  180716_0602.xls ...
Skipping:
Doing: globals
Doing: subjects
*** 180716_0603.xls is file 3 / 4 ***
reading  180716_0603.xls ...
Skipping:
Doing: globals
Doing: subjects
*** 180716_0604.xls is file 4 / 4 ***
reading  180716_0604.xls ...
Skipping:
Doing: globals
Doing: subjects
```

© Oliver Kirchkamp

save in R-format:

```
save(trustGS,zTreeTables,file="180716_060x.Rdata")
```

save in Stata-format:

```
xx<-with(trustGS,merge(globals,subjects))
write.dta(xx,file="180716_060x.dta")
```

save in Stata-13 format:

```
save.dta13(xx,file="180716_060x.dta13")
```

save as csv:

```
write.csv(xx,file="180716_060x.csv")
```

```
fn<-list.files(pattern="180716_060x\\.[^.]*")
xtable(cbind(name=fn,size=file.size(fn)))
```

	name	size
1	180716_060x.csv	28178
2	180716_060x.dta	59300
3	180716_060x.dta13	60354
4	180716_060x.Rdata	17356

As long as we need only a single table, we can access, e.g. the subjects table with `$subjects`.

If we need, e.g. the `globals` table together with the `subjects` table, we can merge:

```
with(trustGS,merge(globals,subjects))
```

7.2.2 Reading and writing R-Files

If we want to save one or more R objects in a file, we use `save`

```
save(trustGS,zTreeTables,file="data/180716_060x.Rdata")
```

To retrieve them, we use `load`

```
load("data/180716_060x.Rdata")
```

Advantages:

- Rdata is very compact, files are small.
- All attributes are saved together with the data.
- We can save functions together with data.

© Oliver Kirchkamp

7.2.3 Reading Stata Files

package	command	limitation	generates	attributes
`foreign`	`read.dta`	Stata version 5-12	data.frame	data.frame
	`write.dta`	Stata version 5-12		
`memisc`	`Stata.file`	Stata version 5-12	Data set	variable
`readstata13`	`read.dta13`	Stata version 13+	data.frame	data.frame
`haven`	`read_dta`	Stata version 8+	tibble	variable

```
library(foreign)
sta <- read.dta("data/180716_060x.dta")
```

```
sta2 <- Stata.file("data/180716_060x.dta")
```

Stata attributes (formats,value labels,variable labels) are stored either with data.frame (`foreign` and `readstata13`) or variables.

```
str(sta)
```

```
'data.frame': 432 obs. of  16 variables:
 $ Date           : chr  "180716_0601" "180716_0601" "180716_0601" "180716_0601" ...
 $ Treatment      : num  1 1 1 1 1 1 1 1 1 ...
 $ Period         : num  1 1 1 1 1 1 1 1 1 ...
 $ NumPeriods     : num  6 6 6 6 6 6 6 6 6 ...
 $ RepeatTreatment: num  0 0 0 0 0 0 0 0 0 ...
 $ Subject        : num  1 2 3 4 5 6 7 8 9 10 ...
 $ Pos            : num  2 2 1 2 1 1 2 1 2 2 ...
 $ Group          : num  1 4 5 2 4 3 5 2 9 7 ...
 $ Offer          : num  0 0 0.495 0 0.558 ...
 $ Receive        : num  1.53 1.67 0 2.53 0 ...
 $ Return         : num  0.586 1.132 0 1.471 0 ...
 $ GetBack        : num  0 0 0.425 0 1.132 ...
 $ country        : num  6 15 8 16 17 1 18 12 7 98 ...
 $ siblings       : num  1 3 3 3 0 0 3 1 2 3 ...
 $ sex            : num  1 1 1 99 1 2 2 2 2 2 ...
 $ age            : num  27 19 18 28 30 21 25 17 20 99 ...
 - attr(*, "datalabel")= chr "Written by R.              "
 - attr(*, "time.stamp")= chr ""
 - attr(*, "formats")= chr [1:16] "%11s" "%9.0g" "%9.0g" "%9.0g" ...
 - attr(*, "types")= int [1:16] 138 100 100 100 100 100 100 100 100 100 ...
 - attr(*, "val.labels")= chr [1:16] "" "" "" "" ...
 - attr(*, "var.labels")= chr [1:16] "Date" "Treatment" "Period" "NumPeriods" ...
 - attr(*, "version")= int 7
```

The data frame created by `Stata.file` looks different:

© Oliver Kirchkamp

```
str(sta2)

