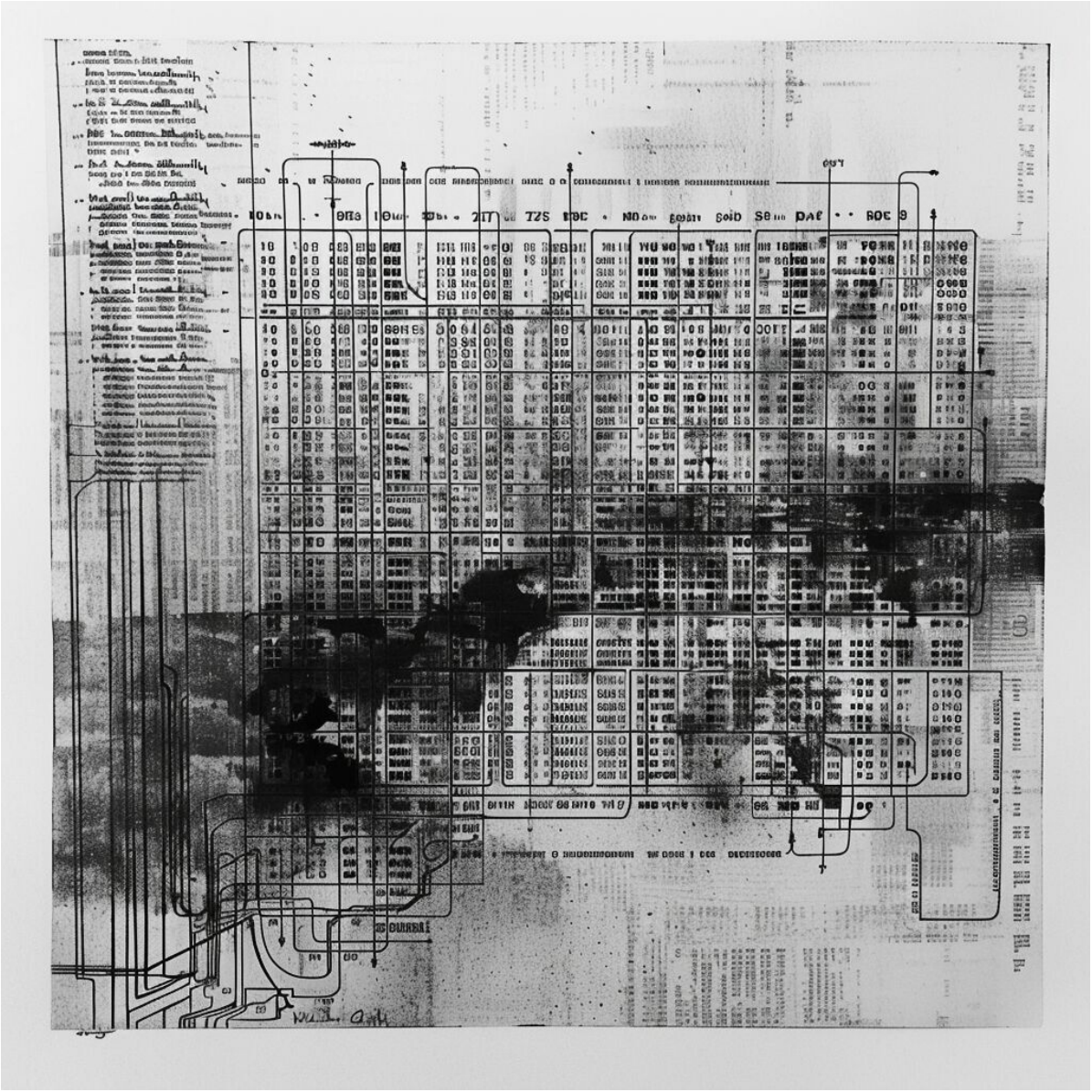
Data management
for the IMPRS BeSmart



Oliver Kirchkamp

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.

Contents

1 Introduction

Literature:

General Literature

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

- 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.

R

- Hadley Wickham; Advanced R.

Topics covered in the course

- Introduction
  - Aims of statistical data analysis
  - Why worry about data management?
- Documentation (1): literate programming.
- Documentation (2): version control (git)
- Organising work
  - Programming techniques
  - Repetition
  - Robustness
  - Data manipulation

- Working with data
  - Reading data
  - Cleaning data
  - Organising data
  - Descriptive statistics
  - Specific results

1.1 Data management

Data management…?

Data management
…handling data as a valuable resource.
(as opposed to process management).

↓

Ensure

- availability of data
- quality of data

under constraints (legal, technical, …).

Data management
Ensure

- availability of data
- quality of data

under constraints (legal, technical, …).

→
which
data

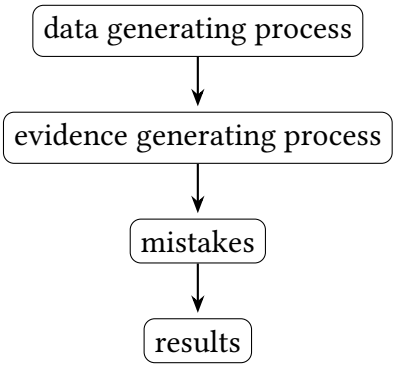
Data Lifecycle in Research

1. Data Planning
   - Data Management Plan (DMP)
   - Data types and formats
   - Planning for data storage, access, and sharing
2. Data Collection

- Reliable data collection methods and tools
- Data quality and consistency
- Metadata

3. Data Processing
   - Data cleaning (handling missing values, outliers)
   - Data transformation and normalization
   - Data integration from multiple sources
4. Data Analysis
   - Statistical methods or computational models
   - Software tools for data analysis (e.g., R, Python,…)
   - Visualizations to explore data patterns
5. Data Preservation
   - Storage
   - Backup
6. Data Sharing
   - Repositories
   - Consent
   - Licenses
7. Data Reuse
   - Replication
   - Meta-analyses

A statistical perspective



Aims of data management

1. Ensure Data Quality:
   Data integrity and accuracy.
2. Reproducibility:
   Clear documentation and consistent practices.
3. Collaboration:
   Data sharing practices.
4. Compliance:
   Ethical, legal, and institutional requirements.
5. Impact:
   Data and methods can be reused in future research.

1.2 Reproducibility

Definition 1:

- *Reproducible* $\leftrightarrow$ with the same data and the same routines, we obtain the same result.

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.

Definition 2:

- *Reproducible* $\leftrightarrow$ "…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?

- It helps if we can reproduce our own work!
- 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*!)

Reproducibility – Anecdotical evidence Buckheit, J. B., and D. L. Donoho. 1995. 'WaveLab and Reproducible Research'. Tech Rep 474. Stanford University:

- Computational methods are lost
- Researchers can't reproduce their own work
- Researchers can't communicate their work
- Researchers can't reproduce the work of others

Yale Law School Roundtable on Data and Code Sharing. 2010. 'Reproducible Research – Addressing the Need for Data and Code Sharing in Computational Science'. *Computing in Science & Engineering* 12

→ Make "computational research details readily available"!

Consistency in the analysis, mistakes 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."

Reproducibility – Evidence generating process 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.

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

Is data management 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. 2024. "Nonstandard Errors". *Journal of Finance*. 79(3), 2339-2390.

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

  data: 17 years of EuroStoxx 50 index futures (720 000 000 trade records).

  RQs: market-efficiency, bid-ask spread, …gross trading revenue of clients → effect size + sd. error

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

  - DGP, data generating process (sampling error, "standard error")
  - EGP, evidence generating process (nonstandard error).

Computational Reproducibility in Finance Computational Reproducibility in Finance: Evidence from 1,000 Tests

Pérignon, Akmansoy, Hurlin, Dreber, Holzmeister, Huber, Johannesson, Kirchler, Menkveld, Razen, Weitzel. 2024.

The Review of Financial Studies. 1-36.

168 research teams from 37 countries.

- 52%: results could be reproduced exactly.
- 9.5%: results differ, but conclusion remains the same.
- 9.4%: results differ and conclusions change.
- 29.1% verifier was unable to make the code run.

Correlated with reproducibility:

- \- tails of result distribution
- \- technically challenging
- \+ coding skills
- \+ quality of documentation
- 0 seniority, publications, citations, gender, location, coauthors

Spreadsheet errors 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.

Reproducibility – Summary Without proper management…

- Data is not available.
- Results can't be reproduced.
- Type of analysis (EGP) is unclear.
- Analysis is inconsistent.

1.3 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.

Replicability — Why is this a problem? 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
- ⋮

1.4 Back to data management

Research integrity Data related

1. Honesty, Accuracy
   - Data collection.
   - Analysis.
   - Reporting results.
2. Reliability, Reproducibility, Replicability
   - Consistent methods.
   - Documentation (data+methods).
   - Storage.
3. Confidentiality
   - Data Privacy.
   - Data Security.
4. Responsibility
   - Data Management, protocols, standards, storage, backup, and sharing.
   - Compliance.
5. Data Management Policy
   - Developing a DMP.
   - Training, Mentorship, Audits, Reviews.

General

1. Openness
   - Sharing data, methods, results.
2. Objectivity
   - Avoid Bias.
   - Impartial Analysis.
3. Fairness
   - Credit and Acknowledgment.
   - Equitable Treatment.
4. Ethical Considerations

- Informed Consent.
- Avoiding Harm.

5. Institutional Support
   - Policies, Guidelines.
   - Ethics Committees and Review Boards.

Data management – most of this I am going to leave out

1. Data Organization and Documentation
   - Best practices for data organization (naming conventions, folder structure)
   - Metadata creation and management
     Data Documentation Initiative (DDI)
     Dublin Core Metadata Initiative (DCMI)
   - Writing effective data documentation (data dictionaries, codebooks)
2. Data Storage and Access
   - Options for data storage (cloud storage, institutional repositories)
   - Data accessibility (formats, interoperability)
   - Backup strategies and data versioning
3. Data Privacy and Ethical Considerations
   - Compliance with ethical guidelines (IRB, consent)
   - Sensitive data (anonymization, encryption)
   - Data privacy laws relevant to research
4. Data Quality and Cleaning
   - Datacleaning and preprocessing
   - Missing data and outliers
   - Data accuracy and consistency
5. Data Integration and Collaboration
   - Combining data from multiple sources
   - Collaborative tools for data sharing and project management
   - Version control systems (e.g., Git)
6. Data Analysis and Visualization
7. Data Preservation and Sharing

- Long-term storage and archiving of data
- Data sharing policies and practices (e.g., open data, data repositories)
- Citing and attributing data properly

8. Reproducibility and Open Science
   - Principles of reproducible research
   - Open science frameworks and tools
   - Creating reproducible workflows

Topics in this course

- Storing data
- Processing data
- Documentation…
  - …of the data.
  - …what we do with the data (our analysis).

Data Sharing Tools

1. Sharing code: Git (GitHub/GitLab/Bitbucket)
   - Purpose: Version control and collaborative coding.
   - Features: Repositories, pull requests, issue tracking, branches, and collaborative code reviews.
2. Sharing data openly (Dryad/Zenodo/Figshare)
   - Purpose: Open data repositories for sharing research data.
   - Features: DOI generation, metadata management, compliance with funder requirements, and integration with publisher platforms.
3. Sharing data with access control (Dataverse/DSpace/CKAN)
   - Purpose: Research data repository.
   - Features: Data citation, metadata standards compliance, and controlled access features.
4. Cloud storage (Google Drive/Dropbox/OneDrive)
   - Purpose: Cloud storage and file sharing.
   - Features: Document synchronization, real-time collaborative editing, and accessibility from multiple devices.

Project Management Tools

1. Trello
   - Purpose: Task and project management.
   - Features: Kanban boards, checklists, due dates, labels and tags, and integration with other apps.
2. Asana
   - Purpose: Team and project management.
   - Features: Task assignments, timelines (Gantt charts), milestones, and project tracking dashboards.
3. Slack, Microsoft Teams:
   - Purpose: Team communication and collaboration.
   - Features: Channels, direct messaging, file sharing, calendar integration, and video conferencing.
4. Notion:
   - Purpose: All-in-one workspace for note-taking, task management, and project planning.
   - Features: Databases, tables, calendars, Kanban boards, collaborative document editing, and templates.
5. Jira:
   - Purpose: Agile project management, especially useful for software development projects.
   - Features: Issue and bug tracking, Scrum and Kanban boards, sprint planning, and reporting tools.
6. Confluence:
   - Purpose: Team collaboration and knowledge management.
   - Features: Wikis, collaborative workspaces, document sharing, and integration with Jira.

Writing Tools

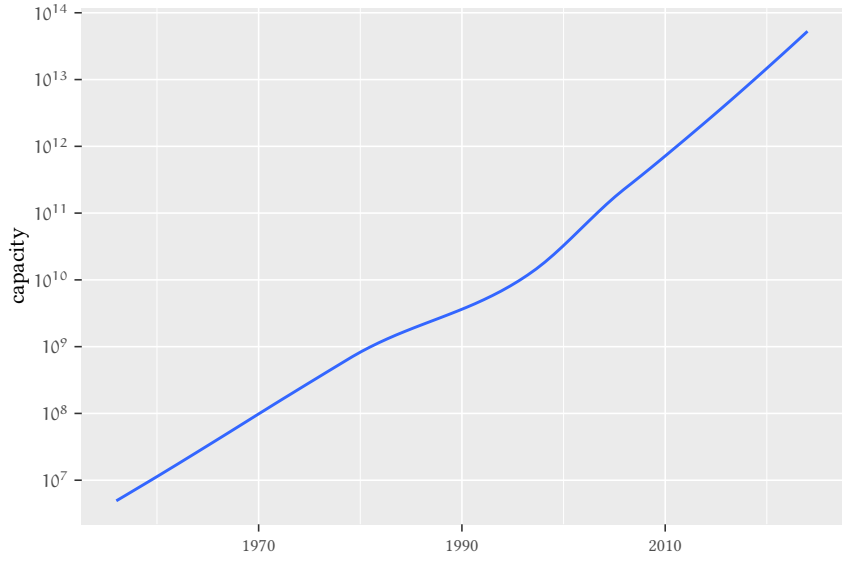
1. Reference managers: Bibtex/Zotero/EndNote/Mendeley:
   - Purpose: Reference management.
   - Features: Bibliography creation, citation management, PDF annotation, and collaborative bibliographies.
2. Overleaf:

- Purpose: Collaborative writing for LaTeX documents.
- Features: Real-time collaborative editing, version control, and templates for academic papers.

Best Practices

1. Data Management Plans (DMPs)
   - How data will be managed and shared during and after the research project.
2. Documentation and Metadata
   - Data remains understandable and reusable.
3. Security and Privacy
   - Data security
   - Privacy concerns
   - Compliance with institutional and legal regulations.

Storage Development of storage capacity over time:



Abstract representation

We translate the physical representation of data on a hard disk into files and folders, etc.

Filesystems

	Filename length	different characters	Pathname length	Filesize (bytes)	Volumesize (bytes)	Files	Data-Checksums	Snapshots
NTFS	255	16400	32800	$9 \cdot 10^{15}$	$9 \cdot 10^{15}$	$4 \cdot 10^{9}$		
ext4	255	1050000	-	$2 \cdot 10^{13}$	$1 \cdot 10^{18}$	$4 \cdot 10^{9}$		
APFS	255	131000	-	$9 \cdot 10^{18}$	?	$9 \cdot 10^{18}$		X
btrfs	255	1050000	-	$2 \cdot 10^{19}$	$2 \cdot 10^{19}$	$2 \cdot 10^{19}$	X	X
ZFS	255	1050000	-	$2 \cdot 10^{19}$	$3 \cdot 10^{38}$	$3 \cdot 10^{38}$	X	X

Why care about checksums?

- Bit flips on disk $\approx$ every $4 \cdot 10^{12}$ read bytes. (1TiB = $1 \cdot 10^{12}$ bytes)
- Bit flips in RAM $\approx$ every $7 \cdot 10^{7}$ bytes/h. (1GiB = $1 \cdot 10^{9}$ bytes)

Data with structure

- Relational

  SQL (tables with relations, 1986)

  Transactions are: Atomic, consistent, isolated, durable (1973).

- Structured storage, NoSQL

  Distributed storage

- Data Warehouse

  Extract $\rightarrow$ Transform $\rightarrow$ Load

- Data Lake

  Unstructured

1.5 Structure of a paper

- Describe the research question

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

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

- Describe the 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*

Why should reproducibility be a problem?

- 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`...
  - 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,...)

The problem of "reproducibility" is enormous 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 is 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:

- Both problems are solved independently:
  - Probability of (undiscovered) mistake A: $0.1 \cdot 0.2 = .02$.
  - Probability of (undiscovered) mistake B: $0.1 \cdot 0.2 = .02$.
  - 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

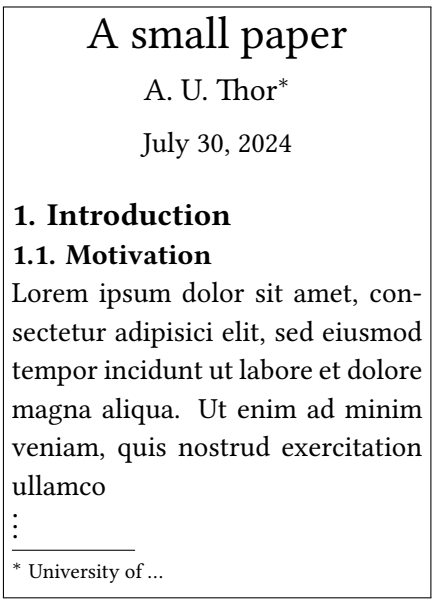
- Clear division of labour
  - the "experimenter" decides how the experiment is actually run
  - the "empiricist" decides what statistics and graphs are produced
  - the "writer" decides how to present the text
  - help, do not interfere
- In our communication: concentrate on the essentials:
  - exchange one file
  - make only essential changes to this file
  - clearly explain why these changes are necessary

2 Weaving and tangling

2.1 Markup languages

In this course we mention only two markup languages, LaTeX and Markdown. There are many more markup languages.

Visual apprearance:



A small paper

A. U. Thor*

July 30, 2024

1. Introduction

1.1. Motivation

Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco

⋮

* University of ...

Markup as LaTeX

```
\documentclass{article}
\begin{document}
\title{My first paper}
\author{A. U. Thor\thanks{University...}}
\date{\today}
\maketitle
\section{Introduction}
\subsection{Motivation}
...
\end{document}
```

- Very flexible, complex
- Creates PDF

Markup as Markdown

```
---
author:
- A. U. Thor^[University of ...]
title: My first paper
date: "2024-07-27"
...

# Introduction
## Motivation

...
```

- Limited layout
- Creates PDF, HTML, odt, docx…

Describing the document with a markup language

- …helps separating the content and the visuals (division of labour).
- …simplifies managing versions of the document.
- …simplifies collaboration.

