Advanced R workshop
Day 1

Assoc Prof Dr Christina Ramsenthaler MSc PhD
ramn@zhaw.ch, ramsenthalerchristina@gmail.com (Dr R's Stats Corner)
Research methods and Statistics
Winterthur (ZHAW), King's College London & Hull York Medical School

Advanced R workshop
Freiburg, 2022-10-18

Roadmap

Day 1	R projects, Workflow, Data management
Session 1: 09.00-10.50	
09.00-09.30	Introduction: Course info, rules, personal questions
09.35-10.05	R function quiz
10.10-10.40	Data structures in R: Tidy data
10.40-10.50	Longer break, buffer 1
Session 2: 10.50-12.30	
10.50-11.20	Input R project setup
11.25-11.55	Workshop: Workflow in action
12.00-12.30	Working on own dataset
12.30-13.30	1h lunch break
Session 3: 13.30-15.00	
13.30-14.00	Open Science, Data cleaning scripts and functions 1
14.05-14.35	Workshop: Data cleaning functions 1
14.40-15.10	Working on own dataset
15.10-15.20	Longer break, buffer 2
Session 4: 15.20-17.00	
15.20-15.50	Data cleaning functions 2
15.55-16.25	Workshop: Data cleaning functions 2
16.30-17.00	Working on own dataset

A possible inspiration

Tom Sachs‘s Workshop rules:

I. Sacred Space
II. Always be knolling
III. Keep a list
IV. Work to code (creativity is the enemy)
V. I understand
VI. Leatherface
VII. Thoroughness counts
VIII. Be on time
IX. Reset
X. Persistence

Pictures from:
https://www.tomsachs.com/

Code of conduct / Workshop rules

Our workshop needs to be a safe place for everybody. We will respect and follow the rules. The rules are:

- We are committed to making participation a harassment-free experience for everyone, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, ethnicity, age, or religion.
- Examples of unacceptable behavior by participants include the use of sexual language or imagery, derogatory comments or personal attacks, trolling, public or private harassment, insults, or other unprofessional conduct.
- English is the second language for most of us. We help each other and are kind when it comes to mistakes, pronounciation or word choice etc.
- We are not interested in each other's academic credentials. However, we are deeply interested in each other's work.
- We are all learners. We are still learning. Everybody can contribute a lot. Nobody has all the answers.
- We honour the work by focusing on it. This means staying off mobile phones and other distractive stimuli (as far as possible, if you have work/family commitments that need monitoring, please do).
- If in doubt, choose kindness. Be kind to yourself and to others. Respect your own well-being. If you need a break for whichever reason, please quietly leave the room. We understand.

When does this course work for me?

- This course is right for you if:
  - You want to learn how to better manage your R projects.
  - You have a basic understanding of how base R and the tidyverse work.
  - You either do all your analysis work in R or are using it in conjunction with other programmes.

- What is not needed:
  - Being fluent in R.
  - Solely working with R.
  - Planning to transition to R and using it as your sole programme.

- This course may not be right for you if:
  - You are fluent in R.
  - You expect a lot of in-depth information about certain techniques. This course is an overview of advanced techniques (like a smorgasbord). I aim for breadth, not depth.
  - You expect the „be-all-and-end-all“ solution to all your data wrangling problems.

Like on a smorgasbord, you may pick & choose and only attend certain sessions. I don‘t mind.

Managing expectations and learning goals

- **My absolute 'Must-have's':**

- 1
- 2
- 3

- **My 'Nice-to-have's':**

- 1
- 2
- 3

Getting to know you a bit better

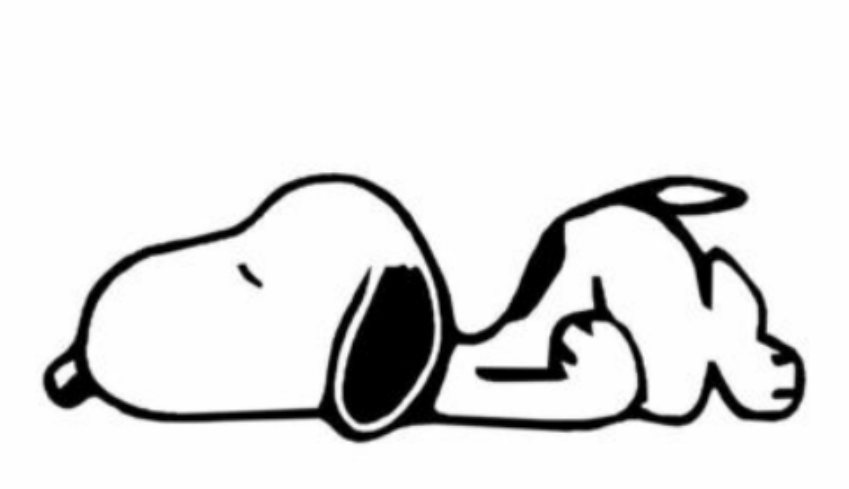
I would like to get to know you better. Here are 6 questions to do so. I'll start.