Formal class 'Stata.importer' [package "memisc"] with 6 slots
  ..@ .Data    :List of 16
  .. ..$ : Nmnl. item  chr(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  .. ..$ : Itvl. item  num(0)
  ..@ data.spec:List of 9
  .. ..$ names         : chr [1:16] "Date" "Treatment" "Period" "NumPeriods" ...
  .. ..$ types         : Named raw [1:16] 0b ff ff ff ...
  .. .. ..- attr(*, "names")= chr [1:16] "Date" "Treatment" "Period" "NumPeriods" ...
  .. ..$ nobs          : int 432
  .. ..$ nvar          : int 16
  .. ..$ varlabs       : Named chr [1:16] "Date" "Treatment" "Period" "NumPeriods" ...
  .. .. ..- attr(*, "names")= chr [1:16] "Date" "Treatment" "Period" "NumPeriods" ...
  .. ..$ value.labels  : Named chr(0)
  .. .. ..- attr(*, "names")= chr(0)
  .. ..$ missing.values: NULL
  .. ..$ missval_labels: NULL
  .. ..$ version.string: chr "Stata 7"
  ..@ ptr      :<externalptr>
  .. ..- attr(*, "file.name")= chr "data/180716_060x.dta"
  ..@ document : chr(0)
  ..@ encoded  : chr "cp1252"
  ..@ names    : chr [1:16] "Date" "Treatment" "Period" "NumPeriods" ...
```

Also the attributes are different:

```
attributes(sta)

$datalabel
[1] "Written by R.              "

$time.stamp
[1] ""

$names
 [1] "Date"            "Treatment"       "Period"          "NumPeriods"
 [5] "RepeatTreatment" "Subject"         "Pos"             "Group"
```

© Oliver Kirchkamp

```
 [9] "Offer"           "Receive"         "Return"          "GetBack"
[13] "country"         "siblings"        "sex"             "age"

$formats
 [1] "%11s"  "%9.0g" "%9.0g" "%9.0g" "%9.0g" "%9.0g" "%9.0g" "%9.0g" "%9.0g"
[10] "%9.0g" "%9.0g" "%9.0g" "%9.0g" "%9.0g" "%9.0g" "%9.0g"

$types
 [1] 138 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100

$val.labels
 [1] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""

$var.labels
 [1] "Date"            "Treatment"       "Period"          "NumPeriods"
 [5] "RepeatTreatment" "Subject"         "Pos"             "Group"
 [9] "Offer"           "Receive"         "Return"          "GetBack"
[13] "country"         "siblings"        "sex"             "age"

$row.names
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "15"
[16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29" "30"
[31] "31" "32" "33" "34" "35" "36" "37" "38" "39" "40"
 [ reached getOption("max.print") -- omitted 392 entries ]

$version
[1] 7

$class
[1] "data.frame"
```

Stata.file stores variable labels as attributes of the variables:

```
attributes(sta2)

$ptr
<pointer: 0x561bd06bc500>
attr(,"file.name")
[1] "data/180716_060x.dta"

$document
character(0)

$encoded
[1] "cp1252"

$names
 [1] "Date"            "Treatment"       "Period"          "NumPeriods"
 [5] "RepeatTreatment" "Subject"         "Pos"             "Group"
 [9] "Offer"           "Receive"         "Return"          "GetBack"
[13] "country"         "siblings"        "sex"             "age"
```

© Oliver Kirchkamp

```
$data.spec
$data.spec$names
 [1] "Date"            "Treatment"       "Period"          "NumPeriods"
 [5] "RepeatTreatment" "Subject"         "Pos"             "Group"
 [9] "Offer"           "Receive"         "Return"          "GetBack"
[13] "country"         "siblings"        "sex"             "age"

$data.spec$types
           Date       Treatment          Period      NumPeriods RepeatTreatment
             0b              ff              ff              ff              ff
        Subject             Pos           Group           Offer         Receive
             ff              ff              ff              ff              ff
         Return         GetBack         country        siblings             sex
             ff              ff              ff              ff              ff
            age
             ff

$data.spec$nobs
[1] 432

$data.spec$nvar
[1] 16

$data.spec$varlabs
            Date         Treatment            Period        NumPeriods
          "Date"       "Treatment"          "Period"      "NumPeriods"
  RepeatTreatment           Subject               Pos             Group
"RepeatTreatment"         "Subject"             "Pos"           "Group"
           Offer           Receive            Return           GetBack
         "Offer"         "Receive"          "Return"         "GetBack"
         country          siblings               sex              age
       "country"        "siblings"             "sex"            "age"

$data.spec$value.labels
named character(0)

$data.spec$missing.values
NULL

$data.spec$missval_labels
NULL

$data.spec$version.string
[1] "Stata 7"


$class
[1] "Stata.importer"
attr(,"package")
[1] "memisc"
```

Within the `memisc` world you can obtain more information with `codebook`.

© Oliver Kirchkamp

```
codebook(sta2)
```

```
================================================================================

   Date 'Date'

--------------------------------------------------------------------------------

   Storage mode: character
   Measurement: nominal


================================================================================

   Treatment 'Treatment'

--------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min: 1.000
        Max: 1.000
       Mean: 1.000
   Std.Dev.: 0.000

================================================================================

   Period 'Period'

--------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min: 1.000
        Max: 6.000
       Mean: 3.500
   Std.Dev.: 1.708

================================================================================

   NumPeriods 'NumPeriods'

--------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min: 6.000
        Max: 6.000
       Mean: 6.000
```

© Oliver Kirchkamp