Again: the structure of a paper

- Describe the research question.

  Which model do we use to structure this question?

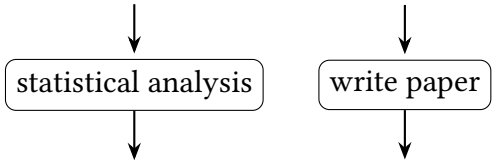
  Which hypotheses do we want to test?
- Describe the method.
- Describe the sample.

  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

2.2 When paper and results develop independently...



If statistical analysis and the paper develop independently, what can we do to link one to the other? Lots of notes in the paper, e.g. the following:

In your LaTeX-file...:

```
%
% the following table was created by tableAvgProfits()
%                              from projectXYZ_240621.R
% \begin{table}
% ...
```

Better: Weave (Sweave, knitr)

2.3 A history of literate programming

Donald Knuth Donald Knuth

- Task: Writing a book on compilers for computer programs (≈ 1963).

  → "The art of computer programming"...(≈ 1968)

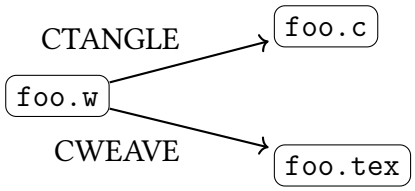
  → Develop typsetting software...(≈ 1970)

  → Develop a system to write documentation for this software...(≈ 1987)

  → ...

  → ...

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

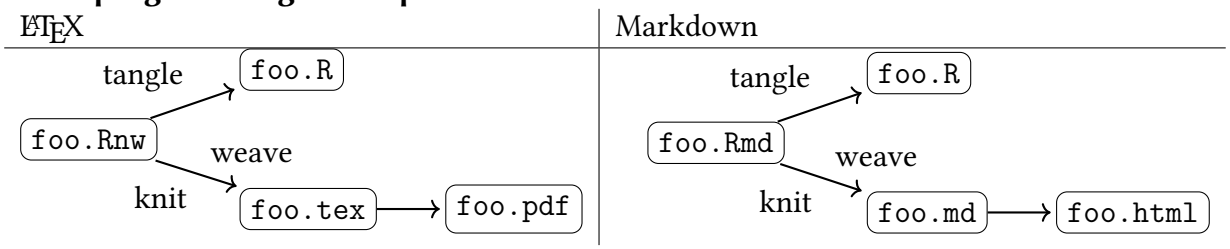


What is "literate programming":

- 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:



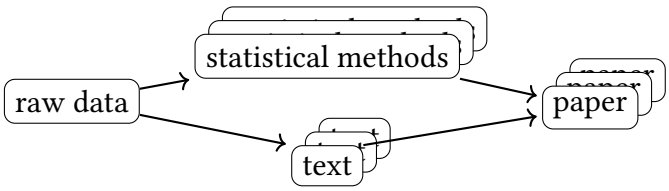
- tangle (Stangle, knit(..., tangle=TRUE)): `foo.Rnw` → `foo.R`
- weave (Sweave, knit): `foo.Rnw` → `foo.tex`
  (may contain parts of `foo.R`)

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)

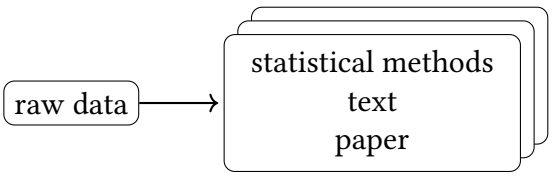
Nonliterate versus literate work

Nonliterate:



- Easy to mix up different versions of analysis and their relation to the versions of the paper.

Literate:



- Avoids confusion about which parts of text and analysis belong together and which do not.

Platforms for literate programming

You can choose among a large number of platforms.

- Emacs
- RStudio
- Jupyter Notebook
- Google Colab
- Deepnote
- Apache Zeppelin
- Kaggle Notebooks
- Observable
- Polynote
- Nextjournal

  ⋮

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.
```

or

```
We ran `r numSession` sessions with
`r numPart` participants.
```

2.4 An Rnw document with LaTeX

Here is a brief Rnw-document:

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

```
<<someCalculations,results='asis',echo=FALSE>>=
library(Ecdat)
library(xtable)
library(ggplot2)
data(Caschool)
Caschool |>
   lm(testscr ~  avginc,data=_) |>
   anova() |>
   xtable()
@
```

```
<<aFigure,echo=FALSE,fig.width=4,fig.height=3>>=
Caschool |>
ggplot(aes(x=avginc,y=testscr)) + geom_point() + geom_smooth() +
                          labs(x="average income",y="testscore")
@
```

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

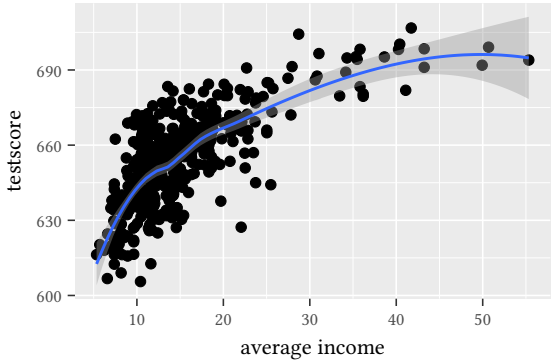
To compile this Rnw-file, we can do the following:

```
library(knitr)

knit("filename.Rnw")

system("pdflatex filename.tex")
```

…or use a front end like RStudio.
The result, after knitting:

text explaining 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 ...

2.5 Why use markup languages?

Visual representation

1. Introduction

...

1.1 Literature

...[1][2]

[1][2] We should add that this is more complicated...

- We describe how the text should look like.
- But we are not typographers.
- If the logic of our text changes, it is hard to adjust all visuals.
- Version control is hard.

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.
- Version control is easy.

2.6 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(ggplot2)
data(Caschool)
Caschool |>
   lm(testscr ~  avginc,data=_) |>
   anova() |>
   xtable()
```
```{r aFigure,echo=FALSE,fig.width=4,fig.height=3}
Caschool |>
ggplot(aes(x=avginc,y=testscr)) + geom_point() + geom_smooth() + labs(x="average income",
```

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

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

2.7 Text chunks

What we have seen:

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

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

LaTeX(Rnw)	Markdown (Rmd)	
`<<anyName,...>>=`	```` ```{r anyName,...} ````	Not necessary, but identifies the chunk.
`<<...,engine="python",...>>=`	```` ```{python ...} ````	Language to use for this chunk.
`<<...,eval=FALSE,...>>=`	```` ```{r ...,eval=FALSE,...} ````	This chunk will not be evaluated (too time consuming...)
`<<...,echo=FALSE,...>>=`	```` ```{r ...,echo=FALSE,...} ````	The code of this chunk will not be shown.
`<<...,include=FALSE,...>>=`	```` ```{r ...,include=FALSE,...} ````	Neither code nor output of this chunk will be shown.
`<<...,fig.width=3,...>>=`	```` ```{r...,fig.width=3...} ````	All figures produced in this chunk will have this width.
`<<...,fig.height=3,...>>=`	```` ```{r...,fig.height=3,...} ````	All figures produced in this chunk will have this height.
`<<...,results='asis',...>>=`	```` ```{r ...,results='asis',...} ````	The chunk produces LaTeX-output (markdown-output) which should be inserted here 'as is'.
`\Sexpr{...}`	`` `r ...` ``	The output of these expressions goes into the text.

More elements of a knitr-document

We can set "default" parameters at the beginning of the document:

We can set "default" parameters at the beginning of the document:

```
\documentclass{article}
\begin{document}
<<>>=
opts_chunk[["set"]](dev='tikz', external=FALSE,
                    fig.width=4.5, fig.height=3,
                    echo=FALSE, 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 generates the following text:

> This document has been generated on July 30, 2024, with R version 4.2.2 Patched (2022-11-10 r83330), on x86_64-pc-linux-gnu.

To include information about attached packages, use `sessionInfo()`:

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

ggplot2 3.5.1, xtable 1.8-4, Ecdat 0.4-2, Ecfun 0.3-2, knitr 1.48 .

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 is 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
```

With psycho we can present this information not as a tabled, but with more text:

```
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).

2.8 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)

2.9 Practical issues

What if some calculations take too much time Often 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:

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

2.10 When R produces tables

2.10.1 Tables

You can save a lot of work if you harness R to create and format your tables for you. Interesting functions are `xtable`, `kable` (and also `huxtable`):

In this example we generate a small table:

```
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
```

So far, the table does not look very nice.

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

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

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

```
@
```

```
kable(x)
```

-0.5604756	1.5587083	0.1292877
-0.2301775	0.0705084	1.7150650

We 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 `$\alpha$`. This is not what we want. We define our own sanitation function:

```
<<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

```
@
```

The same with `kable`

```
x <- matrix(rnorm(12),4,3)
colnames(x)<-c("$\\alpha$","$\\beta$",
               "$\\gamma$")
library(kableExtra)
kable(x,escape=FALSE,digits=2,booktabs=TRUE) |>
  kable_styling(latex_options = c("striped", "hold_position"),
  stripe_color = "gray!30")
```

Table 1: A small table with kable.

α	β	γ
0.46	1.22	-0.56
-1.27	0.36	1.79
-0.69	0.40	0.50
-0.45	0.11	-1.97

The output is shown in Table 1.

2.10.2 Regression results

Single estimations

```
library(Ecdat)
data(Caschool)
lm(testscr ~ str + elpct + avginc,data=Caschool) |>
           summary() |>
           xtable()
```

Table 2: Output of a simple regression with kable.

term	estimate	std.error	statistic	p.value
(Intercept)	640.31550	5.77489	110.87934	0.00000
str	-0.06878	0.27691	-0.24837	0.80397
elpct	-0.48827	0.02928	-16.67394	0.00000
avginc	1.49452	0.07483	19.97124	0.00000

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	640.3155	5.7749	110.88	0.0000
str	-0.0688	0.2769	-0.25	0.8040
elpct	-0.4883	0.0293	-16.67	0.0000
avginc	1.4945	0.0748	19.97	0.0000

Single estimations with `kable`

```
library(Ecdat)
data(Caschool)
lm(testscr ~ str + elpct + avginc,data=Caschool) |>
           summary() |>
           broom::tidy() |>
           kable(digits=5,booktabs=TRUE) |>
           kable_styling(latex_options = c("striped"),
           stripe_color = "gray!30")
```

The output is shown in Table 2.
We can still improve the heading of the table:

```
library(Ecdat)
data(Caschool)
lm(testscr ~ str + elpct + avginc,data=Caschool) |>
           summary() |>
           broom::tidy() -> est
           colnames(est) <- paste("\\multicolumn{1}{c}{",
                c("","$\\beta$","$\\sigma$","$t$","Pr($>|t|$)"),"}")
           kable(est,escape=FALSE,format="latex",digits=5,booktabs=TRUE) |>
           kable_styling(latex_options = c("striped"),stripe_color = "gray!30")
```

The output is shown in Table 3.

2.10.3 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.

Table 3: Nicer headings for the regression.

	β	σ	t	Pr(> |t|)
(Intercept)	640.31550	5.77489	110.87934	0.00000
str	-0.06878	0.27691	-0.24837	0.80397
elpct	-0.48827	0.02928	-16.67394	0.00000
avginc	1.49452	0.07483	19.97124	0.00000

Table 4: Random effects for the regression with mixed effects with kable.

grp	var1	var2	vcov	sdcor
Consumer	(Intercept)	NA	0.8254	0.9085
Residual	NA	NA	4.3769	2.0921

```
library(lme4)
library(lmerTest)
lmer(Informed.liking ~ Product + (1|Consumer) , data=ham) |>
      summary() -> est.lme4
      est.lme4$varcor |> data.frame() |> xtable()
```

	grp	var1	var2	vcov	sdcor
1	Consumer	(Intercept)		0.83	0.91
2	Residual			4.38	2.09

```
      est.lme4$coefficients |> xtable()
```

	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

Mixed effects with `kable` We start with the random effects.

```
est.lme4$varcor |>
      kable(digits=4,booktabs=TRUE) |>
    kable_styling(latex_options = c("striped"),stripe_color = "gray!30")
```

The output is shown in Table 4. Next we show the fixed effects:

```
est.lme4$coefficients |>
      kable(digits=4,booktabs=TRUE) |>
      kable_styling(latex_options = c("striped"),stripe_color = "gray!30")
```

The output is shown in Table 5.

Table 5: Fixed effects for the regression with mixed effects with kable.

	Estimate	Std. Error	df	t value	Pr(>|t|)
(Intercept)	5.8086	0.1929	320.6842	30.1129	0.0000
Product2	-0.7037	0.2325	564.0000	-3.0273	0.0026
Product3	0.2840	0.2325	564.0000	1.2215	0.2224
Product4	0.1173	0.2325	564.0000	0.5045	0.6141

2.10.4 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`).

```
list(small=testscr~str,
   medium=testscr~str+elpct,
   large=testscr~str+elpct+avginc) |>
purrr::map( ~ lm(.x,data=Caschool)) |>
texreg::texreg(table=FALSE)
```

	small	medium	large
(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

2.10.5 Comparing models with mixed effects

```
library(lme4)
library(lmerTest)
list(small=Informed.liking~Product +
                           (1|Consumer),
     larger=Informed.liking~Product*Information+
                             (1|Consumer)) |>
purrr::map( ~ lmer(.x,data=ham)) |>
texreg::texreg(table=FALSE,single.row=TRUE)
```

	small	larger
(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

2.11 Alternatives to LaTeX

Other formats

- `knitr` can create other formats:
  - Markdown (`md`) → `html`, `docx`, `odt`...
- pander: pandoc, HTML, PDF, docx, odt

Incremental assembly

- ReportRs: docx, odt

(similar to Stata's `putdocx`)

2.11.1 Markdown

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

```r
 library(Ecdat)
 library(xtable)
 library(ggplot2)
 data(Caschool)
 Caschool |>
    lm(testscr ~  avginc,data=_) |>
    anova() |>
    kable()
````
````

````
```

```r
 Caschool |>
 ggplot(aes(x=avginc,y=testscr)) + geom_point() + geom_smooth() +
                            labs(x="average income",y="testscore")
```
the correlation between average income and testscore is
`r with(Caschool,round(cor(testscr,avginc),4))`
more text ...
````

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

```
library(knitr)
knit("filename.Rmd")
pandoc("filename.md","html")
```

2.11.2 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)

2.12 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_240601.Rnw`. Then here is my `Makefile`

```
PROJECT = myProject_240601

pdf: $(PROJECT).pdf

%.pdf: %.tex
     pdflatex $<

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

Let us go through the individual lines of this Makefile.
```

```
PROJECT = myProject_240601
```

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_240601.pdf`, and now make knows that we require a file `myProject_240601.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`.

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_240601
RPARTS        = $(wildcard $(PROJECT)_[1-9].Rnw)
TEXPARTS      = $(RPARTS:.Rnw=.tex)

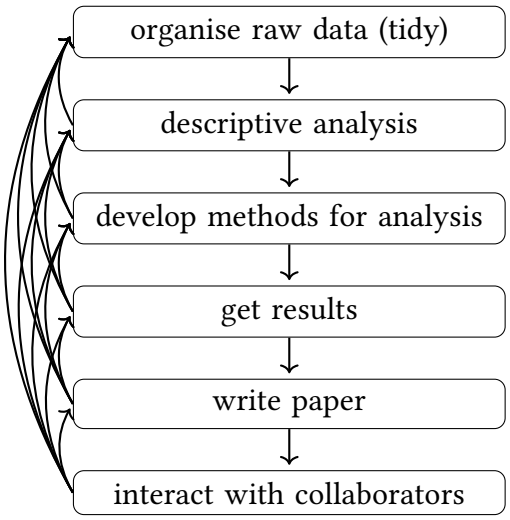
pdf: $(PROJECT).pdf

# 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

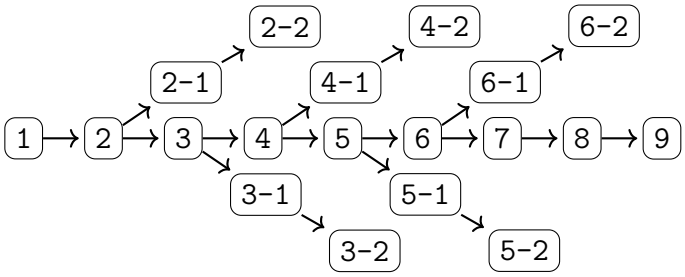


Solutions and restrictions: During our research we create a lot of intermediate results. How can we organise these results?

- 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 snapshots) $\xrightarrow[\text{text}]{\text{only}}$ 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
```

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_240521.Rdata
cleaned_data_240522.Rdata
cleaned_data_240522b.Rdata
⋮
preparingData_240521.R
preparingData_240522.R
descriptives_240522.R
econometrics_240523.R
⋮
paper_240524.Rnw
paper_240525.Rnw
paper_240527.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`!

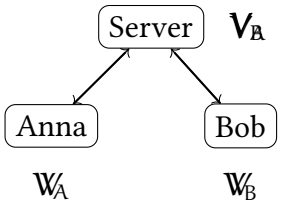
- …more feedback from Prof. Smith.
- Student without a name types…
- …Prof. Smith reading `FINAL_rev.8.comments5.CORRECTIONS.doc`…
- …`FINAL_rev.24.comments7.corrections9.MORE.30.doc`…
- ⋮

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

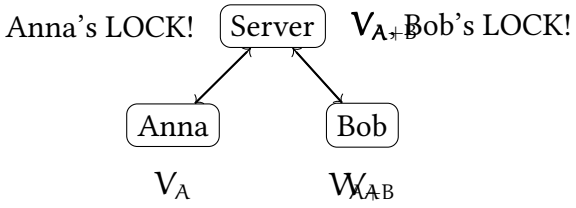
What happens if two authors, Anna and Bob, simultaneously want to work on the same file. Chances are that one is deleting the changes of the other. (This problem is similar to one author working on two different machines)



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

3.4 A "simple" solution: locking

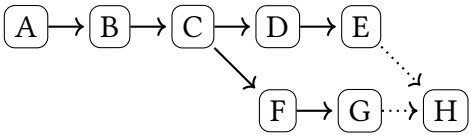
Serialising the workflow might help. Anna could put a "lock" on a file while she wants to edit this file. Only when she is finished, the "unlocks" the file and Bob can continue.



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

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”

- Cloud storage with limited snapshots
- Keep different version of files with different names…

  ```
  FINAL.doc
  FINAL_rev.2.doc
  FINAL_rev.8.comments5.CORRECTIONS.doc
  FINAL_rev.24.comments7.corrections9.MORE.30.doc
  ...
  ```

- Filesystems with snapshots
  - ZFS, BtrFS,…
- Version control

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
- 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
```

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...