1) What is your most beloved R function (and why)?
2) What is your most hated R feature (and why)?
3) What recent R project are you most proud of? If you can, show us what you built (i.e. a nice graph, a dashboard, a table, a map). If you can't show us, please describe what you have done.
4) What music do you like to listen to while working with R or other coding/statistical programmes?
5) For you, have there been positive aspects of the pandemic? For instance, have you learned something new during the pandemic? Have you been surprised about yourself during that time?
6) What do you think your secret superpower is and would you be willing to share with us what it is?

Datasets – How does your data normally look like?

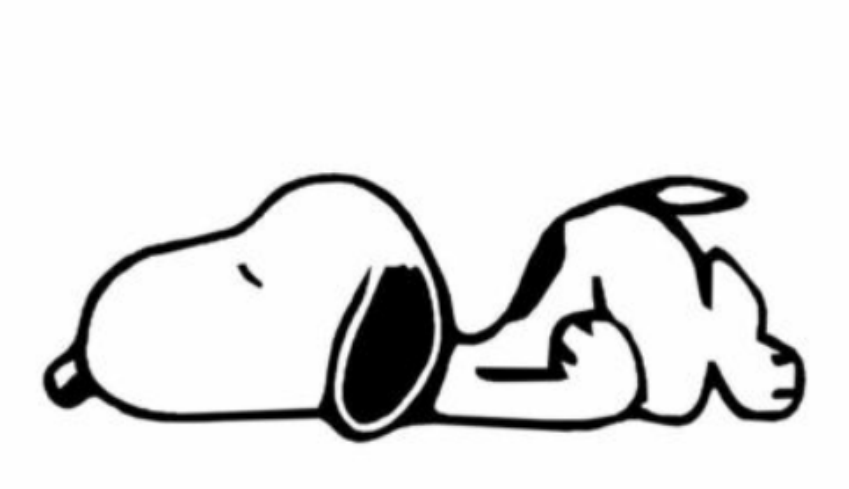
- Would you like to tell me how your data normally looks like?
- Please consider:
  - What (rectangular) format does your data normally take?
  - Which type of data structure do you normally follow?
  - What are observations in your dataset? How often are they repeated?
  - Do you interact with database systems, and – if so – which systems do you use?
  - Where do you normally get your data from? What is the most common input format you work with?
  - Where and how do you do your data cleaning?
  - Do you transfer your data back into the open domain? Do you need to upload it into a repository or are you prohibited from doing so (due to GDPR or industry requirements)?
  - Do you need to transfer datasets between team members or are datasets frequently transferred to you? Do you work in teams not sharing the same office? How does communication work?
  - How do you archive your data?

Short break



Over to the R quiz

Short break



A very generic schematic of data structures

	var1	var2
Person A		value 1
Person B	value 1	value 2
Person C	value 2	value 3

	Person A	Person B	Person C
var1	value1	value1	value1
var2	value2	value2	value2

- Elements:
  - Observations (here: persons with 3 possible values)
  - Variables (here: two variables, may constitute time points or other scenarios of repetition/replication)
  - Values / results (here 5 or 6 values, depending on how we count the NA in table 1)x

name	variable	value
Person A	1	value 1
Person B	1	value 2
Person C	1	value 3
Person A	2	value 4
Person B	2	value 5
Person C	2	value 6