```
   Std.Dev.: 0.000

================================================================================

   RepeatTreatment 'RepeatTreatment'

--------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min: 0.000
        Max: 0.000
       Mean: 0.000
   Std.Dev.: 0.000

================================================================================

   Subject 'Subject'

--------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min:  1.000
        Max: 18.000
       Mean:  9.500
   Std.Dev.:  5.188

================================================================================

   Pos 'Pos'

--------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min: 1.000
        Max: 2.000
       Mean: 1.500
   Std.Dev.: 0.500

================================================================================

   Group 'Group'

--------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval
```

© Oliver Kirchkamp

```
        Min: 1.000
        Max: 9.000
       Mean: 5.000
   Std.Dev.: 2.582

------------------------------------------------------------------------------

   Offer 'Offer'

------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min: 0.000
        Max: 1.000
       Mean: 0.327
   Std.Dev.: 0.370

------------------------------------------------------------------------------

   Receive 'Receive'

------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min: 0.000
        Max: 3.000
       Mean: 0.981
   Std.Dev.: 1.109

------------------------------------------------------------------------------

   Return 'Return'

------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min: 0.000
        Max: 2.763
       Mean: 0.409
   Std.Dev.: 0.617

------------------------------------------------------------------------------

   GetBack 'GetBack'
```

© Oliver Kirchkamp

```
-------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min: 0.000
        Max: 2.763
       Mean: 0.409
   Std.Dev.: 0.617

===============================================================================

   country 'country'

-------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min:  1.000
        Max: 99.000
       Mean: 18.069
   Std.Dev.: 26.863

===============================================================================

   siblings 'siblings'

-------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min:  0.000
        Max: 99.000
       Mean:  2.903
   Std.Dev.: 11.464

===============================================================================

   sex 'sex'

-------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min:  1.000
        Max: 99.000
       Mean: 10.986
   Std.Dev.: 28.747
```

© Oliver Kirchkamp

```
================================================================================

   age 'age'

--------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min: 16.000
        Max: 99.000
       Mean: 32.694
   Std.Dev.: 23.778
```

The `memisc` approach preserves more information. Often this is more intuitive. Some packages are, however, confused by these attributes.

Stata 13 Every now and then stata changes their file format:

```
library(readstata13)
sta13<-read.dta13("data/180716_060x.dta13")
```

7.2.4 Reading CSV Files

CSV-Files (Comma-Separated-Value) Files are in no way always *comma* separated. The term is rather used to denote any table with a constant separator. Some of the parameters that always change are:

- Separators: , ; TAB
- Quoting of strings: ” ’ –
- Headers: with / without

As a result, the `read.table` has many parameters.

```
csv <- read.csv("data/180716_060x.csv",sep="\t")
str(csv)
```

The advantage of CSV as a medium to exchange data is: CSV can be read by any software.
The disadvantage is: No extra information (variable labels, levels of factors, ...) can be stored.

7.2.5 Reading Microsoft Excel files before 2007 (xls)

```
library(readxl)
read_excel(path,sheet)
```

© Oliver Kirchkamp

Sometimes the `xls` file is not really a data frame but has to be parsed before one can translate it into a data frame. You might find the following approach helpful if records contain an unequal number of entries.

First extract all the lines...

```
file<-"data/180716_0601.xls"
system(paste("ssconvert  --export-type Gnumeric_stf:stf_assistant -O 'separator=\"\t\"'",
             file,"tmp.csv"))
aa<-readLines("tmp.csv")
```

Determine the number of entries for each record. Here we subset only records with the same number of entries as the previous to the last one:

```
aa2l<-unlist(lapply(strsplit(aa,"\t"),length))
xx<-ldply(strsplit(aa[aa2l==aa2l[length(aa2l)-1]],split="\t"))
```

7.2.6 Reading writing Microsoft Office Open XLS files (xlsx)

```
library(xlsx)
df <- read.xlsx(path,sheet)
#
write.xlsx(data,path)
#
wb <- createWorkbook()
sheet <- createSheet(wb)
addDataFrame(data,sheet)
saveWorkbook(wb,path)
```

7.2.7 Filesize

For our example we obtain the following sizes:

Format	Size / Bytes
xlsx	128892
xls	454656
dta	59300
dta13	60354
csv	28178
Rdata	17356

7.3 Checking Values

```
load("data/180716_060x_C.Rdata")
```

7.3.1 Range of values

© Oliver Kirchkamp

```
codebook(data.set(trustC))
```

⋮

```
================================================================================

   trustC.Offer 'trustor's offer'

--------------------------------------------------------------------------------

   Storage mode: double
   Measurement: interval

        Min: 0.000
        Max: 1.000
       Mean: 0.654
   Std.Dev.: 0.244
```

```
================================================================================

   trustC.country 'country of origin'