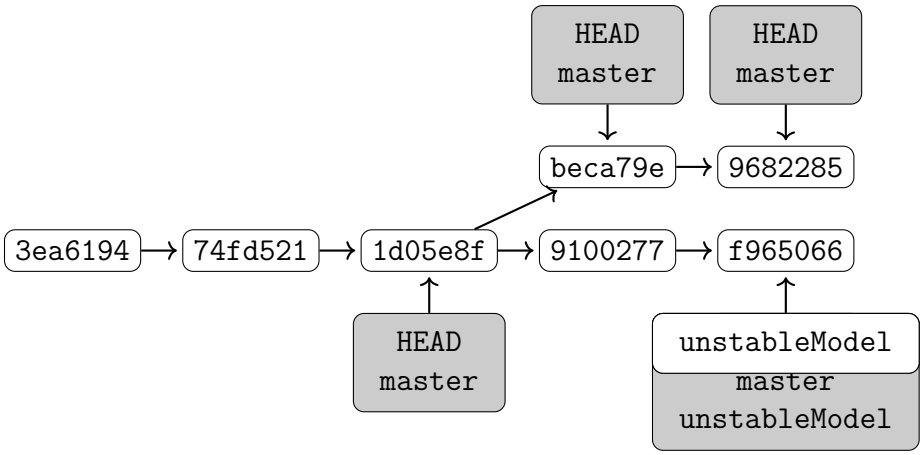
```
git commit -a -m "improved regression results (do not fully work)"
```

more changes and...

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

git log --oneline

```
f965066 added unstable 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 unstableModel
```

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

```
git checkout 1d05e8f
```

```
Previous HEAD position was f965066 added unstable model (does not fully work yet)
HEAD is now at 1d05e8f draft conclusion
```

do more work...

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

do even more work...

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

eventuelly we want to join the two branches:

```
git merge unstableModel
```

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

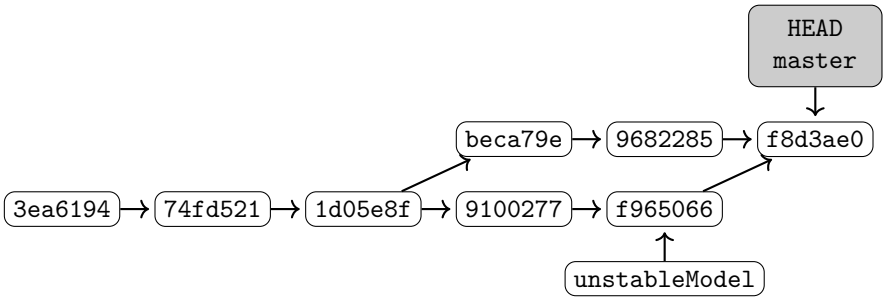
Normal merge conflict for 'test.Rnw':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (meld):
```

Now we can make detailed merge decisions in an editor.

```
git commit -m "merged unstableModel"
```

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`



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”

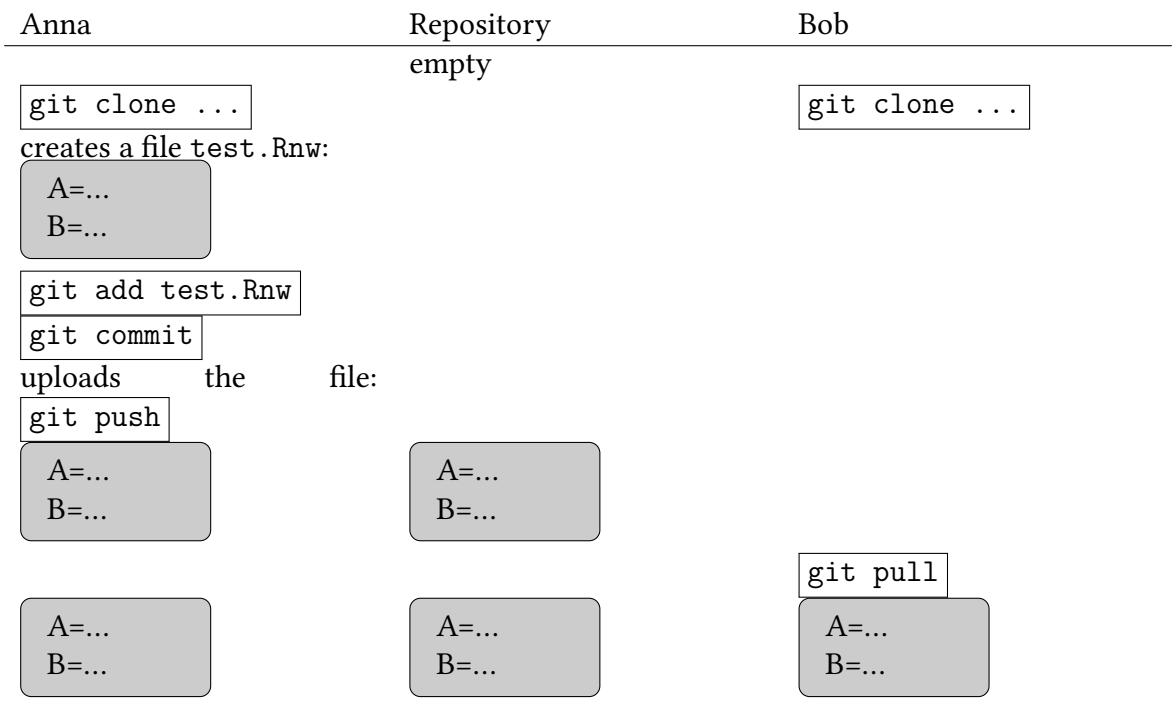
```
git --bare init
```

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

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

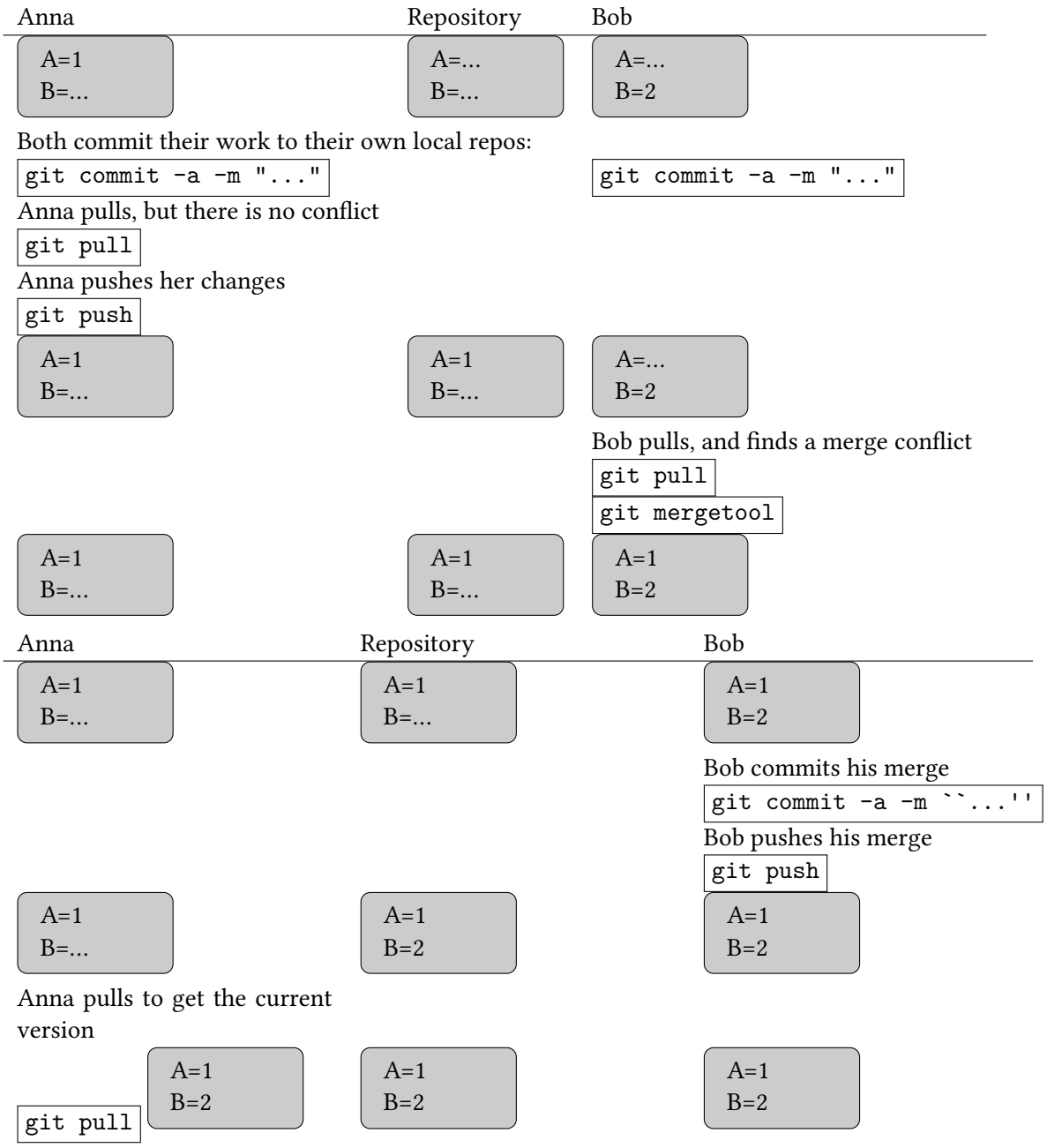
…or on a different machine via `ssh` (where `user` has access rights):

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



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



3.11 github

- Create account at `https://github.com`
- Generate ssh key: `ssh-keygen -t ed25519 -f ~/.ssh/github -C "your_email@example.`

  → `~/.ssh/github`

  → `~/.ssh/github.pub`

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAILu7NYDBPiwiSRJZz5J8DyPaMGQga3woLk7hDS3p9Yyq your_email@example.com
```

- In github / settings / SSH and GPG keys: add this public key
- On your computer, in `~/.ssh/config`

```
Host github
Hostname github.com
User git
IdentityFile ~/.ssh/github
```

Then, on your computer, test

```
ssh -T github
```

- github / home / create repository

  Owner / test

  [x] private

→ Quick setup:

`HTTPS: https://github.com/<username>/test.git`

`SSH: git@github.com:<username>/test.git`

- On your computer:

```
git clone ssh://github/<username>/test.git
```

or

```
git remote add origin ssh://github/<username>/test.git
```

3.12 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
```

```
f965066 added unstable 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 log -S "some text"
```

```
1d05e8f ... some text ...
```

`git show 1d05e8f`

`git grep` allows us to find files that contain a specific string.

`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 grep "some text"`

```
test.Rnw: ... some text ...
```

Now we know the file, namely `test.Rnw`. Next we use `git blame` to find changes in this file:

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

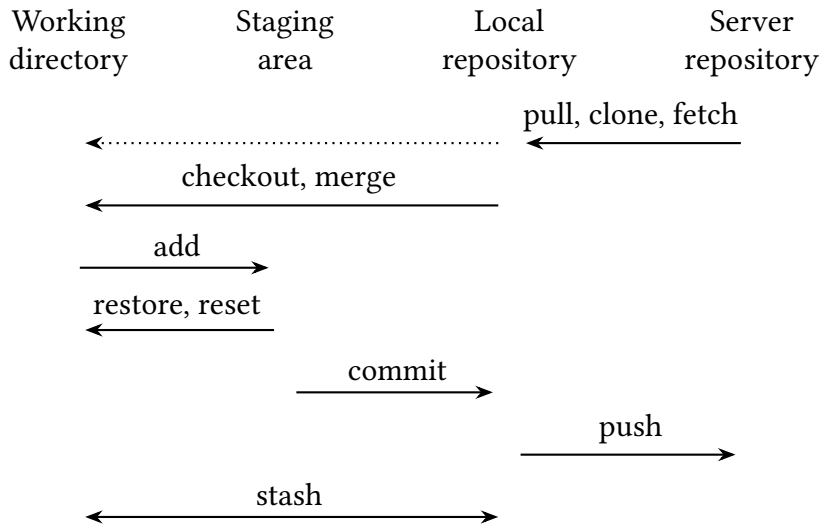
```
19eb9bac (w6kiol2 2016-06-17 ...) therefore important to study something specific which
dd0647f7 (w6kiol2 2016-06-21 ...) switched our focus to something else and continue with
```

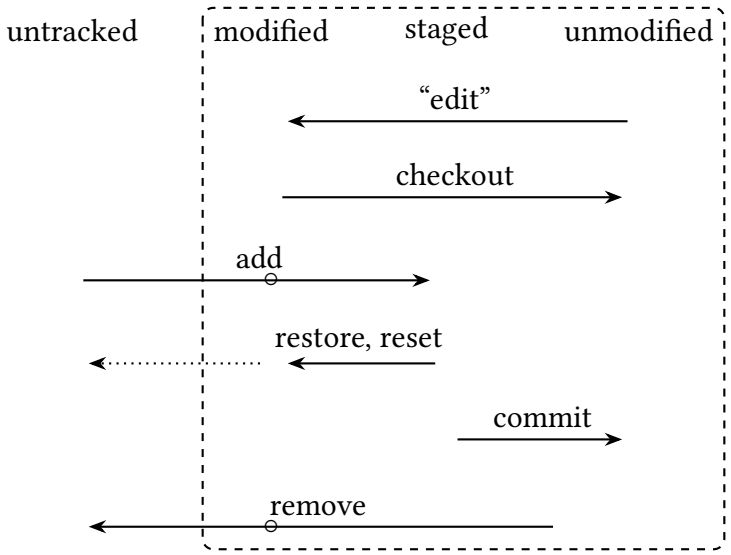
There is a range of GUIs that allow you to browse the commit tree.

Try, e.g., `gitk`

3.13 git – Data integrity

- Data integrity (checksums)
- Data integrity (persistent commits)






```
git clone ssh://user@my.server.org/path/to/repository
cd repository
git remote -v
```

```
origin ssh://user@my.server.org/path/to/repository (fetch)
origin ssh://user@my.server.org/path/to/repository (push)
```

```
git remote add altRep ssh://user@my.altserver.org/path/to/repository
git remote -v
```

```
origin ssh://user@my.server.org/path/to/repository (fetch)
origin ssh://user@my.server.org/path/to/repository (push)
altRep ssh://user@my.altserver.org/path/to/repository (fetch)
altRep ssh://user@my.altserver.org/path/to/repository (push)
```

```
git fetch altRep
git push altRep
```

3.14 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)

3.15 Limitations

3.15.1 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...).

- 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.

3.15.2 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.

3.16 Usual workflow with git

Here, we assume that there is already a central repository that is used by all collaborators.

- Initially, `git clone`.
- From then on
  `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
- Create repository on USB stick

  ```
  cd /media/user/STICK1
  git --bare init
  ```

- Go to some working directory and clone the empty repository:

  ```
  cd /home/user/Work/
  git clone /media/user/STICK1
  cd STICK
  git remote -v
  ```

- Create a file in this directory.

  ```
  git add someFile.Rnw
  git commit -m "add my first file"
  git push
  ```

- Unmount USB stick and let your neighbour clone the repository, too.

3.17 Versions of software

Python

```
python -m venv ...
```

R

- `renv::init()`

  creates the following:

  - `renv/`
  - `renv.lock` (should be under version control)
  - `.Rprofile`

- `renv::snapshot()`
- `renv::restore()`
- `renv::install()` → `renv::snapshot()`
- `renv::update()` → `renv::snapshot()`

- Dependency management / version conflicts / clean global environment.
- Reproducibility.
- Security (vulnerabilities affect only the virtual environment).
- Testing (experimental features/bugs affect only the virtual environment)

3.18 Exercise

3.18.1 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`

3.18.2 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

- 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_240715.R")
source("clean_data_240715.R")
source("create_figures_240715.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_240715.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.
- 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)

```
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:

```
# 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)
}
```

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.

- 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_240715.R")
getAndCleanData() # takes a lot of time
save(cleanedData,file="cleanedData240715.Rdata")

load("cleanedData240715.Rdata")
doBootstrap() # takes a lot of time
save(bsData,file="bsData240715.Rdata")

load("cleanedData240715.Rdata")
load("bsData240715.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() {
  ...
}
```

```
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
- ...

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

```
# scriptExample240715.R
#
# the purpose of this script is to illustrate the use of
#                                                  comments
#
# first version: 230715
# this version:  240715
# last change by: Oliver
# requires:      test230715.Rdata, someFunctions230715.R
# provides:      ...
#
set.seed(123)
```

- 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

```
  ...
# 24/07/21: 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

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

- Variables names

  short but not too short

  ```
  lm ( otherinvestment ~ trustworthiness + inequalityaversion + sexOfProposer + ageOfPropos
  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 ...
```

```
 $ 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)
```

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:

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

	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...

- 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: 0x55b359ecdd68>
  .. ..- 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"
```

```
 $ 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: 0x55b359ecdd68>
  .. .. ..- 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

car::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
```

```
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
[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
```

```
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

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,package="Ecdat")
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
```

```
[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
```

```
------------------------------------------------------------
: 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
------------------------------------------------------------
```

```
: 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
```

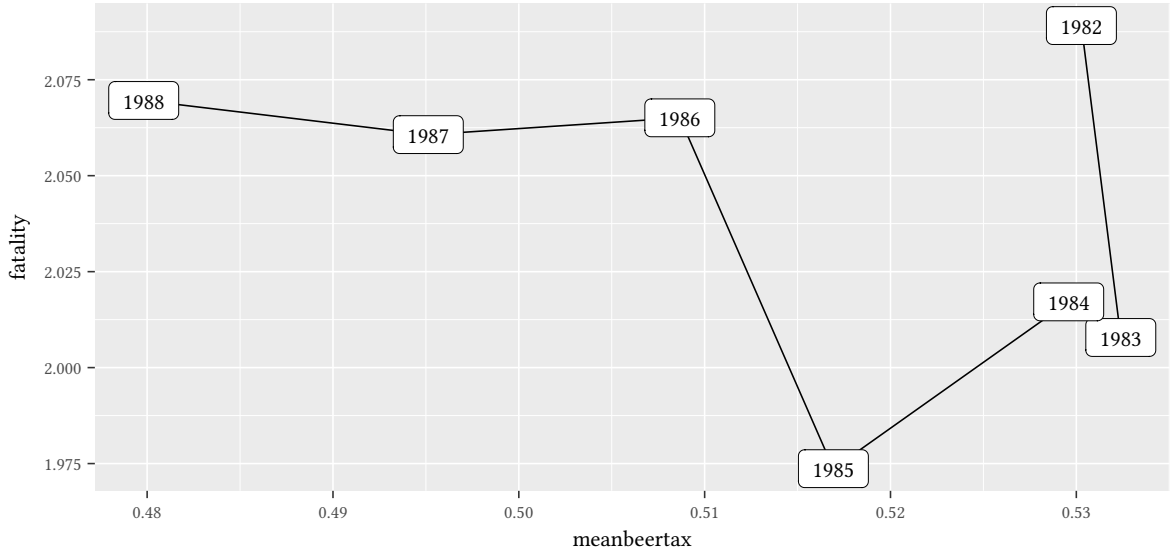
by does not return a vector but an object of class by. If we actually need a vector we have to use c and sapply.