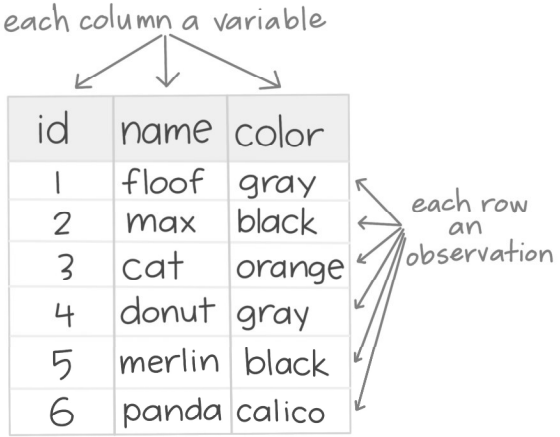
Exkurs: Tidy data (Wickham, 2014)

R processes and R workflow largely fall into two categories:

> "TIDY DATA is a standard way of mapping the meaning of a dataset to its structure."
>
> –HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement



Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

Clean and messy data

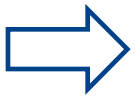
- See Hadley Wickham's Tidy data article, URL: https://vita.had.co.nz/papers/tidy-data.pdf

2. Defining tidy data

> Happy families are all alike; every unhappy family is unhappy in its own way
>
> Leo Tolstoy

Like families, tidy datasets are all alike but every messy dataset is messy in its own way. Tidy datasets provide a standardized way to link the structure of a dataset (its physical layout) with its semantics (its meaning). In this section, I'll provide some standard vocabulary for describing the structure and semantics of a dataset, and then use those definitions to define tidy data.

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.



See article for examples.
See linked Github repo for code examples.

Aufbau von Datendateien

variables in columns

All data in one file (even longitudinal data)
Sometimes it is advisable to split different data formats into several files (or directly interact with an SQL database)

Observations in rows

one observation = one row

variable names follow a specific format

All data follows one coding scheme

Categories are numbers, not text

ID numbers instead of names or other identifiable information

All cells have values

Missing value codes or NA

Categorical data: coding scheme

id	alter	sex	größe	gewicht	bmi	nation	anzahlkh	bereich	raucher	aktiv	aktiv_daue	zufrieden	besch_n	besch_n_f	besch_n_d
43	66	2	180	68	21,00	0	3	0	0	1	1	2	2	0	4
14	38	1	166	60	21,80	0	35	1	1	0	0	1	3	0	4
	41					0	1	1	0	1	3				2
	32					0	0	1	0	1	3				2
22	56					0				1	1				4
30	61	2	175	75	24,50	0				1	1	3	2	0	4
33	27	1	175	86	28,10	0				1	1	3	2	0	4
36	59	2	169	61	21,40	0	10	1	0	0	0	2	3		
39	61	1	175	73	23,80	0	8	1	0	0	0	2	3		
41	56	2	165	55	20,20	0	4	1	0	1	1	2	2		
42	29	2	162	69	26,30	0	1	1	1	1	1	2	2	0	3
51				48	17,60	0	11	1						0	0
55				85	26,20	0	4	1						0	0
56				72	22,20	0	6	1	0	0	0	2	0	0	0
59				92	27,50	0	4	1	0	1	1	3	2	0	2
61				82	23,40	0	1	1	0	1	3	1	1	0	2
63	31	1	184	95	28,10	0	110	1	1	0	0	0	3	0	4
66	47	2	184	75	22,20	0	4	1	0	1	2	3	2	0	3
88	46	1					1	1	1	1	0				
89	9999	2					1	1	0	1	4				
96	64	2					7	1	0	1	2				
97	34	1	172	68	23,00	0	3	1	0	1	2	3	2	0	4
99	28	2	185	70	20,50	0	3	1	0	1	2	3	2	0	1
104	56	2	180	75	23,10	0	1	1	0	1	4	1	1	0	3
105	64	1	174	72	23,80	0	42	1	1	0	0	0	4	1	4
108	49	2	182	90	27,20	0	1	1	0	1	0	1	0	0	0