--------------------------------------------------------------------------------

   Storage mode: double
   Measurement: nominal
   Missing values: 98, 99

   Values and labels     N Valid Total

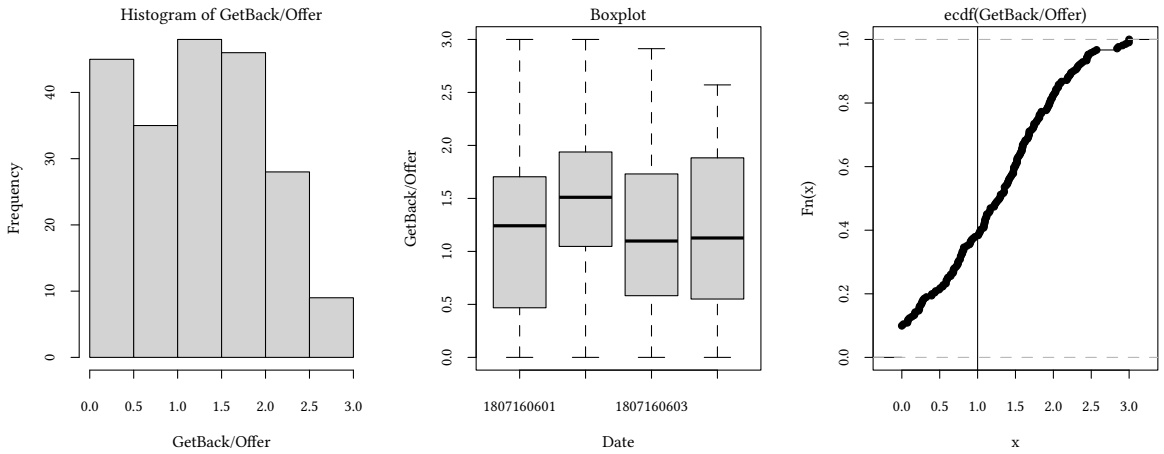
    1   'a'             24   6.2   5.6
    2   'b'             18   4.6   4.2
    3   'c'             18   4.6   4.2
    4   'd'             24   6.2   5.6
    5   'e'             24   6.2   5.6
    6   'f'             24   6.2   5.6
    7   'g'             24   6.2   5.6
    8   'h'             24   6.2   5.6
    9   'i'             18   4.6   4.2
   10   'j'             24   6.2   5.6
   11   'k'             24   6.2   5.6
   12   'l'             18   4.6   4.2
   13   'm'             18   4.6   4.2
   14   'n'             24   6.2   5.6
   15   'o'             24   6.2   5.6
   16   'p'             18   4.6   4.2
   17   'q'             24   6.2   5.6
   18   'r'             18   4.6   4.2
   98 M 'refused'       18         4.2
   99 M 'missing'       24         5.6
```

© Oliver Kirchkamp

7.3.2 (Joint) distribution of values

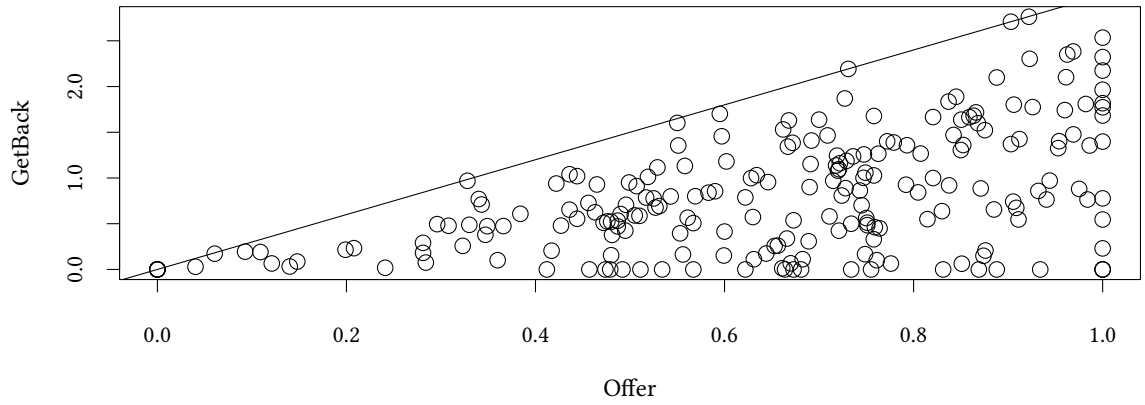
Basic plots