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
```

```
library(ggplot2)
byObj |>
    sapply(c) |>
    t() |>
    data.frame() |>
    ggplot(aes(x=meanbeertax,y=fatality,label=year)) +
    geom_path() + geom_label()
```



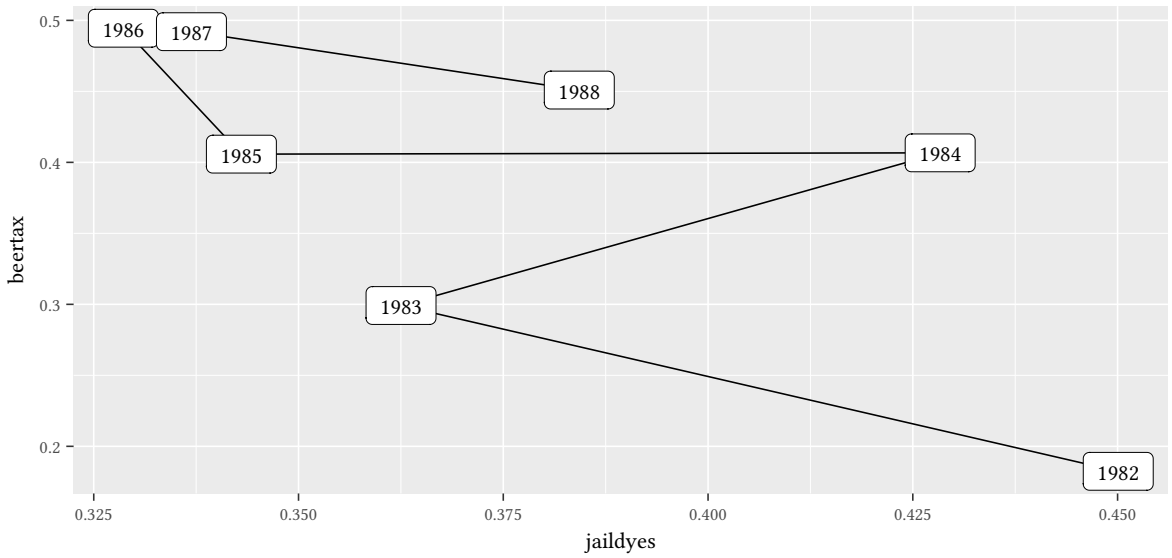
We can do more complicated things in by. In the following example we estimate a regression. To get only the coefficients from the regression (and not fitted values, residuals, etc.) we use the extractor function `coef`.

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

```
                  1982         1983         1984         1985         1986
(Intercept)    1.9079924    1.7503870    1.6768093    1.6567128    1.7108657
beertax        0.1824028    0.2991742    0.4066922    0.4057889    0.4944595
jaildyes       0.4500807    0.3625151    0.4283417    0.3430232    0.3286131
year        1982.0000000 1983.0000000 1984.0000000 1985.0000000 1986.0000000
                  1987         1988
(Intercept)    1.7188081    1.7411593
beertax        0.4920275    0.4509099
jaildyes       0.3369277    0.3842788
year        1987.0000000 1988.0000000
```

```
byObj |>
    sapply(c) |>
    t() |>
    data.frame()  |>
```

```
ggplot(aes(x=jaildyes,y=beertax,label=year)) +
geom_path() + geom_label()
```



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
```

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 ]
```

```
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

```
rbind(x,y)
dplyr::bind_rows(x,y)
```

(In Stata this is done by `append`)

- Matching two datasets (inner join)

```
merge(x,y)
dplyr::inner_join(x,y)
```

(In Stata this is done by `merge`)

- Joining two datasets (left join)

```
merge(x,y,all.x=TRUE)
dplyr::left_join(x,y)
```

(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@...

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/240716_060x.Rdata")
experiment1 <- subset(trustGS$subjects,Date=="240716_0601")
experiment2 <- subset(trustGS$subjects,Date=="240716_0602")
dim(experiment1)

[1] 108  14

dim(experiment2)

[1] 108  14

dplyr::bind_rows(experiment1,experiment2) |> dim()

[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 240716_0601         1      1          6               0
2 240716_0601         1      2          6               0
3 240716_0601         1      3          6               0
4 240716_0601         1      4          6               0
5 240716_0601         1      5          6               0
6 240716_0601         1      6          6               0

head(trustGS$subject)

         Date Treatment Period Subject Pos Group Offer Receive   Return GetBack
1 240716_0601         1      1       1   2     1     0   1.530 0.585990       0
2 240716_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.

```
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`.

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 |> head(4)

         Date Period Subject Pos Group Offer
1 240716_0601      1       1   2     1 0.000
2 240716_0601      1       2   2     4 0.000
3 240716_0601      1       3   1     5 0.495
4 240716_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 |> head(4)

         Date Period Group Offer.2 Subject.2   Offer.1 Subject.1
1 240716_0601      1     1       0         1 0.5100000        13
2 240716_0601      1     4       0         2 0.5580000         5
3 240716_0601      1     5       0         7 0.4950000         3
4 240716_0601      1     2       0         4 0.8422333         8
```

```
trustLong |>
    tidyr::pivot_wider(id_cols=c("Date","Period","Group"),
                       values_from=c("Offer","Subject"),names_from="Pos") -> trustWide2
trustWide2 |> head(4)

# A tibble: 4 x 7
  Date        Period Group Offer_2 Offer_1 Subject_2 Subject_1
  <chr>        <dbl> <dbl>   <dbl>   <dbl>     <dbl>     <dbl>
1 240716_0601      1     1       0   0.51          1        13
2 240716_0601      1     4       0   0.558         2         5
3 240716_0601      1     5       0   0.495         7         3
4 240716_0601      1     2       0   0.842         4         8
```

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

                         Date Period Group Pos Offer Subject
240716_0601.1.1.2 240716_0601      1     1   2     0       1
```

```
240716_0601.1.4.2 240716_0601      1     4   2     0       2
240716_0601.1.5.2 240716_0601      1     5   2     0       7
240716_0601.1.2.2 240716_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.

The same can be done with `tidyr::pivot_longer`:

```
trustWide2 |>
    tidyr::pivot_longer(cols=starts_with(c("Offer","Subject")),names_to=c(".value","Pos"),
                        names_sep="_")

# A tibble: 432 x 6
   Date        Period Group Pos   Offer Subject
   <chr>        <dbl> <dbl> <chr> <dbl>   <dbl>
 1 240716_0601      1     1 2     0           1
 2 240716_0601      1     1 1     0.51       13
 3 240716_0601      1     4 2     0           2
 4 240716_0601      1     4 1     0.558       5
 5 240716_0601      1     5 2     0           7
 6 240716_0601      1     5 1     0.495       3
 7 240716_0601      1     2 2     0           4
 8 240716_0601      1     2 1     0.842       8
 9 240716_0601      1     3 2     0          16
10 240716_0601      1     3 1     0.751       6
# i 422 more rows
```

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


         Date Period Group         1 2
1 240716_0601      1     1 0.5100000 0
2 240716_0601      1     2 0.8422333 0
3 240716_0601      1     3 0.7510000 0
4 240716_0601      1     4 0.5580000 0
5 240716_0601      1     5 0.4950000 0
6 240716_0601      1     6 0.6910000 0
7 240716_0601      1     7 0.5430000 0
8 240716_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})))

   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: 0x55b363712318>
```

```
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))

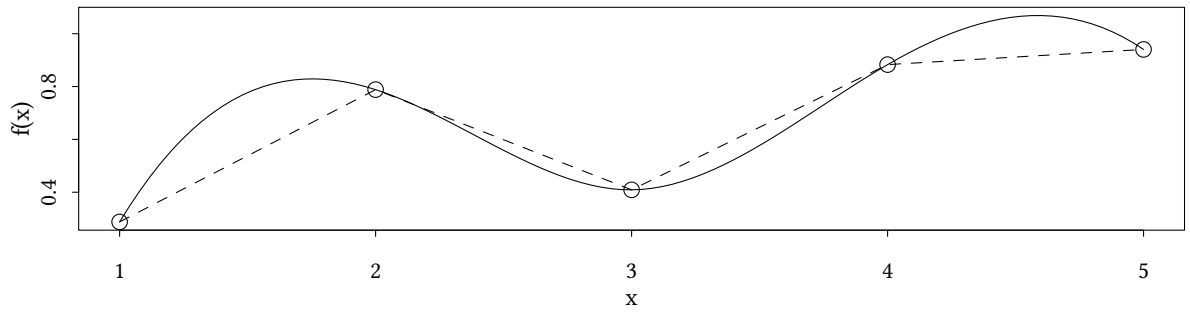
$exponent
[1] 2
```

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)
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
```

We could also say:

```
x |> var() |> sqrt()

[1] 3.02765
```

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

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

# A tibble: 6 x 4
# Groups:   am [2]
     am   cyl `mean(disp)` `mean(hp)`
  <dbl> <dbl>        <dbl>      <dbl>
1     0     4        136.        84.7
2     0     6        205.       115.
3     0     8        358.       194.
4     1     4         93.6       81.9
5     1     6        155        132.
6     1     8        326        300.
```

We could store intermediate results in a variable (xx)

```
xx<-filter(mtcars,!is.na(am) & !is.na(cyl))
xx2<-group_by(xx,am,cyl)
summarize(xx2,mean(disp),mean(hp))

# A tibble: 6 x 4
# Groups:   am [2]
     am   cyl `mean(disp)` `mean(hp)`
  <dbl> <dbl>        <dbl>      <dbl>
1     0     4        136.        84.7
2     0     6        205.       115.
3     0     8        358.       194.
4     1     4         93.6       81.9
5     1     6        155        132.
6     1     8        326        300.
```

We could combine all this into a single chain of functions:
The |> operator allows us to chain functions more transparently.

```
mtcars |>
  filter(!is.na(am), !is.na(cyl))  |>
  group_by(am,cyl) |>
  summarise(mean(disp),mean(hp))

# A tibble: 6 x 4
# Groups:   am [2]
     am   cyl `mean(disp)` `mean(hp)`
  <dbl> <dbl>        <dbl>      <dbl>
1     0     4        136.        84.7
2     0     6        205.       115.
3     0     8        358.       194.
4     1     4         93.6       81.9
5     1     6        155        132.
6     1     8        326        300.
```

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
    - 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] "240716_0601.xls" "240716_0602.xls" "240716_0603.xls" "240716_0604.xls"
```

```
trustGS <- zTreeTables(files)

reading  240716_0601.xls ...
Doing: globals
Doing: subjects
*** 240716_0602.xls is file 2 / 4 ***
reading  240716_0602.xls ...
Doing: globals
Doing: subjects
*** 240716_0603.xls is file 3 / 4 ***
reading  240716_0603.xls ...
Doing: globals
Doing: subjects
*** 240716_0604.xls is file 4 / 4 ***
reading  240716_0604.xls ...
Doing: globals
Doing: subjects
```

save in R-format:

```
save(trustGS,zTreeTables,file="240716_060x.Rdata")
```

save in Stata-format:

```
xx<-with(trustGS,merge(globals,subjects))
write.dta(xx,file="240716_060x.dta")
```

save in Stata-13 format:

```
save.dta13(xx,file="240716_060x.dta13")
```

save as csv:

```
write.csv(xx,file="240716_060x.csv")
```

```
fn<-list.files(pattern="240716_060x\\.[^.]*")
xtable(cbind(name=fn,size=file.size(fn)))
```

	name	size
1	240716_060x.csv	28178
2	240716_060x.dta	59300
3	240716_060x.dta13	60354
4	240716_060x.Rdata	18733

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/240716_060x.Rdata")
```

To retrieve them, we use `load`

```
load("data/240716_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.

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/240716_060x.dta")
```

```
sta2 <- memisc::Stata.file("data/240716_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  "240716_0601" "240716_0601" "240716_0601" "240716_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:

```
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)
  ..@ 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/240716_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"
 [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: 0x55a4df38ca10>
attr(,"file.name")
[1] "data/240716_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"

$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`.

```
memisc::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
   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

        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'

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

   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

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

   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/240716_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/240716_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)
```

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/240716_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	128904
xls	454656
dta	59300
dta13	60354
csv	28178
Rdata	18733

7.3 Checking Values

```
load("data/240716_060x_C.Rdata")
```

7.3.1 Range of values

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

⋮

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

  trustC.Offer 'trustor's offer'

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

```
   Storage mode: double
   Measurement: interval

   Values       N Percent

   NA M       216    50.0

        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

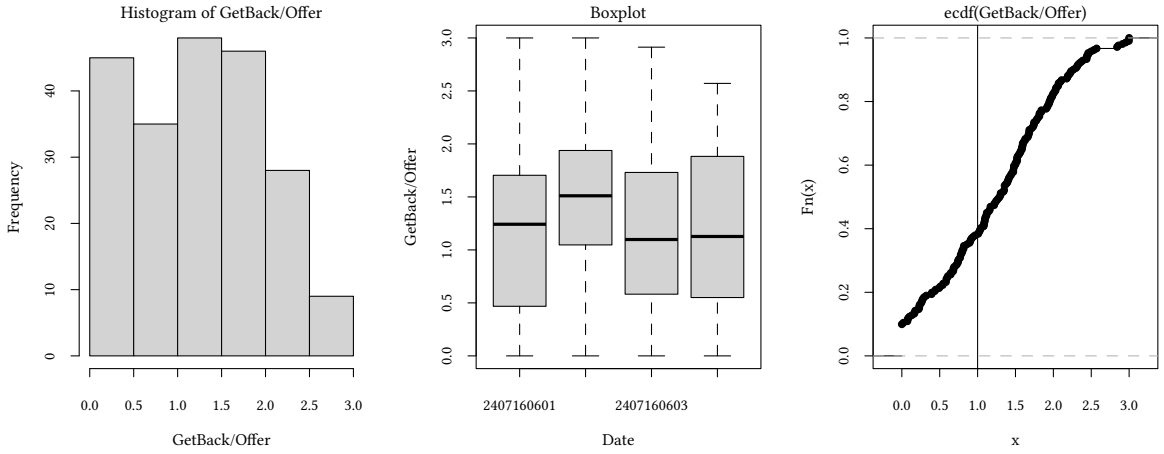
   Values and labels     N Valid Total

    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
```

7.3.2 (Joint) distribution of values

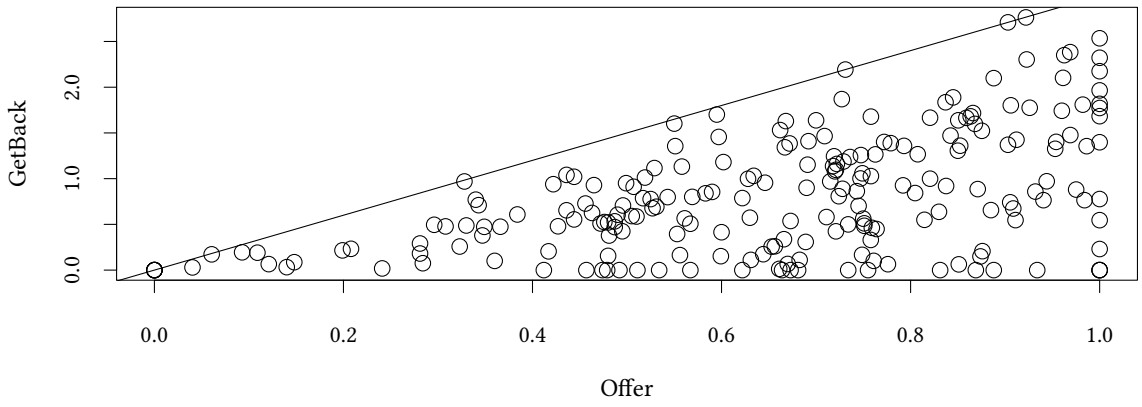
Basic plots

```
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)
```



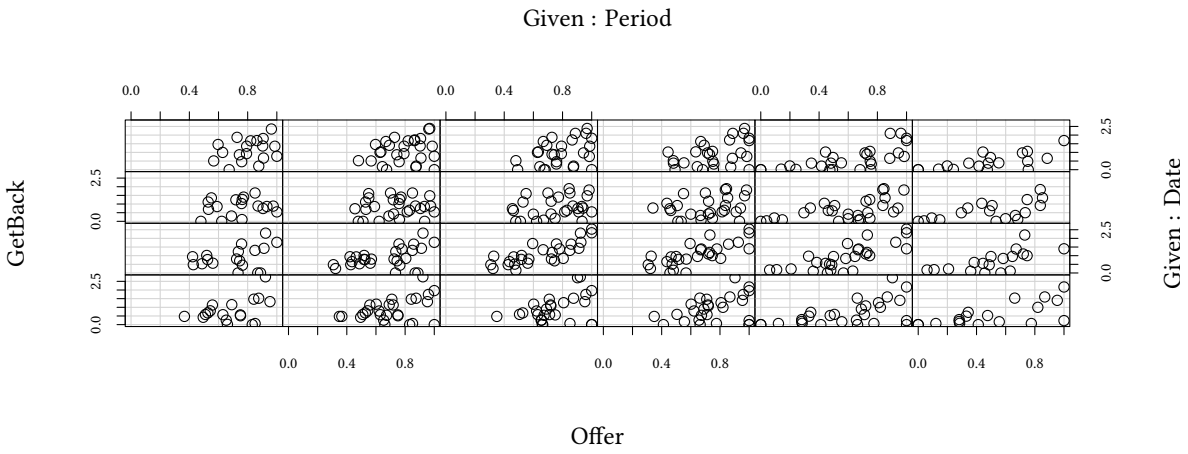
Joint distributions First pool all data:

```
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:

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



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

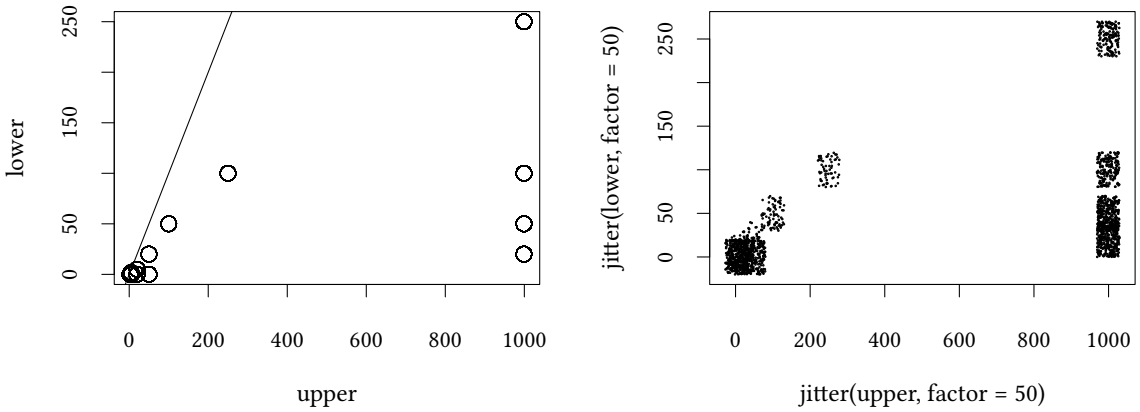
```
data(Kakadu)
nrow(Kakadu)

[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)
```



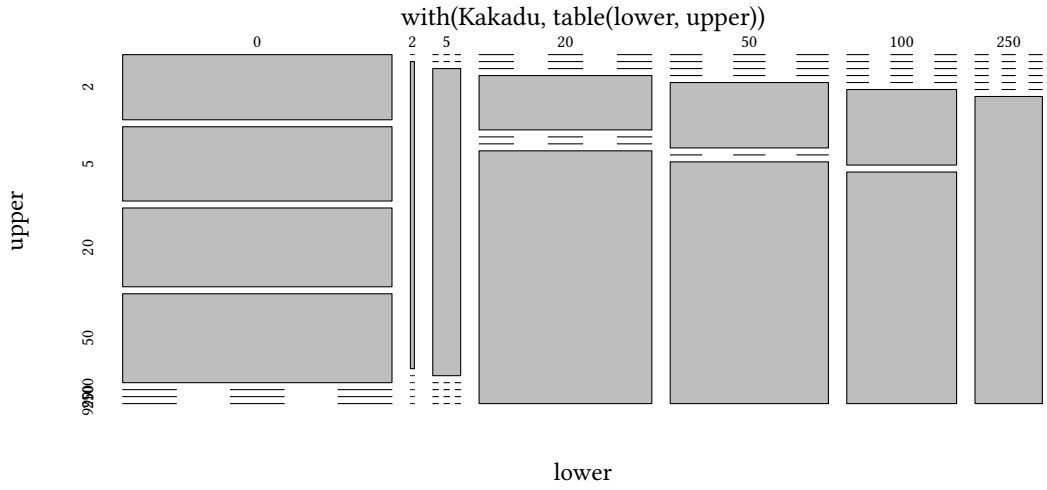
With such a large number of observations, and so few categories, a table might be more informative

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

lower \ upper	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:

```
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/240716_060x.Rdata")

            data/240716_060x.Rdata
"8e07ccaa6ec8e3b4696accd310aac123"
```

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 140.