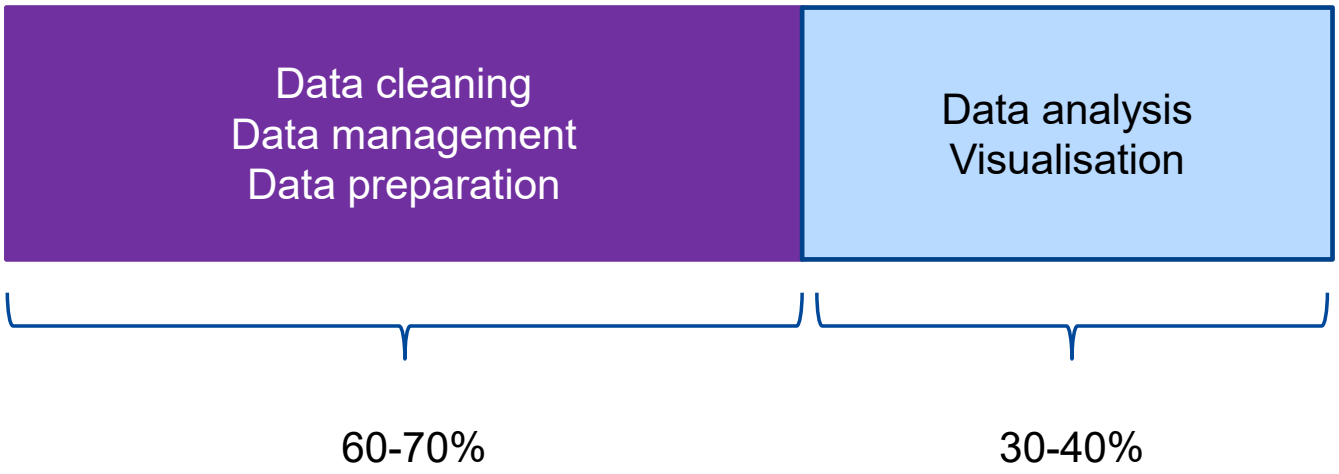
What is a coding scheme?

- Always write a data codebook.
- You can do this in different formats (it doesn‘t really matter as long as your are consistent).

Codebook Dataset HF Pallcare Pilot data

Vrbl no	Variable name	Instrument	Label	Coding	Missing
1	id	-	ID number	continuous	-
2	age	-	Age in years	continuous	no missing
3	sex	-	Gender	0 = male 1 = female	no missing
4	length_care	-	Length of Care (referral date to end in days)	continuous	no missing
5	time_referral	-	Time referral to first telephone call (days)	continuous	no missing
6	time_assess	-	Time referral to 1st assessment F2F	continuous	no missing
7	died	-	Died (censored 31-aug-18)	0 = alive 1 = dead	no missing
8	died_aug20	-	Died (censored 31-aug-20)	0 = alive 1 = dead	no missing
9	time_death	-	Time referral to death (censored aug18)	continuous	no missing
10	time_deathaug20	-	Time referral to death (censored aug20)	continuous	no missing
11	pod	-	Place of death	1 = home 2 = nursing home 3 = hospital 4 = hospice	5 = info missing at aug 20 follow-up sysmis = patient not dead
12	living_alone	-	Lives with	1 = lives alone 2 = lives with family 3 = nursing home or residential home	no missing
13	carer	-	Has carer relationship	0 = no 1 = yes	no missing
14	referral	-	Referral source	1 = General Medical Practitioner 2 = Hospital Cardiology	no missing

The realities of being a data person



How is this dataset messy?

Internal ID	Name	Given name	Sex	Age	Course	Country of bi	Bounty	Twix	Mars	7UP
Pseudo ID	From database			Age in years	Course of study		Categorical	Categorical	Categorical	changed on 3/4
90258773	Doe	John	m	23	Programming	CH	Hate	Best in the world	4	BÄH
90272821	Bumblebee	Agatha	f	20	Biology	CH	Hate	ok	2	okay
90272829	Winterson	Janet	don't want to say	1999	MSc	CH	Hate	999		
90272840	Rocket	Johnny	999	18		Switzerland	Hate	Best in the world	3	ok
90272841	Lewis	Carl	m			America	Hate	Hate	Best in the world	Best in the world
90272852	Ramsenthaler	Christina	f	35	Dozentin		Hate	Hate	Best in the world	4
90272853	Beethoven	Ludwig sine van	m	Music	250		Hate		4	4
90272854	Anonymous	Ano.	f		Bachelor	Mexico	Hate	Best in the world		4
90272858	Anonymous	Ano.	f		Bachelor	Mexico	Hate	Best in the world	999	4
90272858		Mathilda	f			Mexico	Hate	Best in the world	999	4
90272858					BSc	Mexico	Hate	Best in the world		4
90272859	Valentin	Müller	m		Physio	D	Hate	4	BÄH	1
Date	06.05.2020	08.05.2020	03.03.2020	04.03.2020						

- File name: Messydata.xlsx

Tidying messy datasets

- Cleaning messy datasets means coming up with a cleaning data pipeline to transfer datasets from ‚messiness' to tidyness.
- The five most common problems with messy datasets include:
  - 1) Column headers are values, not variable names.
  - 2) Multiple variables are stored in one column.
  - 3) Variables are stored in both rows and columns.
  - 4) Multiple types of observational units are stored in the same table.
  - 5) A single observational unit is stored in multiple tables.