```
with(trustC,hist(GetBack/Offer))
within(trustC,{Date<-sub("_","",Date);boxplot(GetBack/Offer ~ Date,main="Boxplot")})->q
with(trustC,plot(ecdf(GetBack/Offer)))
abline(v=1)
```



Joint distributions First pool all data:

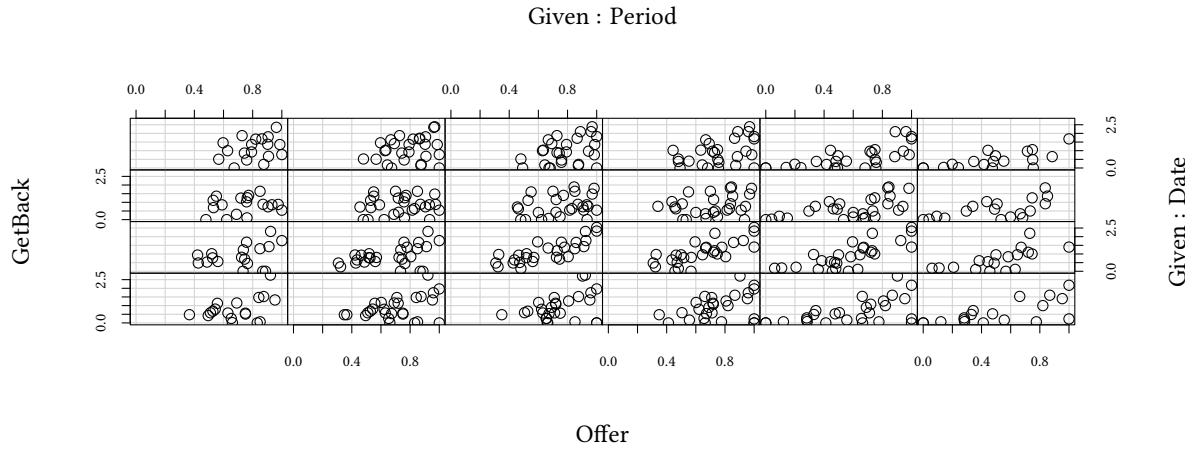
```
plot(GetBack ~ Offer ,data=trustC)
abline(a=0,b=3)
```



If something is suspicious (which does not seem to be the case here) plot the data for subgroups:

© Oliver Kirchkamp

```
coplot(GetBack ~ Offer | Period + Date,data=trustC,show.given=FALSE)
```



The Kakadu data contains variables `lower` and `upper`.

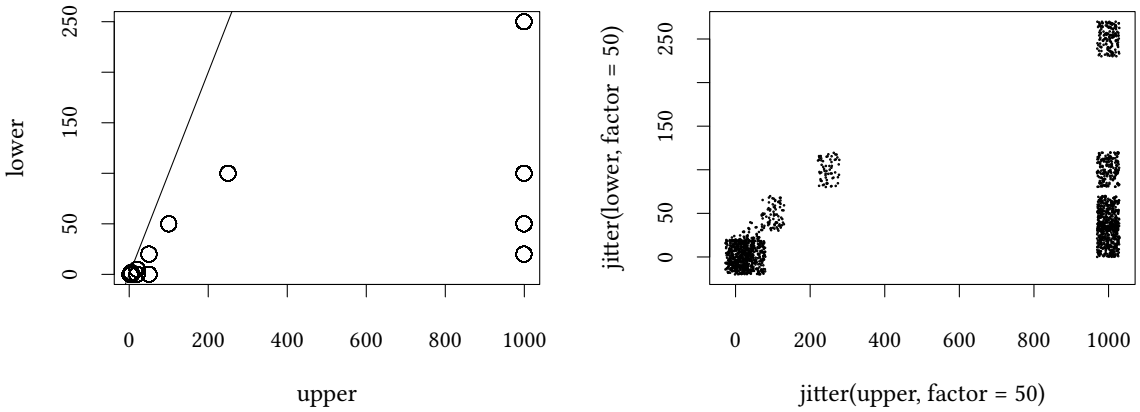
```
data(Kakadu)
nrow(Kakadu)

[1] 1827
```

- `lower`: lowerbound of willingness to pay, 0 if observation is left censored
- `upper` upper bound of willingness to pay, 999 if observation is right censored

When our data falls into a small number of categories a simple scatterplot is not too informative. The right graph shows a scatterplot with some *jitter* added.

```
plot(lower ~ upper,data=Kakadu)
abline(a=0,b=1)
plot(jitter(lower,factor=50) ~ jitter(upper,factor=50),cex=.1,
     data=Kakadu)
```



© Oliver Kirchkamp

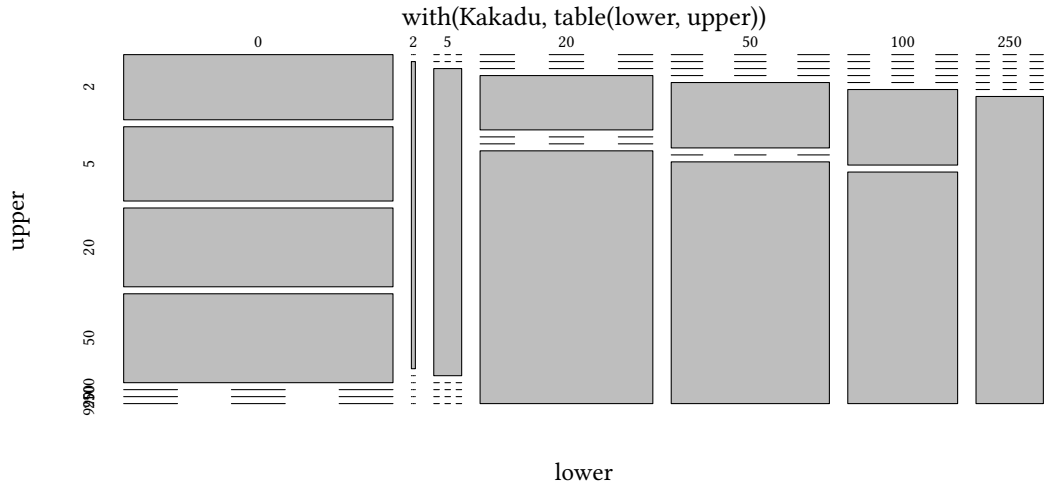
With such a large number of observations, and so few categories, a table might be more informative