- 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

- 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/240716_060x.Rdata")
trust <- within(with(trustGS,merge(globals,subjects)), {
  memisc::description(Pos)<- "(1=trustor, 2=trustee)"
  memisc::description(Offer)<- "trustor's offer"
  memisc::description(Receive)<- "amount received by trustee"
  memisc::description(Return)<- "amount trustee sends back to
                                              trustor"
  memisc::description(GetBack)<- "amount trustor receives back
                                          from trustee"
  memisc::description(country)<- "country of origin"
  memisc::description(sex)<- "participant's sex (1=male, 2=female)"
  memisc::description(siblings)<- "number of siblings"
  memisc::description(age)<- "true age"
  })
memisc::codebook(memisc::data.set(trust))
attr(trust,"annotation")<-"Note: 240716_0601 was a pilot,..."
memisc::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:

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

Better:

```
memisc::description(uncondSend) <-  "how much to send without binding contract"
memisc::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).

```
memisc::codebook(trustC$sex)
```

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

   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
```

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

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

7.7 Recoding data

7.7.1 Replacing meaningless values by missings

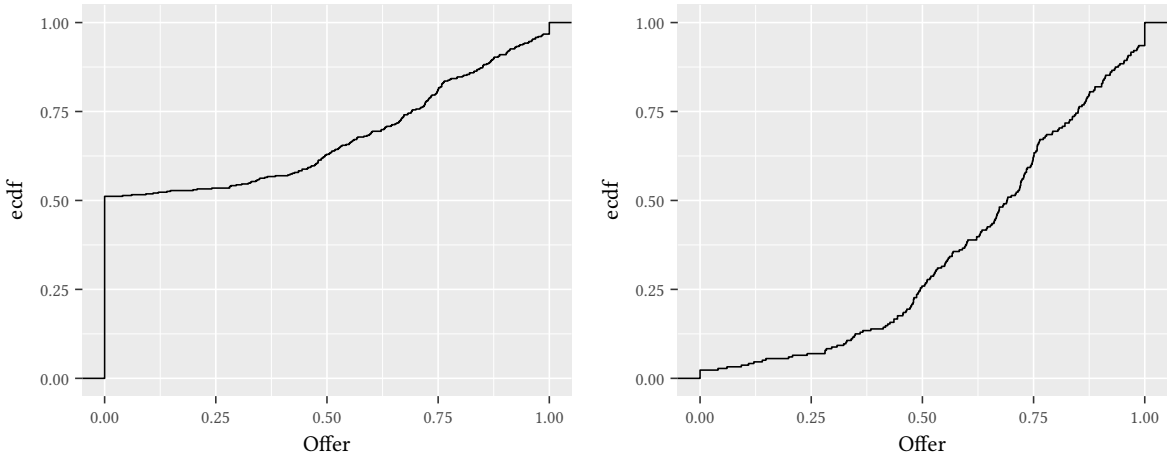
In our trust game not all players have made all decisions. z-Tree coded these "decisions" as zero. This can be misleading. Better code them as missing.

```
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/240716_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*.

```
library(ggplot2)
trust |>  ggplot(aes(x=Offer)) + stat_ecdf() -> p1
trustC |> ggplot(aes(x=Offer)) + stat_ecdf() -> p2
gridExtra::grid.arrange(p1,p2,ncol=2)
```




```
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

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 Data wrangling

8.1 Scope

Working with data...

- Data cleaning
- Data wrangling
- Descriptives, tests, estimates, graphs

In this part of the course...

- Work with text, regular expressions.
- Dates

- Work with HTML, XML, XPath.
- Work with unusual datasets.
- Application/exercise.

8.2 Regular expressions

- How to find things?

Very often we need to find something is a large amount of text. Here, regular expressions help us describing what we want to find and what to do with it:

- Goal:
  - Find structure (in text)
  - Replace with a different structure.
- Examples:
  - Find email, zip-code,…
- Applications:
  - Shell commands: `sed`, `grep`,…
  - R: `sub`, `gsub`, `grep`, `grepl`,…
  - Python: `re.search`, `re.sub`,…
  - ⋮
- They are daunting.
- They can be very helpful!

Regular expressions in R

- Find things:

  `grep`, `grepl`

  For approximate finds:

  `stringdist::afind`
- Replace things:

  `sub`, `gsub`,

  `stringr::str_replace`, `stringr::str_replace_all`, `stringr::str_extract_all`
- Splitting things up:

  `strsplit`

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown","Susan Smith")
grep("Susan",text)
```

```
[1] 3 5
```

```
grep("Susan",text,value=TRUE)
```

```
[1] "Susan Jones" "Susan Smith"
```

```
grepl("Susan",text)
```

```
[1] FALSE FALSE  TRUE FALSE  TRUE
```

```
sub("Susan","Sally",text)
```

```
[1] "Jane Smith"  "John Taylor" "Sally Jones" "Jack Brown"  "Sally Smith"
```

Examples with fixed strings

Elements of regular expressions

- History:
  - Stephen Cole Kleene (1951). “Representation of Events in Nerve Nets and Finite Automata”.
  - Ken Thompson (1968). “Programming Techniques: Regular expression search algorithm.”

  ↓ Today:
- Most characters (`01234...ABCDE...`) just match themselves.
- Sets of characters are denoted by `[...]`. Characters within the brackets can be...
  - listed individually `[,;:\.]`
  - ranges `[a-z0-9]`
  - complemented `[^a-z]`
  - any character: `.`
- Alternatives: `|`
  - `Susan|Sally`
- Groups: `(...)`

  - `(The|That) (dog|cat)`

- Repetition: `* + ?`
  - `[0-9]+`

Most characters (`01234...ABCDE...`) just match themselves, except...

`.`	any character	
`^`	start of the string	`^Susan`
`$`	end of the string	`Susan$`

Characters that match repetitions of preceding RE:

`*`	0 or more repetitions of preceding RE	`.*`	`A*`	`[0-9]*`	`(Susan)*`
`+`	1 or more repetitions of preceding RE	`.+`	`A+`	`[0-9]+`	`(Susan)+`
`?`	0 or 1 repetitions of preceding RE	`.?`	`A?`	`[0-9]?`	`(Susan)?`
`{m}`	exactly m matches of preceding RE	`A{3}`	`AAA`		
`{m,n}`	m to n matches of preceding RE	`A{3,5}`	`AAAA?A?`		

Repetitions are greedy. The can be followed by `?` to make them lazy:

```
sub(".*A","-","xyzABCAgg")

[1] "-gg"

sub(".*?A","-","xyzABCAgg")

[1] "-BCAgg"
```

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown")
grep("s.n",text)

[1] 3

sub("s.n","-",text)

[1] "Jane Smith"  "John Taylor" "Su- Jones"   "Jack Brown"

sub("^..s","-",text)

[1] "Jane Smith"  "John Taylor" "-an Jones"   "Jack Brown"

sub("t.*$","-",text)

[1] "Jane Smi-"   "John Taylor" "Susan Jones" "Jack Brown"

sub("(?i)t.*$","-",text) ## (?i) ignores case

[1] "Jane Smi-"   "John -"      "Susan Jones" "Jack Brown"
```

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown")
sub("[a-z]*","-",text)
```

```
[1] "-Jane Smith"  "-John Taylor" "-Susan Jones" "-Jack Brown"
```

```
sub("[a-z]+","-",text)
```

```
[1] "J- Smith"  "J- Taylor" "S- Jones"  "J- Brown"
```

```
gsub("[a-z]*","-",text)
```

```
[1] "-J- -S-" "-J- -T-" "-S- -J-" "-J- -B-"
```

```
gsub("[a-z]?","-",text)
```

```
[1] "-J--- -S----"  "-J--- -T-----" "-S---- -J----" "-J--- -B----"
```

Repetition

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown")
gsub("[a-z]","-",text)    ## [a-z] is a "set"
```

```
[1] "J--- S----"  "J--- T-----" "S---- J----" "J--- B----"
```

```
gsub("[^a-z]","-",text)
```

```
[1] "-ane--mith"  "-ohn--aylor" "-usan--ones" "-ack--rown"
```

Sets

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown")
gsub("Susan|John","-",text)  ## .../... is an alternative
```

```
[1] "Jane Smith" "- Taylor"   "- Jones"    "Jack Brown"
```

```
gsub("(Susan|John)"," [\\1] ",text)  ## (...) is a group
```

```
[1] "Jane Smith"      " [John]  Taylor" " [Susan]  Jones" "Jack Brown"
```

Alternatives

```
text <- c("Jane T. Smith","John W. Taylor","Susan S. Jones","Jack Brown, Jr.","Susan Smith")
sub("Susan (.*)$","\\1, S.",text)
```

```
[1] "Jane T. Smith"   "John W. Taylor"  "S. Jones, S."    "Jack Brown, Jr."
[5] "Smith, S."
```

```
gsub("(.)(.*)\\1","-\\2-",text)
```

```
[1] "Jane T. Smith"   "John W. Taylor"  "-usan -. Jones"  "-ack Brown, -r."
[5] "-usan -mith"
```

```
sub("(.*) (.*)$","\\2, \\1",text)   ## greedy \1
```

```
[1] "Smith, Jane T."   "Taylor, John W."  "Jones, Susan S."  "Jr., Jack Brown,"
[5] "Smith, Susan"
```

```
sub("(.*?) (.*)$","\\2, \\1",text)  ## lazy \1
```

```
[1] "T. Smith, Jane"   "W. Taylor, John"  "S. Jones, Susan"  "Brown, Jr., Jack"
[5] "Smith, Susan"
```

Groups

```
text <- "Susan Jones, susan.jones@some.domain.com, https://www.susan-jones.net/,
         01234 Somecity,..."
sub("^.*?([a-z\\.]+@[a-z\\.]+).*$","\\1",text)
```

```
[1] "susan.jones@some.domain.com"
```

```
sub("^.*(https?://[a-z\\.-]+/?).*$","\\1",text)
```

```
[1] "https://www.susan-jones.net/"
```

```
sub("^.*[^0-9]([0-9]{4,5})[^0-9a-zA-Z]+([a-zA-Z]+).*$","\\2, \\1",text)
```

```
[1] "Somecity, 01234"
```

Example

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown")
stringr::str_replace_all(text,c("Jane"="Joan","John"="Jack","Jack"="Tim"))
```

```
[1] "Joan Smith"  "Tim Taylor"  "Susan Jones" "Tim Brown"
```

```
stringr::str_count(text,c("J|a|o"))
```

```
[1] 2 4 3 3
```

```
stringr::str_extract_all(text,c("J|a|o"))
```

```
[[1]]
[1] "J" "a"

[[2]]
[1] "J" "o" "a" "o"

[[3]]
[1] "a" "J" "o"

[[4]]
[1] "J" "a" "o"
```

The stringr library

```
text <- c("Mary Jane Smith","John Taylor","Susan Jones","Jack-Brown")
strsplit(text," ")
```

```
[[1]]
[1] "Mary"  "Jane"  "Smith"

[[2]]
[1] "John"   "Taylor"

[[3]]
[1] "Susan" "Jones"

[[4]]
[1] "Jack-Brown"
```

```
strsplit("Jack-Brown","[ -]")
```

```
[[1]]
[1] "Jack"  "Brown"
```

```
##         123456789 123456789 123456789 12
text <- c("Today, Anne and John do the work",
          "Tomorrow, Susan and Joan are singing")
```

```
stringdist::afind(text,
                  c("Anna","John","Liz"),
                  value=FALSE,method="jw")
```

```
$location
     [,1] [,2] [,3]
[1,]    7   17    1
[2,]   15   21   30

$distance
          [,1]      [,2]      [,3]
[1,] 0.1666667 0.0000000 1.0000000
[2,] 0.2777778 0.1666667 0.4444444
```

```
stringdist::afind(text,
                  c("Anna","John","Liz"),
                  value=FALSE,method="soundex")
```

```
$location
     [,1] [,2] [,3]
[1,]    8   17    1
[2,]   14   21    1

$distance
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    0    0    1
```

Splitting strings

8.3 Date and Time

```
as.POSIXct("01-02-03")
```

```
[1] "1-02-03 LMT"
```

```
as.POSIXct("01-02-03") |> class()
```

```
[1] "POSIXct" "POSIXt"
```

```
as.POSIXct("01-02-03") |> unclass()
```

```
[1] -62132748808
attr(,"tzone")
[1] ""

as.POSIXct("01-02-03") |> format("%d.%m.%Y")

[1] "03.02.1"

as.POSIXct("01-02-03",format="%d-%m-%y",tz="UCT")

[1] "2003-02-01 UTC"

as.POSIXct("01-02-03",format="%d:%m:%y",tz="UCT")

[1] NA

as.POSIXct("01-02-03",format="%y-%m-%d",tz="UCT")

[1] "2001-02-03 UTC"
```

```
date.pct <- as.POSIXct("01-02-03"  ,
                       format="%d-%m-%y",tz="UCT")
format(date.pct,"%Y")

[1] "2003"

format(date.pct,"%A, %d %B %Y")

[1] "Saturday, 01 February 2003"

format(date.pct,"%s seconds since the epoch")

[1] "1044054000 seconds since the epoch"

unclass(date.pct)

[1] 1044057600
attr(,"tzone")
[1] "UCT"
```

- POSIX time is stored (at least) as a 32 bit signed integer.

  (modern desktop computers use 64 bits, but embedded systems (routers, cars…) may not.)

```
as.POSIXct(0,origin="1970-01-01")

[1] "1970-01-01 01:00:00 CET"

as.POSIXct(-2^31,origin="1970-01-01")
```

```
[1] "1901-12-13 21:45:52 CET"

as.POSIXct(2^31,origin="1970-01-01")

[1] "2038-01-19 04:14:08 CET"
```

- POSIX time "ignores" leap seconds.

Flexible formats:

```
date <- "01-02-03"
date2 <- "02:03:04"
as.POSIXct(c(date,date2),format="%d-%m-%y",tz="UCT")

[1] "2003-02-01 UTC" NA

##
library(lubridate)
parse_date_time(c(date,date2),orders="dmy",
                locale = "de_DE.UTF-8")

[1] "2003-02-01 UTC" "2004-03-02 UTC"

dmy(c(date,date2))

[1] "2003-02-01" "2004-03-02"
```

Arithmetic with dates:

```
dmy(date2)-dmy(date)

Time difference of 395 days

mean(c(dmy(date) ,dmy(date2)))

[1] "2003-08-17"

mean(c(dmy(date) ,dmy(date2))) + dweeks(2)

[1] "2003-08-31 12:00:00 UTC"

dmy(date) + dweeks(3)/2

[1] "2003-02-11 12:00:00 UTC"
```

8.4 Python

```
import pandas as pd
ModuleNotFoundError: No module named 'pandas'

text = pd.Series(["Mary Jane Smith","John Taylor","Susan Jones","Jack-Brown"])

NameError: name 'pd' is not defined

text.str.count("Susan")

NameError: name 'text' is not defined

text.str.contains("Susan")

NameError: name 'text' is not defined
```

```
text.str.replace("Susan","Sally")

NameError: name 'text' is not defined

text.str.split(" ")

NameError: name 'text' is not defined
```

8.5 Working with HTML

Much of the information on the internet is represented as HTML. Here is some HTML:

- HTML = HyperText Markup Language (Tim Berners-Lee, 1980)
- XML = Extensible Markup Language (1998)
- other markup languages, like LaTeX, markdown, ...

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head>
    <title>My webpage</title>
  </head>
  <body>
    <div class="heading">
      <h1>This is a Title</h1>
    </div>
    <div class="abstract">
      Lorem ipsum dolor sit amet, consectetur adipisici elit,...
    </div>
    <div class="intro"><h2>Introduction</h2>
      Ut enim ad minim <a href="https://www.veniam.org/"><i>veniam</i></a>,...
      <ul><li class="myItems">Item one.</li><li class="otherItems">Item two.</li></ul>
    </div>
```

```
  </body>
</html>
```

- HTML is (for the browser and the server) the internal representation of a page.
- A browser renders the HTML according to the requirements of the user interface (screen of different sizes, printed page, braille terminal, screen reader…).

Elements `<html>`, `<head>`, `<body`, `<div>`, `<h2>`, `<ul>`, `<li>`…

Elements and tags

	Start-tag	End-tag
The entire page	`<html>`	`</html>`
The body of the page	`<body>`	`</body>`
Heading	`<h2>`	`</h2>`
Block of text	`<div class=...>`	`</div>`
Unordered list	`<ul>`	`</ul>`
List item	`<li>`	`</li>`
Anchor	`<a href=...>`	`</a>`

Attributes `class='intro'`, `href=…`,

Elements and attributes tell us more about the type of the information.
Example:

```
<img src='https://www.some.dom/P123.jpeg' alt='A picture of a dog'>
```

Reading HTML in R

- If we want to read many pages into R, we need a command.

  → `httr::GET` reads pages and stores them somehow.

- We also want to translate these pages into
  - text.
  - a tree.

  `httr::content` does the translation.
  - `httr::content(...,as='text')`
  - `httr::content(...,as='parsed')`

  The `xml2` library provides commands to navigate the tree.