- Tidying messy data (data cleaning) should not be confused with data wrangling!
  - Raw data = messy → data cleaning → tidy data for further analyses
  - Tidy data (in .rds format as master file) → data wrangling for restructuring to fit to purpose → creating tables/graphs/other visualisations

I recommend splitting these two processes in your R projects into subroutines!

- Tidying messy data involves melting, string splitting, casting functions

1) Column names are values, not variable names

var1	var<10	var10-20	var20-30	var30-40	var40-50	var50-60
Category1	27	34	60	81	76	137
Category2	12	27	37	52	35	70
Category3	27	21	30	34	33	58
Category4	418	617	732	670	638	1116
Category5	15	14	15	11	10	35
Category6	575	869	1064	982	881	1486
Category7	1	9	7	9	11	34
Category8	228	244	236	238	197	223
Category9	20	27	24	24	21	30
Category10	19	19	25	25	30	95

row	a	b	c
A	1	4	7
B	2	5	8
C	3	6	9

(a) Raw data

row	column	value
A	a	1
B	a	2
C	a	3
A	b	4
B	b	5
C	b	6
A	c	7
B	c	8
C	c	9

(b) Molten data

var1	var2	freq
Category1	var<10	27
Category1	var10-20	34
Category1	var20-30	60
Category1	var30-40	81
Category1	var40-50	76
Category1	var50-60	137
Category2	var<10	12
Category2	var10-20	27
Category2	var20-30	37
Category2	var30-40	52
Category2	var40-50	35
Category2	var50-60	70

We need to make this into a long dataset (from wide).

= Melting a dataset

= The result is a molten dataset.

melt() function

2) Multiple variables stored in one column

- After melting data from wide to long format, a column variable name might become a combination of multiple underlying variable names.
- This usually happens when data in wide format combined values from categorical variables. Often, frequency is combined with a category of data.

- Example: WHO tuberculosis dataset (number of tuberculosis cases) by
  - country: abbreviation for country
  - year: YYYY format
  - m014 to m65: males in different age categories 0 to 14, 15 to 24, 25 to 34, 35 to 44, 45 to 54, 55 to 64, 65 and older
  - mu: end of data for males
  - f014 to f65: females in different age categories 0 to 14, 15 to 24, 25 to 34, 35 to 44, 45 to 54, 55 to 64, 65 and older
- After melting the dataset, age group and sex (male/female) are combined in one variable. We need to split this variable into two.

2) Multiple variables stored in one column

country	year	m014	m1524	m2534	m3544	m4554	m5564	m65	mu	f014
AD	2000	0	0	1	0	0	0	0	—	—
AE	2000	2	4	4	6	5	12	10	—	3
AF	2000	52	228	183	149	129	94	80	—	93
AG	2000	0	0	0	0	0	0	1	—	1
AL	2000	2	19	21	14	24	19	16	—	3
AM	2000	2	152	130	131	63	26	21	—	1
AN	2000	0	0	1	2	0	0	0	—	0
AO	2000	186	999	1003	912	482	312	194	—	247
AR	2000	97	278	594	402	419	368	330	—	121
AS	2000	—	—	—	—	1	1	—	—	—

2) Multiple variables stored in one column

country	year	column	cases
AD	2000	m014	0
AD	2000	m1524	0
AD	2000	m2534	1
AD	2000	m3544	0
AD	2000	m4554	0
AD	2000	m5564	0
AD	2000	m65	0
AE	2000	m014	2
AE	2000	m1524	4
AE	2000	m2534	4
AE	2000	m3544	6
AE	2000	m4554	5
AE	2000	m5564	12
AE	2000	m65	10
AE	2000	f014	3