```
with(Kakadu,table(lower,upper))
```

	upper						
lower	2	5	20	50	100	250	999
0	129	147	156	176	0	0	0
2	0	9	0	0	0	0	0
5	0	0	63	0	0	0	0
20	0	0	0	69	0	0	321
50	0	0	0	0	76	0	281
100	0	0	0	0	0	61	187
250	0	0	0	0	0	0	152

Tables of frequencies can be displayed as a mosaicplot:

```
mosaicplot(with(Kakadu,table(lower,upper)))
```



7.3.3 (Joint) distribution of missings

- Do we expect any missings at all?
- Are missings where they should be?
  - e.g. number of siblings=0, age of oldest sibling=NA ☑
  - e.g. number of siblings=NA, age of oldest sibling=25 🗲

In our dataset we do not have the age of the oldest sibling, but let us just pretend:

© Oliver Kirchkamp

```
with(trustGS$subjects,table(siblings,age,useNA='always'))

        age
siblings 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 98 99 <NA>
    0     6 12  0  0  6 12 18  0  0  6 12  0  0  6  6  0  6  6 12  6    0
 [ reached getOption("max.print") -- omitted 5 rows ]

with(trustGS$subjects,table(siblings,is.na(age)))


siblings FALSE
      0    114
      1     90
      2     96
      3    126
      99     6
```

The discussion of value labels in section 7.6 contains more details on missings.

7.3.4 Checking signatures

How can we make sure that we are working on the "correct dataset"?

Assume you and your coauthors work with what you think is the same dataset, but you get different results.

Solution: compare checksums.

```
library(tools)
md5sum("data/180716_060x.Rdata")

            data/180716_060x.Rdata
"a03dfad4539631553d292b2746bd81c2"
```

It might be worthwile to include in the draft version of your paper the checksum of your datasets.

7.4 Naming variables

We already mentioned variable names in section 121.

- short but not too short

```
lm ( otherinvestment ~ trust + ineq + sex + age + latitude + longitude)
lm ( R100234 ~ R100412 + R100017 + R100178 + R100671 + R100229 + R100228 )
lm ( otherinvestment ~ trustworthiness + inequalityaversion + sexOfProposer + ageOfProposer + latit
lm ( oi ~ t + i + s + a + l1 + l2)
```

- changing existing variables creates confusion, better create new ones

© Oliver Kirchkamp

- Keep related variables alphabetically together.

  `... ProfitA ProfitB ProfitC ...`

  and not

  `... AProfit BProfit CProfit ...`

- How do we order variable names anyway?

```
trustC[,sort(names(trustC))]
```

7.5 Labeling (describing) variables

- Variable names should be short…
- but after a while we forget the exact meaning of a variable
  - What was the difference between `Receive` and `GetBack` ?
  - Did we code `male=1` and `female=2` or the opposite?
- Labels provide additional information.

Either…

- use a small number of source files, and keep the information somewhere in the file

…or…

- use many source files and few data files, and keep the information with the data.

```
load("data/180716_060x.Rdata")
trust <- within(with(trustGS,merge(globals,subjects)), {
  description(Pos)<- "(1=trustor, 2=trustee)"
  description(Offer)<- "trustor's offer"
  description(Receive)<- "amount received by trustee"
  description(Return)<- "amount trustee sends back to
                                                trustor"
  description(GetBack)<- "amount trustor receives back
                                          from trustee"
  description(country)<- "country of origin"
  description(sex)<- "participant's sex (1=male, 2=female)"
  description(siblings)<- "number of siblings"
  description(age)<- "true age"
  })
codebook(data.set(trust))
attr(trust,"annotation")<-"Note: 180716_0601 was a pilot,..."
annotation(trust)["note"]="Note: This is not a real dataset..."
```

- labels can be long, but they should be meaningful even if they are truncated.

  The following is not a label but a wording:

© Oliver Kirchkamp

```
description(uncondSend) <-  "how much would you send to the
            other player if no binding contract was possible"
description(condSend) <-  "how much would you send to the
 other player if you had the possibility of a binding contract"
```

Better:

```
description(uncondSend) <-  "how much to send without binding contract"
description(condSend) <-  "how much to send with binding contract"
wording(uncondSend) <-  "how much would you send to the other
  player if no possibility of a binding contract was possible"
wording(condSend) <-  "how much would you send to the other
     player if you had the possibility of a binding contract"
```

General attributes

`description()`	short description of the variable	always
`wording()`	wording of a question	if necessary
`annotation()["..."]`	e.g. specific property of dataset	if necessary
	how a variable was created	if necessary

7.6 Labeling values

Let us again list some interesting datatypes:

- numbers: 1, 2, 3
- characters: “male”, “female”, …
- factors: “male”=1, “female”=2,…
  - *factors* are *integers* + *levels*, often treated as *characters*.
  - *factors* have only one type of *missing* (this is not a restriction, since the type of missingness could be stored in another variable)

The `memisc`-package provides another type: *item*

- item: “male”=1, “female”=2,…

  items are *numbers* + *levels*, often treated as *numbers*.

  items can have several types of *missings*. Useful for questionnaire (or from z-Tree).

```
codebook(trustC$sex)
```

© Oliver Kirchkamp

```
================================================================================

   trustC$sex 'participant's sex (1=male, 2=female)'

--------------------------------------------------------------------------------

   Storage mode: double
   Measurement: nominal
   Missing values: 98, 99

   Values and labels     N Valid Total

    1   'male'         174  44.6  40.3
    2   'female'       216  55.4  50.0
   98 M 'refused'       18         4.2
   99 M 'missing'       24         5.6
```

```
table(as.factor(trustC$sex),useNA="always")


  male female   <NA>
   174    216     42

table(as.numeric(trustC$sex),useNA="always")


   1    2 <NA>
 174  216   42

table(as.character(trustC$sex),useNA="always")


female   male   <NA>
   216    174     42
```

`useNA="always"` allows us to count missings. `mean(is.na())` allows us to calculate the *fraction* of missings. The result depends on the representation.

```
mean(is.na(trustC$sex))

[1] 0

mean(is.na(as.factor(trustC$sex)))

[1] 0.09722222

mean(is.na(as.numeric(trustC$sex)))

[1] 0.09722222

mean(is.na(as.character(trustC$sex)))

[1] 0.09722222
```

© Oliver Kirchkamp

How do we add labels to values? (requires `memisc`)

```
trust <- within(trust,{
  labels(sex)<-c("male"=1,"female"=2,"refused"=98,"missing"=99)
  labels(siblings)<-c("refused"=98,"missing"=99)
  labels(age)<-c("refused"=98,"missing"=99)
  labels(country)<-c("a"=1, "b"=2, "c"=3, "d"=4, "e"=5, "f"=6, "g"=7, "h"=8, "i"=9, "j"=10, "k"=11, "l"=1
  missing.values(sex)<-c(98,99)
  missing.values(siblings)<-c(98,99)
  missing.values(age)<-c(98,99)
  missing.values(country)<-c(98,99)
  })
```

7.7 Recoding data

7.7.1 Replacing meaningless values by missings

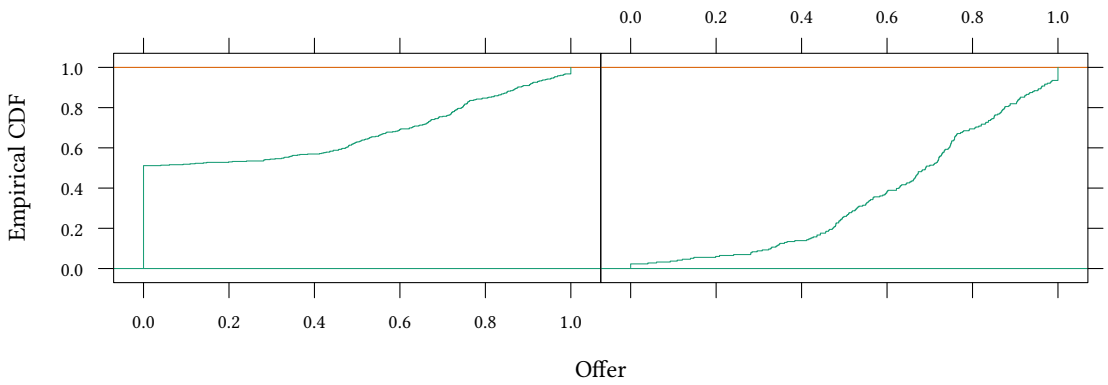
In our trust game not all players have made all decisions. z-Tree coded these "decisions" as zero. This can be misleading. Better code them as missing.

```
trustC <- within(trust, {
  Offer   [Pos==2 & Offer==0]   <-NA
  GetBack[Pos==2 & GetBack==0]<-NA
  Receive[Pos==1 & Receive==0]<-NA
  Return  [Pos==1 & Return==0] <-NA
  })
```

```
save(trustC,file="data/180716_060x_C.Rdata")
```

Introducing missings makes a difference. The left graph shows the plot where *missings* were coded (wrongly) as *zeroes*, the right graph shows the plot with *missings* coded as *missings*.