```
library(httr)
library(xml2)
thisPage <- httr::GET("https://www.some.domain.org/")
httr::content(thisPage,as="text")
xmlTree <- httr::content(thisPage,as="parsed")
```

```
xmlTree

{xml_document}
<html>
[1] <head>\n  <title>My webpage</title>\n</head>
[2] <body>\n  <div class="heading">\n    <h1>This is a Title</h1>\n  </div>\n ...

xml_children(xml_children(xmlTree)[2])

{xml_nodeset (3)}
[1] <div class="heading">\n  <h1>This is a Title</h1>\n</div>
[2] <div class="abstract">\n      Lorem ipsum dolor sit amet, consectetur adi ...
[3] <div class="intro"><h2>Introduction</h2>\n      Ut enim ad minim <a href= ...
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head>
    <title>My webpage</title>
  </head>
  <body>
    <div class="heading">
      <h1>This is a Title</h1>
    </div>
    <div class="abstract">
      Lorem ipsum dolor sit amet, consectetur adipisici elit,...
    </div>
    <div class="intro"><h2>Introduction</h2>
      Ut enim ad minim <a href="https://www.veniam.org/"><i>veniam</i></a>,...
      <ul><li class="myItems">Item one.</li><li class="otherItems">Item two.</li></ul>
    </div>
  </body>
</html>
```

8.6 XPath

- HTML pages can have a complicated structure.
- The tree-representation can a complicated structure, too.

We may need a tool to find things in these trees.

→ *XPath* is a notation to describe search requests for XML trees.

XPath is a notation to select nodes in an XML or HTML document.

- `xml_find_all(document,'//a')`
  finds all `<a...> ... </a>` nodes (all hyperlinks) in that document.

```
xml_find_all(xmlTree,"//a")

{xml_nodeset (1)}
[1] <a href="https://www.veniam.org/">\n  <i>veniam</i>\n</a>
```

```
node <- xml_find_all(xmlTree,"//a")
xml_children(node)

{xml_nodeset (1)}
[1] <i>veniam</i>

xml_parent(node)

{xml_nodeset (1)}
[1] <div class="intro"><h2>Introduction</h2>\n      Ut enim ad minim <a href= ...
```

```
xml_children(xml_parent(node))

{xml_nodeset (3)}
[1] <h2>Introduction</h2>
[2] <a href="https://www.veniam.org/">\n  <i>veniam</i>\n</a>
[3] <ul>\n  <li class="myItems">Item one.</li>\n  <li class="otherItems">Item ...

xml_text(node)

[1] "veniam"

xml_attrs(node)

[[1]]
                     href
"https://www.veniam.org/"

xml_find_all(xmlTree,"//li[@class='myItems']")

{xml_nodeset (1)}
[1] <li class="myItems">Item one.</li>
```

Here is again the HTML-page we have seen above:

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
```

```
  <head>
    <title>My webpage</title>
  </head>
  <body>
    <div class="heading">
      <h1>This is a Title</h1>
    </div>
    <div class="abstract">
      Lorem ipsum dolor sit amet, consectetur adipisici elit,...
    </div>
    <div class="intro"><h2>Introduction</h2>
      Ut enim ad minim <a href="https://www.veniam.org/"><i>veniam</i></a>,...
      <ul><li class="myItems">Item one.</li><li class="otherItems">Item two.</li></ul>
    </div>
  </body>
</html>
```

How would we search for text on that page?

```
"https://www.some.domain.org/" |>
   httr::GET() |>
   httr::content(as="parsed") -> xmlTree
```

```
xmlTree |> xml_find_all("//*[contains(text(),'veniam')]")

{xml_nodeset (1)}
[1] <i>veniam</i>

xmlTree |> xml_find_all("//*[contains(text(),'veniam')]") |> xml_parent()

{xml_nodeset (1)}
[1] <a href="https://www.veniam.org/">\n  <i>veniam</i>\n</a>

xmlTree |> xml_find_all("//*[contains(text(),'veniam')]") |> xml_parent() |> xml_attrs()

[[1]]
                     href
"https://www.veniam.org/"
```

Here are some commands from the `xml2` library that help us navigating the tree representation of a HTML page:

- `xml_find_all(document,'//table')`
  finds all `<table...> ... </table>` nodes (all tables) in that document.
- `xml_find_all(document,'//table[@summary="Notes"]')`
  finds all `<table ... summary="Notes"> ... </table>` nodes
  (all tables with a summary attribute which has the value ”Notes”) in that document.

- `xml_find_all(document,'//p[@class="Relevant"')`

  finds all `<p ... class="Relevant"> ... </p>` nodes

  (all paragraphs with a class attribute which has the value "Relevant") in that document.

- `xml_find_all(document,"//*[contains(text(),'veniam')]")`

  finds all nodes that contain the text `veniam`.

- `xml_children(node)`

  returns all children of a node (e.g. if the node is a table (`<table>...</table>`), then the children are often the rows of that table. If the node is a list (`<ul><li>...</li><li>...</li`
  then the children are the items of that list...).

- `xml_parent(node)`

  returns the parent of that node (e.g. if a node is a row of table, then the parent could be the entire table...)

- `xml_text(node)`

  A textual representation of the node.

- `xml_attrs(node)`

  All attributes of a node.

```
library(rvest)
"https://www.some.domain.org/" |>
   httr::GET() |>
   httr::content() |>
   rvest::html_table()
```

...provides a list with all tables on that page.

8.7 Working with funny data

Data is a list

```
lapply(data,function(x) {
    ...
    do_something_with_x()
    ...
    data.frame(a=...,b=...,c=...)
}) |>
    bind_rows()

sapply(data,function(x) {...})
```

Data is a grouped dataframe

```
data |>
group_by(variable) |>
mutate(...) |>      ## → df with same # of rows
summarise(...) |>   ## → df with 1 row
reframe(...) |>     ## → df with new # of rows
ungroup()
```

- The data is a list.

  Perform operation for each item in the list:

```
lapply(data,function(x) {...})  ## → list
bind_rows()              ## list → dataframe
sapply(data,function(x) {...})  ## → matrix
```

- The data is a dataframe.

  Perform operation for groups of data:

```
data |>
group_by(variable) |>
mutate(...) |>      ## → df with same # of rows
summarise(...) |>   ## → df with 1 row
reframe(...) |>     ## → df with new # of rows
ungroup()
```

Grouping

```
group_by(variable)
rowwise()           ## each row is one group
##
ungroup()
```

Perform operation for several variables in a data frame.

```
mutate(across(...variables...,function() {}))
```

8.8 Dataframes with unusual elements

Using dataframes makes it easier to keep things together (in one row) that belong together.

$$\begin{bmatrix} a_1 & b_1 & \cdots & x_1 \\ a_2 & b_2 & \cdots & x_2 \\ a_3 & b_3 & \cdots & x_3 \\ \vdots & \vdots & \ddots & \vdots \\ a_n & b_n & \cdots & x_n \end{bmatrix} \tag{1}$$

Each row is one record, e.g. one person, one name, one birth-date,…

But what if a record (a row) contains information about a variable number of things (languages spoken, books read, past employment, degrees…)

Columns in dataframes can contain as elements lists or even other dataframes:

$$\begin{bmatrix} a_1 & b_1 & \cdots & \{x_{11}, x_{12}\} \\ a_2 & b_2 & \cdots & \{x_2\} \\ a_3 & b_3 & \cdots & \{x_{31}, x_{32}, x_{33}\} \\ \vdots & \vdots & \ddots & \vdots \\ a_n & b_n & \cdots & \{x_{n1}, \ldots, x_{nk}\} \end{bmatrix} \tag{2}$$

To tell R that a `list()` is meant to be a single item:

```
mutate(a=f(b),x=I(list(...)))
data.frame(a=f(b),x=I(list(...)))
```

8.9 An application

Assume, we want to study data from the German Constitutional Court

(as in Engel (2022), "Lucky you: Your case is heard by a seasoned panel—Panel effects in the German Constitutional Court", etc.)

- Fetch court cases
- Extract for each case
  - date
  - judges
  - body
  - topic (econ, justice, political, person)
  - whether case was accepted

```
library(xml2)
library(dplyr)
library(parallel)
options(mc.cores=detectCores()/2)
library(httr)
library(ggplot2)
library(xtable)
```

```
if(!grepl("DataWrangling",getwd()))
    setwd("DataWrangling")
```

Are we welcome?

```
## get robots policy:
HOST <- "https://www.bundesverfassungsgericht.de/"
robots <- httr::GET(paste0(HOST,"robots.txt"))
cat(httr::content(robots))
```

```
User-agent: *
Disallow: /SiteGlobals/
Disallow: /DE/Service/
Disallow: /EN/Service/
Allow: /SiteGlobals/Functions/JavaScript/
Allow: /SiteGlobals/StyleBundles/
Allow: /SiteGlobals/Frontend/
Crawl-delay: 10
```

- Firefox: Right-click on element and select “Inspect Element”.
- Chrome: Right-click on element and select “Inspect”.

The BVerfG presents an overview of 10 decisions on one page. Each entry for one decision looks like

```
<ol id="searchResult" class="links">
   <li>1. ... <a href="SharedDocs/Entscheidungen...">
                    <span class="aktenzeichen">...</span>
                    <div>Beschluss vom ...</div>
               </a>
               <div class="kurztext">...</div>
   </li>
   <li>2. ...</li>
   ...
</ol>
```

The address of these pages follows a structure:
https://www.bundesverfassungsgericht.de/SiteGlobals/...?gtp=5403124_list%3D1
https://www.bundesverfassungsgericht.de/SiteGlobals/...?gtp=5403124_list%3D2
https://www.bundesverfassungsgericht.de/SiteGlobals/...?gtp=5403124_list%3D3

```
## get search results for all judgements:
page <- 1
allTOC <- list()
while(TRUE) {
    thisPage <- httr::GET(paste0("https://www.bundesverfassungsgericht.de/SiteGlobals/Forms/",
                                 "Suche/Entscheidungensuche_Formular.html?gtp=5403124_list%3D",page))
    if(thisPage |> httr::content(as="parsed") |> xml_find_all("//ol[@id='searchResult']") |>
                                                    xml_children() |> length() < 1)
        break; ## stop if there are no more search results
```

```
    cat(page,",") ## some feedback to watch
    allTOC[[page]] <- thisPage
    page <- page + 1
    Sys.sleep(10)
}
save(file="data/TOCresponse.Rdata",allTOC)
```

```
## parse these search results:
lapply(allTOC,function(p) {
    thisPage <- httr::content(p,as="parsed") ## translate search result into xml tree
    links <- xml_find_all(thisPage,
                          "//ol[@id='searchResult']/li/a[@href[contains(.,'Entscheidungen')]]")
    lapply(links,function(n) {
        data.frame(
            az = xml_text(xml_find_first(n,"./span[@class='aktenzeichen']")),
            date = xml_text(xml_find_first(n,"./div")),
            href = xml_attr(n,"href"),
            summary = xml_text(xml_find_first(xml_parent(n),"./div[@class='kurztext']/p"))
        )}) |>
        bind_rows()
}) |> bind_rows() -> allTOC.df
```

```
str(allTOC.df)
```

```
'data.frame': 9071 obs. of  4 variables:
 $ az     : chr  "2 BvQ 52/23" "2 BvQ 51/23" "1 BvR 601/23" "2 BvR 406/23" ...
 $ date   : chr  "Beschluss vom 3. Mai 2023" "Beschluss vom 29. April 2023" "Beschluss vom 24.
 $ href   : chr  "SharedDocs/Entscheidungen/DE/2023/05/qk20230503_2bvq005223.html" "SharedDocs/
 $ summary: chr  "Eilantrag betreffend Teilungsversteigerungsverfahren nach Maßgabe einer Folge
```

```
allTOC.df |>
    group_by(az,date) |>
    summarise(n=n()) |>
    ungroup() |>
    with(table(n))
```

```
n
   1    2    4
8983   42    1
```

```
##
wPages <- list()
for (i in 1:nrow(allTOC.df)) {
    url <- paste0(HOST,allPages.df[i,"href"])
    wPages[[i]] <- GET(url)
    cat(i,",")
    Sys.sleep(10)
}
save(file="data/wPages.Rdata",wPages,allTOC.df)
```

- Judgement → date

- Judgement → href
- Judgement → body (chamber, senate,...)
- Judgement → topic (econ, justice, political, person)
- Judgement → judges (2 ways)

```
...
<div class="absatz">
   <div class="rechts">...</div>
   <div class="links">
     <p class="rr1">
       <span>
          hat die 2. Kammer des Zweiten Senats des Bundesverfassungsgerichts durch
       </span>
     </p>
   </div>
</div>
...
```

We will look for a `<div class="rr1">` to identify the body.

```
str2num <- function(x) {
    x |> stringr::str_replace_all(c(`(?i)first|erst|premier`="1",
                                    `(?i)second|zweite`="2",
                                    `(?i)third|dritte`="3",
                                    `(?i)fourth|vierte`="4")) |>
        gsub("[^0-9]","",x=_) |>
        as.numeric()
}
##
str2num("Le premier Senat")

[1] 1

str2num("The Second Chamber")

[1] 2

str2num("Die dritte Kammer")

[1] 3
```

```
ptree2body <- function (ptree) {
    ptree |> xml_find_all("//*[@class='rr1']/span") |> xml_text() -> ctable
    if(length(ctable)==0)
        ptree |> xml_find_all("//*[@class='rr1']") |> xml_text() -> ctable
    ctable |> stringr::str_count("Kammer|Chamber|Senat") -> cmatchnum
```

```
    which(max(cmatchnum)==cmatchnum)[1] -> cmatch
    ifelse(is.na(cmatch),NA,ctable[cmatch]) |>
    stringr::str_replace_all("[\n  ]+"," ") |>
    stringr::str_replace(paste0("^.*? (([1-9]\\.|First|Second|Third) *(Kammer|Chamber).*",
                                "(Senat[se]?|Panel)|[a-zA-Z]+ [sS][eé]nat[se]?|Beschwerdekammer
                                "Complaints Chamber|Plenum|Plenary).*"),"\\1") -> bvbody
    bvbody |> stringr::str_replace("^((.*) (Kammer|Chamber))?.*$","\\2") |> str2num() -> chambe
    bvbody |> stringr::str_replace("^.*?(([^ ]+) (Senat|Panel|[sS]énat)).*$","\\2") |>
        str2num() -> senat
    "" -> bvbody2
    if(grepl("Plenum|Plenary",bvbody)) "Plenum" -> bvbody2
    if(grepl("Beschwerdekammer|Complaints Chamber",bvbody)) "Beschwerdekammer" -> bvbody2
    if(!is.na(senat)) paste0(senat,".S") -> bvbody2
    if(!is.na(chamber)) paste0(senat,"/",chamber) -> bvbody2
    bvbody2
}
```

```
wpage=wPages[[4103]]
ptree <- content(wpage,as="parsed")
ptree2body(ptree)

[1] "2.S"
```

```
...
<div class="absatz">
   <div class="rechts">...</div>
   <div class="links">
     <p class="rr2">
       <span>
          den Richter ...
       </span>
     </p>
   </div>
   <div class="rechts">...</div>
   <div class="links">
     <p class="rr2">
       <span>
          und die Richterinnen ...
       </span>
     </p>
   </div>
</div>
```

We will search for `<div class="rr2">`
In XPath notation: `"//p[@class='rr2']//span"`
Since sometimes the span is missing, we also look for `"//p[@class='rr2']"`

- Problem: Names of judges are decorated with other things (»den Richter...«)

- Solution: We will try to remove these things as good as we can for most cases.
- This will allow us to identify names of judges.
- We will use these names to process the remaining cases.

```
ptree2judges1 <- function(ptree) {
    jtable1 <- xml_text(xml_find_all(ptree,"//p[@class='rr2']//span"))
    if(length(jtable1)==0)
        lapply(xml_text(xml_find_all(ptree,"//p[@class='rr2']")),strsplit,"\n| und |,") |>
            unlist() -> jtable1
    ##
    paste0("^(nd | *| *und |and |durch |unter Mitwirkung )|([Dd]er |[Dd]ie |[Dd]en |des )?",
           "(Richter|Judge|Justice|(Vi[zc]e[ -]?)?[pP]r[äeé]sident(e)?)(in|en|innen)?s? ?|",
           "$| ?[()]|(und|and)$") |> gsub("",jtable1) |> as.list() |>
    lapply(X=_,strsplit,",") |> unlist() -> jtable2  ## multiple judges in one cell
    paste0(", *|^[  ]*| *$|.*hat die Kammer|.*Senat.*|.*Bundesverfassungsgericht|",
           ".*Mitwirkung|.*in Verbindung|.*Bekanntmachung.*|.*beschlossen.*") |>
    gsub("",jtable2) |> setdiff(c("und","and",""))
}
##
ptree2judges1(content(wPages[[4103]],as="parsed"))

[1] "Voßkuhle"    "Di Fabio"    "Mellinghoff" "Lübbe-Wolff" "Gerhardt"
[6] "Landau"      "Huber"       "Hermanns"
```

```
...
<table class="st" summary="Unterschriften der Richter" width="80%" align="center">
  <tbody ...>
    <tr>
      <td class="st" colspan="4" width="33%">
         <span>Müller</span>
      <td>
      <td class="st" colspan="4" width="33%">
         <span>Langenfeld</span>
      <td>
      <td class="st" colspan="4" width="33%">
         <span>Fetzer</span>
      <td>
    </tr>
   </tbody>
<table>
```

Why collect the same information twice?