(a) Molten data

country	year	sex	age	cases
AD	2000	m	0-14	0
AD	2000	m	15-24	0
AD	2000	m	25-34	1
AD	2000	m	35-44	0
AD	2000	m	45-54	0
AD	2000	m	55-64	0
AD	2000	m	65+	0
AE	2000	m	0-14	2
AE	2000	m	15-24	4
AE	2000	m	25-34	4
AE	2000	m	35-44	6
AE	2000	m	45-54	5
AE	2000	m	55-64	12
AE	2000	m	65+	10
AE	2000	f	0-14	3

(b) Tidy data

2) Multiple variables stored in one column

country	year	column	cases
AD	2000	m014	0
AD	2000	m1524	0
AD	2000	m2534	1
AD	2000	m3544	0
AD	2000	m4554	0
AD	2000	m5564	0
AD	2000	m65	0
AE	2000	m014	2
AE	2000	m1524	4
AE	2000	m2534	4
AE	2000	m3544	6
AE	2000	m4554	5
AE	2000	m5564	12
AE	2000	m65	10
AE	2000	f014	3

(a) Molten data

```r
#melt TB dataset
View(raw)

library(reshape2)
clean <- melt(raw, id = c("country", "year"), na.rm = TRUE)
names(clean)[3] <- "column"
names(clean)[4] <- "cases"

clean <- arrange(clean, country, column, year)
View(clean)
```

2) Multiple variables stored in one column

country	year	sex	age	cases
AD	2000	m	0-14	0
AD	2000	m	15-24	0
AD	2000	m	25-34	1
AD	2000	m	35-44	0
AD	2000	m	45-54	0
AD	2000	m	55-64	0
AD	2000	m	65+	0
AE	2000	m	0-14	2
AE	2000	m	15-24	4
AE	2000	m	25-34	4
AE	2000	m	35-44	6
AE	2000	m	45-54	5
AE	2000	m	55-64	12
AE	2000	m	65+	10
AE	2000	f	0-14	3

(b) Tidy data

```r
clean$sex <- str_sub(clean$column, 1, 1)

#Now we split up the age groups into an age vector
#The following names the elements of the vector and gives the end
#format of the year ranges as they should appear later in the variable
#age.
ages <- c(`04` = "0-4",
          `514` = "5-14",
          `014` = "0-14",
          `1524` = "15-24",
          `2534` = "25-34",
          `3544` = "35-44",
          `4554` = "45-54",
          `5564` = "55-64",
          `65` = "65+",
          u = NA)

#Here, we extract the age groups from the original columns and use the
#named vector ages to pass the categories as factor levels. We store this
#information into the age column in the clean dataset.
clean$age <- factor(ages[str_sub(clean$column, 2)], levels = ages)
class(clean$age)
levels(clean$age)

#Reorder the dataset and drop column
clean <- clean[c("country", "year", "sex", "age", "cases")]
```

3) Variables are stored in both rows and columns

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

- Most complicated form of messy data.
- Weather data from the Global Historical Climatology Network for one weather station in Mexico (id var) for five months in 2010.
- Variables are
  - in individual columns (id, year, month)
  - spread across columns (day, d1-d31; d9 to d31 not shown)
  - spread across rows (tmin, tmax) (minimum and maximum temperature) → element variable not a variable, it stores names of variables.

3) Variables are stored in both rows and columns

- We first deal with the business of getting the days into one variable via melting.

id	date	element	value
MX17004	2010-01-30	tmax	27.8
MX17004	2010-01-30	tmin	14.5
MX17004	2010-02-02	tmax	27.3
MX17004	2010-02-02	tmin	14.4
MX17004	2010-02-03	tmax	24.1
MX17004	2010-02-03	tmin	14.4
MX17004	2010-02-11	tmax	29.7
MX17004	2010-02-11	tmin	13.4
MX17004	2010-02-23	tmax	29.9
MX17004	2010-02-23	tmin	10.7

(a) Molten data

We now need to deal with this col not being a variable but names of variables. → casting this into two variables.

```
raw <- read.csv2("C:/Users/User/Downloads/v59i10-data/weather.csv",
                 header = TRUE)
names(raw)[1] <- "id"

# Melt and tidy
clean1 <- melt(raw, id = 1:4, na.rm = TRUE)
#Clean day by using str_replace function from string_r
#Replaces pattern "d" with nothing "" (so essentially deletes the leading d)
#Then the variable is put as integer.
clean1$day <- as.integer(str_replace(clean1$variable, "d", ""))
#Using ISOdate function
clean1$date <- as.Date(ISOdate(clean1$year, clean1$month, clean1$day))

#Reorder variables in dataset
clean1 <- clean1[c("id", "date", "element", "value")]
#Sort variables in dataset
clean1 <- arrange(clean1, date, element)
```