```
c(ecdfplot(~Offer,data=trust),ecdfplot(~Offer,data=trustC))
```



© Oliver Kirchkamp

```
mean(trust$Offer)

[1] 0.3268388

mean(trustC$Offer)

[1] NA

mean(trustC$Offer,na.rm=TRUE)

[1] 0.6536776
```

7.7.2 Replacing values by other values

Sometimes we want to simplify our data. E.g. the siblings variable might be too detailed.

```
trustC <- within(trustC,altSiblings<-recode(siblings,
                    "single child"=0 <- 0,
                    "siblings"    =1 <- range(1,50),
                    "refused"     =98 <- 98,
                    "missing"     =99 <- 99))
```

7.7.3 Comparison of missings

We can not compare NAs. The following will fail in R:

```
if(NA == NA) print("ok")

Error in if (NA == NA) print("ok"): missing value where TRUE/FALSE needed

if(7 < NA) print("ok")

Error in if (7 < NA) print("ok"): missing value where TRUE/FALSE needed
```

(Note that the equivalent in Stata, `. == .` and `7 < .`, do not fail but return TRUE.)
The following works:

```
x<-NA
if(is.na(x)) print("x is na")

[1] "x is na"
```

7.8 Changing variabes – creating new variables

- give them new names (overwriting “forgets” previous information)
- give them labels
- keep the old variables

© Oliver Kirchkamp

7.9 Select subsets

(See the remarks on subsetting in section 6.1)

- delete records you will never ever use (in the cleaned data, not in the raw data)

```
trust<-subset(trust,Pos!=2)
```

- generate indicator variables for records you will use in a specific context

```
trust<-within(trust,youngSingle <- age<25 & siblings==0)
with(subset(trust,youngSingle),...)
```

8 Exercises

Exercise 1
Have a look at the dataset `Workinghours` from the library `Ecdat`. Compare the distribution of "other household income" for whites and non-whites. Do the same for the different types of occupation of the husband.

Exercise 2
Read the data from a hypothetical experiment from `rawdata/Coordination`. Does the `Effort` change over time?

Exercise 3-a
Read the data from a hypothetical z-Tree experiment from `rawdata/Trust`. Do you find any relation between the number of siblings and trust?

Exercise 3-b
For the same dataset: Attach a label (description) to `siblings`. Attach value labels to this variable.

Exercise 3-c
Make the above a function.

Also write a function that compares the offers of all participants with n siblings with the other offers. This function should (at least) return a p-value of a two-sample Wilcoxon test (`wilcox.test`). The number n should be a parameter of the function.

Exercise 4
Read the data from a hypothetical z-Tree experiment from `rawdata/PublicGood`. The three variables `Contrib1`, `Contrib2`, and `Contrib3` are contributions of the participants to the other three players in their group (in gruops of four).

1. Check that, indeed, in each period, players are equally distributed into four groups.
2. Produce for each period a boxplot with the contribution (i.e. 16 boxplots in one graph).

© Oliver Kirchkamp

3. Add a regression line to the graph.
4. Produce for each contribution partner a boxplot with the contribution (i.e. 3 boxplots in one graph).
5. Produce an `Sweave` file that generates the two graphs. In this file also write when you estimate the average contribution reaches zero.

Working with R

1. Basics
   a) Interacting, getting help
   b) How R is organised: Packages, CRAN
   c) Data types (numbers, characters, …)
   d) Scope
   e) Randomness
2. Functions (to structure our work)
3. Control structures (repetition)
4. Structuring data (grouping, aggregating,…)
5. Graphs (plot, lattice, ggplot2)

Workflow with R

read data
clean data
describe, estimate, test } none of this is obvious

- What is a “good” workflow (for us)?
- How can we document work?
  (for us / for others)

1. Replication (robustness…)
2. Cleaning (reading, recoding, …)
3. Structuring work (repetition, …)
4. Documenting work (weaving, knitting…)
5. Version control (git)

© Oliver Kirchkamp

Working with R

Video (≈ 60) min ↓ Exercise (5-10 min) ↓ Interaction (≈ 60) min	Video (≈ 60) min ↓ Exercise (5-10 min) ↓ Interaction (≈ 60) min	Video (≈ 60) min ↓ Exercise (5-10 min) ↓ Interaction (≈ 60) min
Video (≈ 60) min ↓ Exercise (5-10 min) ↓ Interaction (≈ 60) min	Video (≈ 60) min ↓ Exercise (5-10 min) ↓ Interaction (≈ 60) min	

Workflow with R

	Video (≈ 60) min ↓ Exercise (5-10 min) ↓ Interaction (≈ 60) min	Video (≈ 60) min ↓ Exercise (5-10 min) ↓ Interaction (≈ 60) min
Video (≈ 60) min ↓ Exercise (5-10 min) ↓ Interaction (≈ 60) min	Video (≈ 60) min ↓ Exercise (5-10 min) ↓ Interaction (≈ 60) min	Video (≈ 60) min ↓ Exercise (5-10 min) ↓ Interaction (≈ 60) min