→ No version is 100% reliable, so we better compare.

```
ptree2judges2 <- function(ptree) {
    jtable2 <- xml_find_all(ptree,"//table[@summary='Unterschriften der Richter']//td//span")
    if(length(jtable2)==0)
        jtable2 <- xml_find_all(ptree,"//table[@summary='Unterschriften der Richter']//td")
    ##
    jtable2 |> xml_text() |> lapply(strsplit," und |:|,") |> unlist() |>
        gsub("[, :]*$","",x=_) |>
        gsub("[_ \n]"," ",x=_) |>
        gsub("^[ ]*","",x=_) |>
        gsub(paste0("Dr. |Die |Der |(Richter|Vizepräsident)(in)? |(ist|deshalb).*gehindert|",
                    "Amt ausgeschieden|deshalb an der Unterschrift|gehindert."),"",x=_) |>
        gsub("- *","-",x=_) |>
        setdiff(c("","  ","Judges","unter","Mitwirkung"))
}
##
ptree2judges2(content(wPages[[4103]],as="parsed"))
```

```
[1] "Voßkuhle"    "Di Fabio"    "Mellinghoff" "Lübbe-Wolff" "Gerhardt"
[6] "Landau"      "Huber"       "Hermanns"
```

In the text of the judgement:

> *...Art. 2 Abs. 1 GG gewährleistet in Verbindung mit Art. 1 Abs. 1 GG das allgemeine Persönlichkeitsrecht. Dieses Recht...*

```
read.csv(text="art,topic
2.1|3.1|9.3|12.1|14.[12],econ
2.2|10.|11.|13.|16.2|19.4|101.|103.[123],justice
5.[123]|8.|9.[12]|28.2|33.[2345]|38.,political
4.[12]|6.|7.|16a.|140.,person") |>
    mutate(art = paste0("^(",gsub("\\.","\\\\.",art),").*$"),
           pat=setNames(topic,art)) |> pull(pat) -> art2topic
```

Now we have a vector with unusual names:

```
art2topic
```

```
                     ^(2\\.1|3\\.1|9\\.3|12\\.1|14\\.[12]).*$
                                                        "econ"
^(2\\.2|10\\.|11\\.|13\\.|16\\.2|19\\.4|101\\.|103\\.[123]).*$
                                                     "justice"
        ^(5\\.[123]|8\\.|9\\.[12]|28\\.2|33\\.[2345]|38\\.).*$
                                                   "political"
                       ^(4\\.[12]|6\\.|7\\.|16a\\.|140\\.).*$
                                                      "person"
```

```
ptree2topic <- function(ptree) {
    ptree  |>
    xml_find_all("//*[contains(text(),' GG')]")   |>
        lapply(function(n) stringr::str_extract_all(xml_text(n),
```

```
                "Art\\.[  ]*[0-9]+[  ,]*(Abs.[  ][0-9]+)?.{0,10}GG")) |>
    unlist() |>
    stringr::str_replace_all("^Art\\.[  ]*([0-9]+)[  ,]*(Abs.[  ]([0-9]+))?.*$","\\1.\\3") |>
    stringr::str_replace_all(art2topic) -> dirtyMatches
    dirtyMatches[dirtyMatches %in% art2topic] |>
        table() |> sort(decreasing=TRUE) |> names() |> head(1) -> topic
    if(is.null(topic)) topic<-NA
    topic
}
##
sapply(4104:4110,function(p) ptree2topic(content(wPages[[p]],as="parsed")))
```

```
[1] "justice" "econ"    NA        NA        "econ"    "justice" NA
```

(`mclapply` is similar to `lapply`, except that it is parallel, hence, faster.)

```
mclapply(wPages,function(wpage) {
    href <- sub(HOST,"",wpage[["url"]])
    ptree <- content(wpage,as="parsed")
    bvbody2 <- ptree2body(ptree)
    topic <- ptree2topic(ptree)
    judges1 <- ptree2judges1(ptree)
    judges2 <- ptree2judges2(ptree)
    data.frame(href=href,bvbody=bvbody2,topic=topic,j1=I(list(judges1)),j2=I(list(judges2)))
}) |> bind_rows() -> judgesRaw
```

Judges might need some cleaning…

```
## find names of judges
judgesRaw |> select(j1,j2) |> unlist() |>  table() |>
   sort(decreasing=TRUE) -> jTable
##
head(jTable) |> t() |> xtable()
```

	Papier	Kirchhof	Kessal-Wulf	Huber	Mellinghoff	Hassemer
1	2910	2679	2532	2411	2278	2243

```
tail(jTable,5) |> t() |> xtable()
```

	vom 11. August 1993BGBl I S. 1473 am 13. Mai 2009	W	Wintrich	Zeidler	Zeidler als Vorsitzender
1	1	1	1	1	1

So far:

- We have identified some judges (`realJudges`).

```
(names(jTable)[jTable>10] -> realJudges) |> paste(collapse=", ")
```

[1] "Papier, Kirchhof, Kessal-Wulf, Huber, Mellinghoff, Hassemer, Broß, Osterloh, König, Gaier, Steiner, Paulus, Müller, Di Fabio, Hermanns, Maidowski, Hohmann-Dennhardt, Landau, Hömig, Voßkuhle, Lübbe-Wolff, Masing, Baer, Schluckebier, Britz, Bryde, Eichberger, Jaeger, Gerhardt, Hoffmann-Riem, Limbach, Sommer, Langenfeld, Haas, Jentsch, Christ, Kühling, Ott, Grimm, Radtke, Harbarth, Wallrabenstein, Härtel, Winter, Seidl, Graßhof, Kruis, Seibert, Henschel, Söllner, Herzog, Wolff, Dieterich, Böckenförde, Klein, Offenloch, Fetzer, s durch, Mahrenholz, mit § 93a BVerfGG in der"

(We still have do do some cleaning)

- There are some "unknowns" which contain the correct string ("Voßkuhle als Vorsitzender")

  We identify them in `knownJ`.

- There are some spelling mistakes ("Vosskuhle" instead of "Voßkuhle")

  We get some help from `stringdist::afind` to match these, but we will do this "manually".

```
realJudges[!grepl("\\.|- Erster|BVerfGG|s durch",realJudges)]  |>
    sub("^Wolff","H.A.Wolff",x=_) -> realJudges
unrealJ <- setdiff(names(jTable),realJudges)
sapply(realJudges,function(j) grep(paste0("^(.*[^A-Z])?",j,"([^a-z].*)?$"),unrealJ)) |>
    unlist() |>  unique() |> sort() -> knownJ
unrealJ[-knownJ] -> unknown ## ← these contain missspelled names
str(unknown)

 chr [1:129] "Wolff" "s durch" " mit § 93a BVerfGG in der" "Benda" ...

stringdist::afind(unknown,realJudges,value=FALSE,method="jw") -> jDist
str(jDist)

List of 2
 $ location: int [1:129, 1:58] 1 1 10 1 1 1 4 4 1 1 ...
 $ distance: num [1:129, 1:58] 1 0.556 0.333 1 0.333 ...
```

```
lapply(1:length(unknown),
       function(i) {
           best=min(jDist$distance[i,]);
           ji=which(best==jDist$distance[i,])[1];
           data.frame(best=best,ji=ji,judge=realJudges[ji],
                      match=substr(unknown[i],jDist$location,40))}) |>
    bind_rows() |> filter(best < .2) |>
    summarise(pat=paste0(paste0(match,collapse="|"),",",judge[1]),.by="judge") |
    xtable()
```

	judge	pat
1	Baer	mit § 93a BVerfGG in der|Haager| mit § 93 a BVerfGG in|Berger|BVerfGG|daher
2	Voßkuhle	Vosskuhle,Voßkuhle
3	Hassemer	Hesse|Hasseme|Hasssemer,Hassemer
4	Kessal-Wulf	Kessal-Wulff,Kessal-Wulf
5	Hömig	Hmig,Hömig
6	König	Hörnig,König
7	Gaier	Niedermaier,Gaier
8	Steiner	Dr. Stein|Stein|Steinberger|Steine ist ausgeschieden,Steiner
9	Kühling	Khling|Khling ist aus dem Amt geschieden,Kühling
10	Broß	Bro|Brox,Broß
11	Seibert	Dr. Zweigert|Zweigert,Seibert
12	Gerhardt	Gerhard,Gerhardt
13	Henschel	Hentschel,Henschel
14	Hoffmann-Riem	Hofmann-Riem,Hoffmann-Riem
15	Lübbe-Wolff	Lübbe-Woff|Lübbe -Wolff|Lübbe- Wolff|Lübbe-Wollf,Lübbe-Wolff
16	Mellinghoff	Meßling,Mellinghoff
17	Wallrabenstein	Wallrabensein,Wallrabenstein
18	Britz	als Vorsitzenden|Seuffert als Vorsitzender|Zeidler als Vorsitzender,Britz
19	Graßhof	and Grasshof,Graßhof
20	Haas	Dr. Klaas|Klaas,Haas
21	Winter	Dr. Wintrich|Wintrich,Winter
22	Hermanns	Herrmanns,Hermanns
23	Hohmann-Dennhardt	Hohmann -Dennhardt|Hohmann-Dennhard,Hohmann-Dennhardt
24	Seidl	Zeidler,Seidl

```
## ↑ this gives us a template for the following table ↓
```

```
## correct spelling of some judges:
read.csv(text="pattern,replace
Bro$|Brox,Broß
Gerhard$,Gerhardt
Grasshof,Graßhof
Hasseme$|Hasssemer,Hassemer
Hentschel,Henschel
Herrmanns,Hermanns
Hofmann-Riem,Hoffmann-Riem
Hohmann -Dennhardt|Hohmann-Dennhard$,Hohmann-Dennhardt
Hmig,Hömig
Kessal-Wulff,Kessal-Wulf
Khling,Kühling
Lübbe-Woff|Lübbe -Wolff|Lübbe- Wolff|Lübbe-Wollf,Lübbe-Wolff
Stein$|Steine ,Steiner
Vosskuhle,Voßkuhle
Wallrabensein,Wallrabenstein
^Wolff,H.A.Wolff") |>
    with(data=_,setNames(replace,pattern)) ->
```

```
    judgeTrans
```

```
str(judgeTrans)
```

```
 Named chr [1:16] "Broß" "Gerhardt" "Graßhof" "Hassemer" "Henschel" ...
 - attr(*, "names")= chr [1:16] "Bro$|Brox" "Gerhard$" "Grasshof" "Hasseme$|Hasssemer" ...
```

```
## spell-correct realJudges:
stringr::str_replace_all(realJudges,judgeTrans) -> realJudges2
## clean all judges:
cleanJudges <- function(j) {
    stringr::str_replace_all(unlist(j),judgeTrans) |>
    sapply(function(text) realJudges2[sapply(realJudges2,grepl,text )]) |>
        unlist() |> unique() |> sort()
}
##
cleanJudges(c("Der Vorsitzende Bro","Lübbe-Woff und Vosskuhle"))
```

```
[1] "Broß"        "Lübbe-Wolff" "Voßkuhle"
```

```
judgesRaw |> rowwise() |>
    mutate(across(c(j1,j2),function(j) I(list(cleanJudges(j))))) |>
    ungroup() -> jjClean
```

```
jjClean |>
    rowwise() |>
    mutate(j=list(unique(unlist(j1,j2)))) |>
    tidyr::unnest(cols=c(j)) |> with(table(j))
```

```
j
             Baer       Böckenförde             Britz              Broß
             806                36              770               988
           Bryde            Christ          Di Fabio         Dieterich
             740               462              874                37
      Eichberger            Fetzer             Gaier          Gerhardt
             734                19              922               694
         Graßhof             Grimm        H.A.Wolff              Haas
             123               396               37               525
        Harbarth            Härtel          Hassemer          Henschel
             333               220             1125                75
        Hermanns           Herzog     Hoffmann-Riem Hohmann-Dennhardt
             873                46              677               862
           Hömig            Huber            Jaeger           Jentsch
             860              1206              722               517
     Kessal-Wulf          Kirchhof             Klein             König
            1269              1340               34               931
           Kruis           Kühling            Landau        Langenfeld
             120               429              859               526
         Limbach       Lübbe-Wolff        Mahrenholz         Maidowski
             623               823                8               868
 [ reached getOption("max.print") -- omitted 18 entries ]
```

```
jjClean |> rowwise() |> mutate(n1 = length(j1), n2=length(j2)) -> jjNN
with(jjNN,table(n1,n2)) |> xtable()
```

	0	1	2	3	4	5	6	7	8	15	16
0	162	0	0	10	0	0	0	0	18	0	0
1	0	7	0	0	0	0	0	0	0	0	0
2	0	0	4	1	0	0	0	0	0	0	0
3	19	0	2	7180	0	0	0	0	0	0	0
4	0	0	0	0	8	0	0	0	0	0	0
5	0	0	0	0	0	3	0	0	0	0	0
6	0	0	0	0	0	0	42	1	0	0	0
7	8	0	0	0	0	0	0	285	1	0	0
8	4	1	0	0	0	0	0	0	1301	0	0
15	0	0	0	0	0	0	0	0	0	2	0
16	0	0	0	0	0	0	0	0	0	0	1

```
(probs <- with(jjNN,which(n1>0 & n2>0 & n1 != n2)))

[1] 4771 5810 7668 8754 8755 8823
```

```
## look at cases where number of judges j1 and j2 differ:
for (i in 1:length(probs)) {
    cat("\n--------------------",i,"-",probs[i],"--------------------\n")
    cat("1 clean:",paste(sort(unlist(jjClean[probs[i],"j1"])),collapse=","))
    cat("\n")
    cat("1 raw:  ",paste(sort(unlist(judgesRaw[probs[i],"j1"])),collapse=","))
    cat("--\n")
    cat("2 clean:",paste(sort(unlist(jjClean[probs[i],"j2"])),collapse=","))
    cat("\n")
    cat("2 raw:  ",paste(sort(unlist(judgesRaw[probs[i],"j2"])),collapse=","))
    cat("\n")
}
```

```
-------------------- 1 - 4771 --------------------
1 clean: Gaier,Kirchhof
1 raw:   -Dennhardt,Hohmann Gaier,Kirchhof--
2 clean: Gaier,Hohmann-Dennhardt,Kirchhof
2 raw:   Gaier,Hohmann-Dennhardt,Kirchhof

-------------------- 2 - 5810 --------------------
1 clean: Bryde,Gaier,Haas,Hoffmann-Riem,Hohmann-Dennhardt,Hömig,Steiner
1 raw:    - Erster,Bryde,Gaier,Haas,Hoffmann-Riem,Hohmann-Dennhardt,Hömig,Steiner--
2 clean: Bryde,Gaier,Haas,Hoffmann-Riem,Hohmann-Dennhardt,Hömig,Papier,Steiner
2 raw:   Bryde,Gaier,Haas,Hoffmann-Riem,Hohmann-Dennhardt,Hömig,Papier,Steiner

-------------------- 3 - 7668 --------------------
1 clean: Hohmann-Dennhardt,Hömig,Jaeger,Kühling,Papier,Steiner
1 raw:   Hohmann-Dennhardt,Hömig,Jaeger,Kühling,Papier,Steiner--
```

```
2 clean: Haas,Hohmann-Dennhardt,Hömig,Jaeger,Kühling,Papier,Steiner
2 raw:   Haas,Hohmann-Dennhard,Hömig,Jaeger,Kühling,Papier,Steiner

-------------------- 4 - 8754 --------------------
1 clean: Grimm,Hömig,Seidl
1 raw:   Grimm,Hömig,Seidl--
2 clean: Grimm,Hömig
2 raw:   Grimm,Hömig

-------------------- 5 - 8755 --------------------
1 clean: Grimm,Hömig,Seidl
1 raw:   Grimm,Hömig,Seidl--
2 clean: Grimm,Hömig
2 raw:   Grimm,Hömig

-------------------- 6 - 8823 --------------------
1 clean: Graßhof,Hassemer,Jentsch,Kirchhof,Kruis,Limbach,Sommer,Winter
1 raw:   Graßhof,Hassemer,Jentsch,Kirchhof,Kruis,Limbach,Sommer,Winter--
2 clean: Kruis
2 raw:   Kruis
```

Find patterns in the summaries of cases:

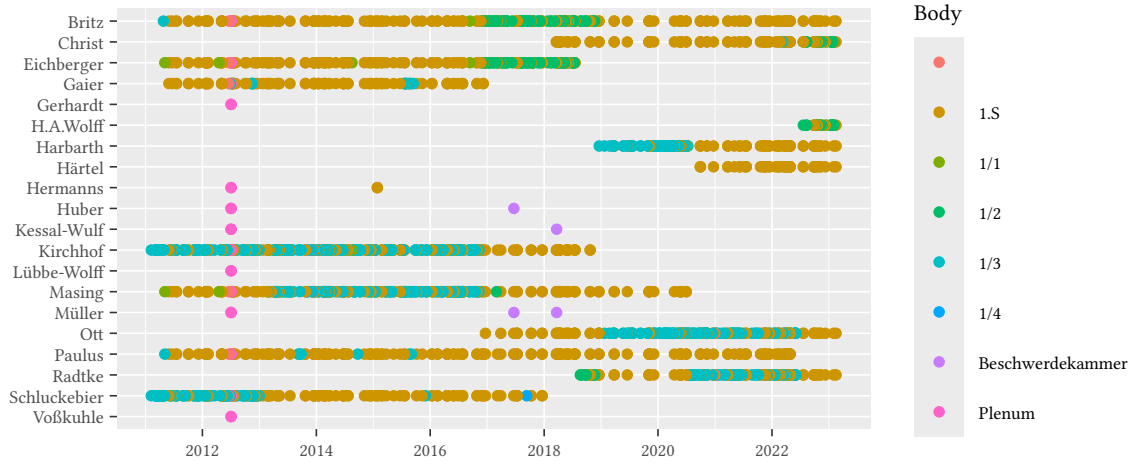
```
PAT <- list()
PAT[["reject"]] <- paste0("unsuccessful|abgelehnt|ablehnung|unzulässig|erfolglos|unbegrün|",
                          "nichtannahme|verfristet|verworfen|zurückgewiesen")
PAT[["accept"]] <- "^successful|erfolgreich| annahme"
##
PAT[["reject2"]] <- paste0("bestätigung| darf |erledigt|einstellung| genüg|",
                           "missbrauchsgebühr|verwerfung|versagung|verfassungsgemäß|",
                            "verfassungsmaßig| vereinbar|zulässig")
PAT[["accept2"]] <- paste0("anspruch|auslagenerstattung|aussetzung|ausgeschlossen|",
           "begründ|angenommen|angeordnet|anordnung|bedenken|bedürfnis|beanstand|",
           "berichtig|berücksichtig|bewillig| muss | bedarf|erfolgreich|fristgerecht|",
           "fehlende| hätte |hinreichen|klarstell|kompetenzwidrig|prozesskosten|",
           "rechtsschutzinteresse| nichtig|verstößt|pflicht|substantiier| verletz|trennung|",
           "unvereinbar|untersag|verfassungswidrig|verstoß|verstößt|verletzung|vorrang")
PAT[["not"]] <- "nicht|kein|trotz|unzureichen|entfällt|wegfall"
```

```
## always take the longer list:
jjClean |> rowwise() |>
  mutate(judge = ifelse(length(j1)>length(j2),I(list(j1)),I(list(j2)))) |>
  ungroup() |>
  left_join(allTOC.df) |>
  mutate(date1 =  lubridate::dmy(date,locale = "de_DE.UTF-8",tz="CET")) |>
  filter(!is.na(date1) & date1 > as.POSIXct("1951-01-01")) |>
  mutate(reject = grepl(PAT[["reject"]],summary,ignore.case=TRUE),
         accept = !reject & grepl(PAT[["accept"]],summary,ignore.case=TRUE),
         not = grepl(PAT[["not"]],summary,ignore.case=TRUE),
         reject2 = xor(not,grepl(PAT[["reject2"]],summary,ignore.case=TRUE)),
         accept2 = xor(not,grepl(PAT[["accept2"]],summary,ignore.case=TRUE)),
         sumx = 10*(accept-reject) + (accept2-reject2),
```

```
         sumdec = ifelse(sumx==0,NA,sumx>0)
         ) -> jjSucc
jjSucc |> with(table(sumx))

sumx
 -11  -10   -9   -1    0    1    9   10   11
 744 1626  674 1444 2505 1399   64  123  444
```