3) Variables are stored in both rows and columns

- We have two measured variables in this dataset: the minimum and maximum temperature.

id	date	tmax	tmin
MX17004	2010-01-30	27.8	14.5
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-02-11	29.7	13.4
MX17004	2010-02-23	29.9	10.7
MX17004	2010-03-05	32.1	14.2
MX17004	2010-03-10	34.5	16.8
MX17004	2010-03-16	31.1	17.6
MX17004	2010-04-27	36.3	16.7
MX17004	2010-05-27	33.2	18.2

(b) Tidy data

```
# Cast (make two variables)--------------
clean2 <- dcast(clean1, ... ~ element)
```

The cast operation is an unstack operation. It is the opposite of melting. We essentially make columns out of rows.
The syntax reads as take data frame clean 1, ignore all elements, then do the inverse of melting by rotating the element variable ~ back out into the columns.

4) Multiple types in one table

	year	artist	track	time	date.entered	wk1	wk2	wk3	wk4
1	2000	2 Pac	Baby Don't Cry	4:22	2000-02-26	87	82	72	7
2	2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02	91	87	92	N
3	2000	3 Doors Down	Kryptonite	3:53	2000-04-08	81	70	68	6
4	2000	3 Doors Down	Loser	4:24	2000-10-21	76	76	72	6
5	2000	504 Boyz	Wobble Wobble	3:35	2000-04-15	57	34	25	1
6	2000	98?	Give Me Just One Nig...	3:24	2000-08-19	51	39	34	2
7	2000	A*Teens	Dancing Queen	3:44	2000-07-08	97	97	96	9
8	2000	Aaliyah	I Don't Wanna	4:15	2000-01-29	84	62	51	4
9	2000	Aaliyah	Try Again	4:03	2000-03-18	59	53	38	2
10	2000	Adams, Yolanda	Open My Heart	5:30	2000-08-26	76	76	74	6
11	2000	Adkins, Trace	More	3:05	2000-04-29	84	84	75	7

Datasets often involve values collected at multiple levels, on different types of observational units.
Tidying may involve storing each type of observational unit in its own table. → Database normalisation.
The Billboard dataset contains:

- Information on different songs. → Observational unit is the song (artist, time)
- Information on the ranks of songs in the charts. → Observational unit is the rank each week.

4) Multiple types in one table

Tidying results in number of weeks being given as rows = observational units.
Song information is repeated for each week the song has a ranking.

	year	artist	time	track	date	week	rank
1	2000	2 Pac	4:22	Baby Don't Cry	2000-02-26	1	87
2	2000	2 Pac	4:22	Baby Don't Cry	2000-03-04	2	82
3	2000	2 Pac	4:22	Baby Don't Cry	2000-03-11	3	72
4	2000	2 Pac	4:22	Baby Don't Cry	2000-03-18	4	77
5	2000	2 Pac	4:22	Baby Don't Cry	2000-03-25	5	87
6	2000	2 Pac	4:22	Baby Don't Cry	2000-04-01	6	94
7	2000	2 Pac	4:22	Baby Don't Cry	2000-04-08	7	99
8	2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-02	1	91
9	2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-09	2	87
10	2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-16	3	92
11	2000	3 Doors Down	3:53	Kryptonite	2000-04-08	1	81
12	2000	3 Doors Down	3:53	Kryptonite	2000-04-15	2	70

4) Multiple types in one table

id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of ...	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98^0	Give Me Just One Nig...	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice Deejay	Better Off Alone	6:50

id	date	rank
1	2000-02-26	87
1	2000-03-04	82
1	2000-03-11	72
1	2000-03-18	77
1	2000-03-25	87
1	2000-04-01	94
1	2000-04-08	99
2	2000-09-02	91
2	2000-09-09	87
2	2000-09-16	92
3	2000-04-08	81
3	2000-04-15	70
3	2000-04-22	68
3	2000-04-29	67
3	2000-05-06	66

Table 13: Normalised billboard dataset split up into song dataset (left) and rank dataset (right). First 15 rows of each dataset shown; `genre` omitted from song dataset, `week` omitted from rank dataset.