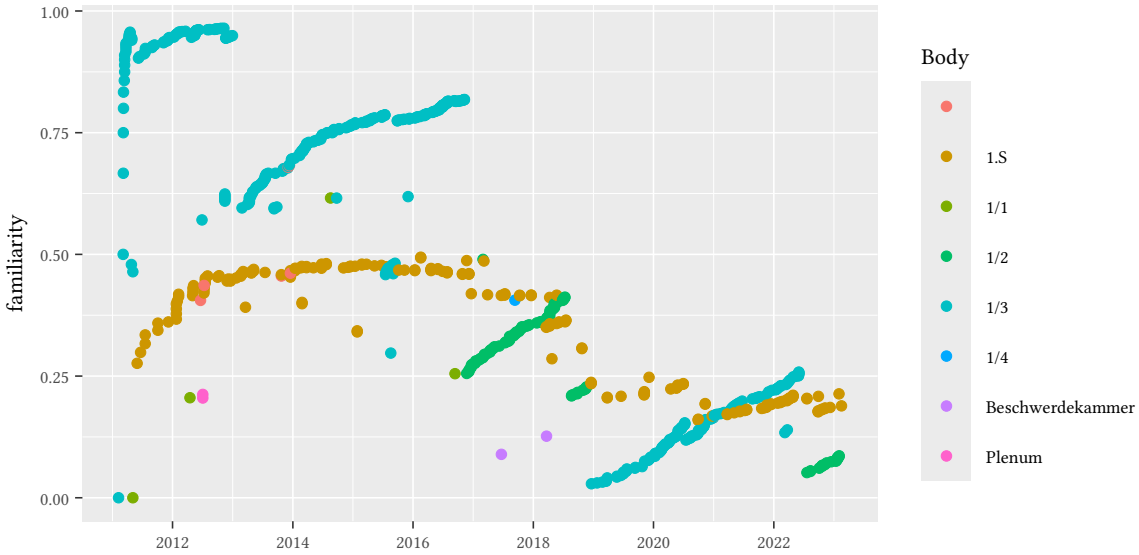
```
jjSucc |> filter(grepl("Baer",judge)) |>
    rowwise() |>
    reframe(data.frame(date=date1,body=bvbody,j=setdiff(judge,"Baer"))) |>
    ggplot(aes(x=date,y=factor(j,levels=rev(sort(unique(j)))),color=factor(body))) +
    geom_point() + labs(x=NULL,y=NULL,color="Body")
```



Define “familiarity” as in Engel (2022), “Lucky you: Your case is heard by a seasoned panel—Panel effects in the German Constitutional Court”.

```
jjSucc |>
    rowwise() |>
    reframe(href,bvbody,date1,other=I(list(judge)),judge=data.frame(judge)) |>
    group_by(judge) |>
    arrange(date1) |>
    mutate(ownExp=1:n()) |>
    rowwise() |>
    reframe(href,bvbody,date1,judge,ownExp,data.frame(other)) |>
    filter(other != judge) |>
    group_by(judge,other) |>
    mutate(otherExp=0:(n()-1)/ownExp) |>
    group_by(href,bvbody,date1,judge) |>
        summarise(familiarity=mean(otherExp),.groups="drop") -> bvFamiliarity
```

```
bvFamiliarity |> filter(judge=="Baer" & date1>as.POSIXct("1950-01-01")) |>
    ggplot(aes(x=date1,y=familiarity,color=bvbody)) + geom_point() + labs(x=NULL,color="Body")
```




```
"https://de.wikipedia.org/wiki/Liste_der_Richter_des_Bundesverfassungsgerichts" |>
   httr::GET() |>
   httr::content() |>
   rvest::html_table() |> first() -> jWiki

stringdist::afind(x=unlist(jWiki[,1]),
                  pattern=sub("H.A.Wolff","Heinrich Amadeus Wolff",realJudges2),
                  value=FALSE,
                  method="jw") -> jwDist

lapply(seq(length(realJudges2)),function(j) {
    distances <- jwDist$distance[,j]
    m <- min(distances)
    pos=which(m==distances)[1]
    data.frame(judge=realJudges2[j],dist=m,jWiki[pos,])}) |>
    bind_rows() -> judgeDF
```

```
judgeDF |> arrange(-dist) |> select(judge,Name..Lebensdaten.,Vorschlag,dist)  |>
    head(12) |> xtable()
```

	judge	Name..Lebensdaten.	Vorschlag	dist
1	Papier	Hans-Jürgen Papier (* 1943)	CDU/CSU	0.00
2	Kirchhof	Ferdinand Kirchhof (* 1950)	CDU/CSU	0.00
3	Kessal-Wulf	Sibylle Kessal-Wulf (* 1958)	CDU/CSU	0.00
4	Huber	Peter M. Huber (* 1959)	CDU/CSU	0.00
5	Mellinghoff	Rudolf Mellinghoff (* 1954)	CDU/CSU	0.00
6	Hassemer	Winfried Hassemer (1940–2014)	SPD	0.00
7	Broß	Siegfried Broß (* 1946)	CDU/CSU	0.00
8	Osterloh	Lerke Osterloh (* 1944)	SPD	0.00
9	König	Doris König (* 1957)	SPD	0.00
10	Gaier	Reinhard Gaier (* 1954)	SPD	0.00
11	Steiner	Udo Steiner (* 1939)	CDU/CSU	0.00
12	Paulus	Andreas Paulus (* 1968)	FDP	0.00

```
judgeDF |> with(table(Vorschlag)) |>
    xtable()
```

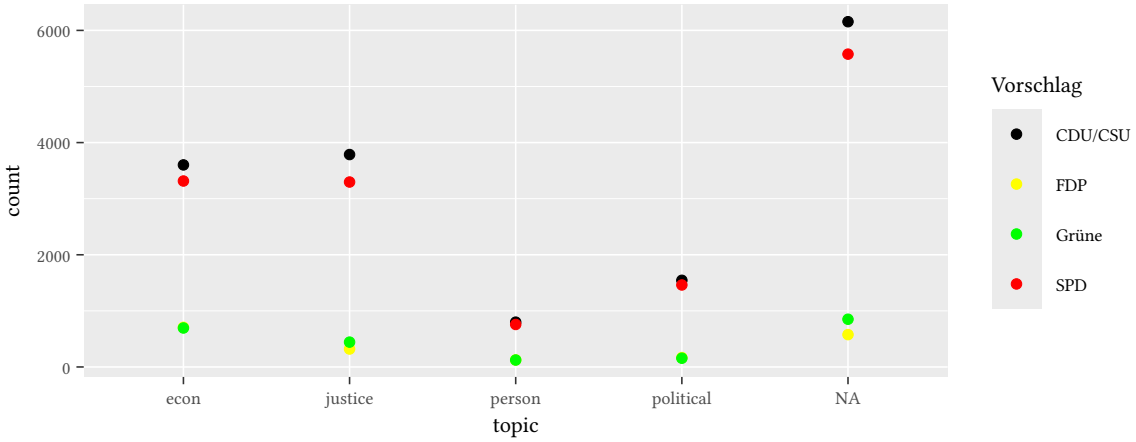
	Vorschlag
CDU/CSU	24
FDP	5
Grüne	4
SPD	25

```
judgesRaw |>
    with(table(topic,useNA="always")) |>
    xtable()
```

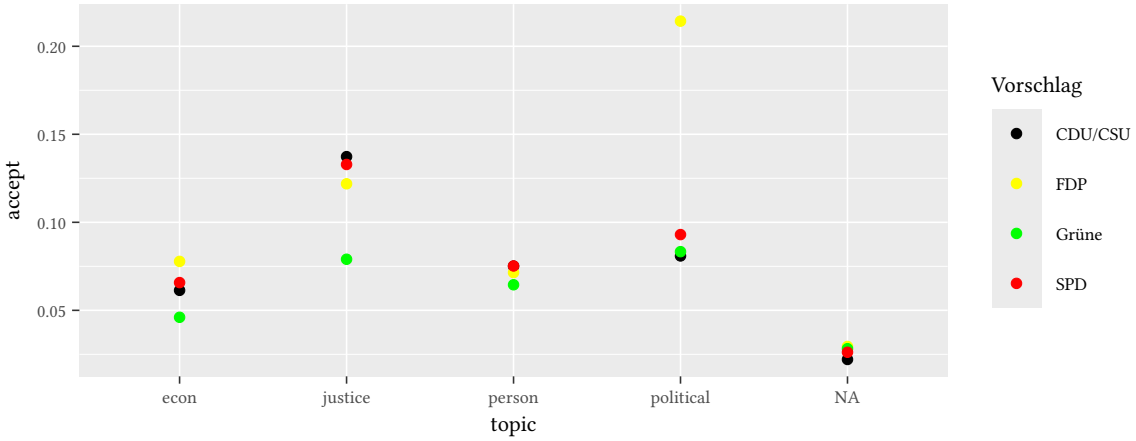
	topic
econ	2200
justice	2336
person	451
political	744
NA.	3329

```
jjSucc |>
    rowwise() |>
    reframe(href,accept,topic,data.frame(judge)) |>
    left_join( judgeDF |> select(judge,Vorschlag)) |>
    mutate(topic=factor(topic),Vorschlag=factor(Vorschlag)) -> judgeVS
```

```
partyCol <- scale_color_manual(values=c("CDU/CSU"="black","FDP"="yellow","Grüne"="green","SPD"="red"))
judgeVS |>
  group_by(Vorschlag,topic) |>
  summarise(count=n()) |>
  ggplot(aes(x=topic,y=count,color=Vorschlag)) + geom_point() + partyCol
```




```
judgeVS |>
    group_by(Vorschlag,topic) |>
    summarise(accept=mean(accept)) |>
    ggplot(aes(x=topic,y=accept,color=Vorschlag)) + geom_point() + partyCol
```




```
jjSucc |>
    filter(sumx != 0) |>
    mutate(accept = sumx > 1) |>
    select(href,accept) |>
    left_join(bvFamiliarity) -> jjAcceptFam

jjAcceptFam |>
    filter(!is.na(familiarity)) |>
    group_by(href,bvbody,date1) |>
    summarise(familiarity=mean(familiarity),
              accept=mean(accept),
              nJudge=n(),
              .groups="drop") -> jjAcceptFamShort
```

```
summary(glm(accept ~ familiarity + nJudge + date1 ,family=binomial,data=jjAcceptFamShort))
```

```

Call:
glm(formula = accept ~ familiarity + nJudge + date1, family = binomial,
    data = jjAcceptFamShort)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.6911  -0.4741  -0.4100  -0.3565   2.6012

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -5.846e+00  3.827e-01 -15.277  < 2e-16 ***
familiarity  2.842e+00  3.642e-01   7.803 6.04e-15 ***
nJudge       6.383e-02  2.422e-02   2.635  0.00842 **
date1        1.399e-09  1.782e-10   7.851 4.14e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

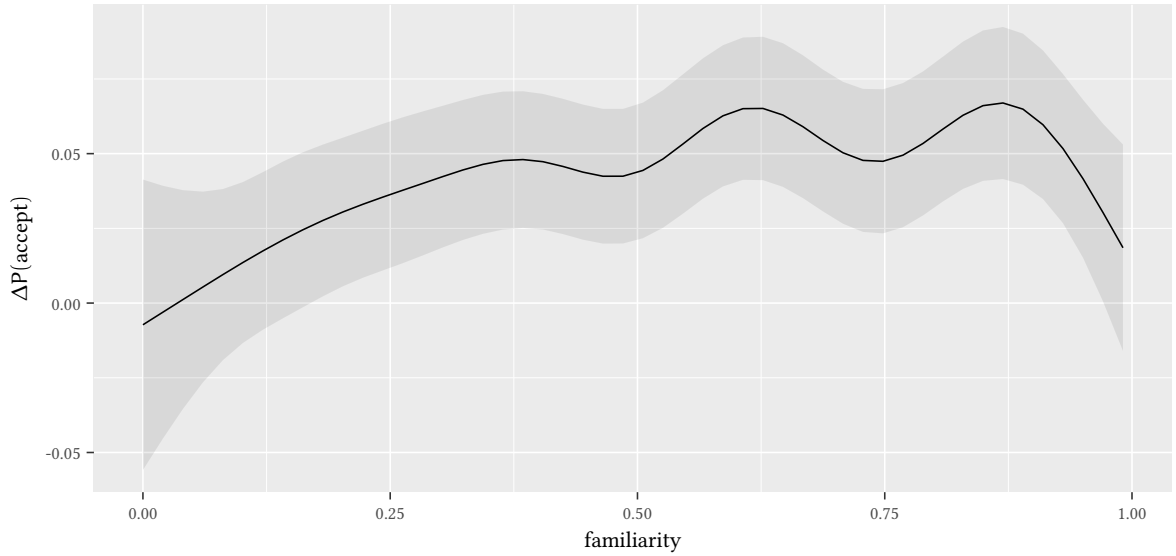
    Null deviance: 4030.9  on 6461  degrees of freedom
Residual deviance: 3934.4  on 6458  degrees of freedom
AIC: 3942.4

Number of Fisher Scoring iterations: 5
```

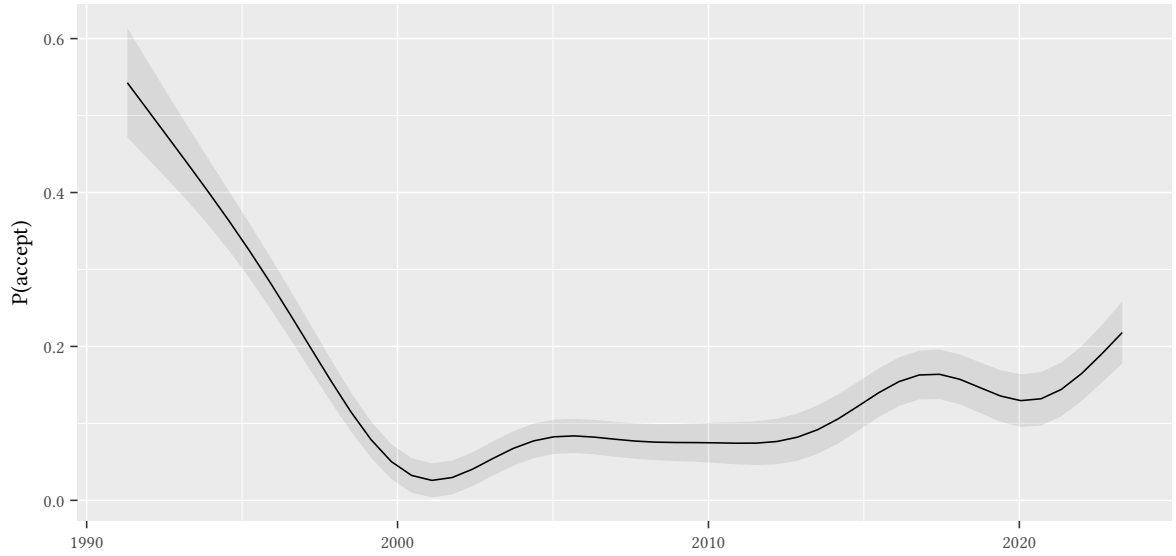
```
library(mgcv)
library(tidymv)

jjAcceptFam |> mutate(date=as.numeric(date1)) |>
    gam(accept ~ s(familiarity) + s(date) + factor(judge),data=_) -> est
```

```
predict_gam(est,values=list(judge="Papier",date=as.numeric(as.POSIXct("2000-01-01")))) |>
    ggplot(aes(familiarity,fit)) + geom_smooth_ci() + labs(y="$\\Delta P($accept$)$")
```

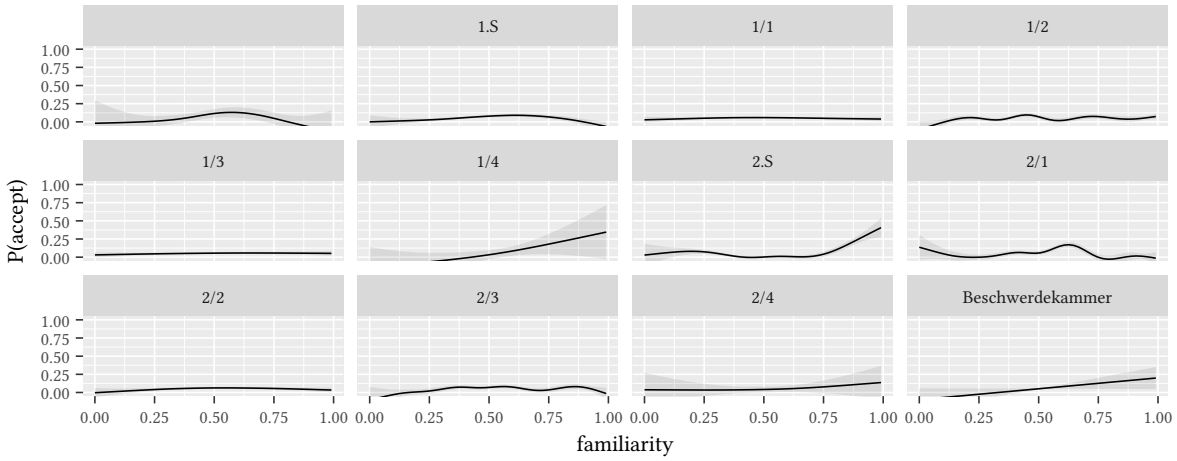



```
predict_gam(est,values=list(judge="Papier",familiarity=.5)) |>
    ggplot(aes(as.POSIXct(date,origin="1970-01-01"),fit)) + geom_smooth_ci() + labs(x=NULL,y="P
```




```
jjAcceptFam |> mutate(date=as.numeric(date1),bvf=factor(bvbody)) |>
    gam(accept ~ s(familiarity,by=bvf) + s(date),data=_) -> estBody
```

```
predict_gam(estBody,values=list(date=as.numeric(as.POSIXct("2000-01-01")))) |>
ggplot(aes(x=familiarity,y=fit)) + geom_smooth_ci() + facet_wrap(~bvf) +
    coord_cartesian(ylim=c(0,1)) + labs(y="P(accept)")
```



9 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).
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)

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