4) Multiple types in one table

- This problem is the most common problem that befalls the data person. This happens in projects which collect information on observational units that are stationary (i.e., patient characteristics) but also collect information that changes for observational units (i.e., different time points).
- There are different solutions and which one you choose depends on your field of research and the overall goal of the data analysis project.

SQL / Relational database structure	Import 1 dataset Maintaining 2+ datasets	Import 1 dataset Maintain 1 dataset
You opt to do all your data holding (and also data cleaning) in a separate system. R has the facility to interact with SQL systems via packages: DBI dplyr & dbplyr odbc https://solutions.rstudio.com/db/getting-started/database-queries/	You import 1 file into R. You split them into individual datasets and clean them. You use procedures for joining datasets in data wrangling and for data analysis. CAVE: Major headache when factor vars are involved.	You import 1 file into R. You either decide to keep this consistently in wide format or in long format. You do all the data cleaning. You then use procedures of splitting the file for individual analytical steps. CAVE: Involves lots of rowwise calculations via group_by if in long format.

5) One type in multiple tables

- We may also have the problem of having data values spread over multiple tables or files.
- These tables and files are often split up by another variable, so that each represents a single year, person, or location.
- As long as the format for individual records is consistent, this can be fixed via:
  - 1) Read the files into a list of tables.
  - 2) For each table, add a new column that records the original file name (because the file name is often the value of an important variable, i.e., date)
  - 3) Combine all tables into a single table or gradually join tables with the existing dataset (if individual data files drop in one by one).
- Very common scenario in outbreak monitoring (COVID Machine Learning Project: 1 file per day, all have the same structure).

5) One type in multiple tables

```
paths <- dir("data", pattern = "\\.csv$", full.names = TRUE)
names(paths) <- basename(paths)
ldply(paths, read.csv, stringsAsFactors = FALSE)
```

- Scenario 1: You have all files in 1 directory.
  - plyr package
  - Working with dir() function also explained
  - in https://epirhandbook.com/en/directory-interactions.html
  - The code generates a vector 'paths' in a directory "data" that matches a regular expression (ends in .csv).
  - Next, we name each element of the vector with the name of the file. Plyr preserves each name in the following step, ensuring that each row in the final data drame is labelled with its source.
  - ldply() loops over each path, reading in the csv file and combining the results into a single dataframe.
- Scenario 2: You get a new file containing new cases and need to join to a master file.
  - You import the new file. You clean the new file.
  - After cleaning, you import your master file (in .rds format).
  - You then add the rows of the new file to the master file, saving the master file anew in .rds format. This only works when new file and master file have the exact same structure of vars with classes (and modes).

Further tools for tidying data and data wrangling

- Further functions for tidying data:
  - Filter: subsetting or removing observations based on some condition.
  - Transforming: adding or modifying variables (can involve single variable or multiple variables)
  - Aggregating: collapsing multiple values into a single value (e.g., by summing or taking means)
  - Sort: changing the order of observations.

- Visualisations
- Modelling

- All this has been poured into a wonderful series of case studies on Youtube:
  - David Robinson‘s Tidy Tuesday Screencast series
  - https://www.youtube.com/user/safe4democracy/playlists

How do I know whether I need to start this process?

- Your data – especially if it‘s data that is collected in a larger consortium or for a group of people - may likely be already in a format that can be analysed without needing major tidying activities.
- Your data – especially if it‘s data that you have collected yourself or if it‘s data for which you oversee the process of data collection - can be already brought into tidy format before you do the adaptation of the dataset into R.
- If you are querying SQL databases, it is a good idea to learn SQL and to use an R package that works with relational databases.
- If you get your data from governmental websites, APIs, via webscraping or from people who have no idea what tidy data looks like, you will definitely need to start the process of tidying your data.

- All of this can also be done for most datasets with WYSIWYG programmes like Excel. It just takes longer.
- There is also the danger of having cleaning steps in a WYSIWYG programme that are not documented and therefore cannot be replicated.

Bottom line – start gradually

- It is perfectly acceptable to gradually learn your way around these things if you need to start tidying large portions of data. Start small, creating little toy datasets, gradually expand.
- You may start by using Excel or similar programmes to do the initial cleaning.
  - Pro Tipp: Have a notebook or text document open and document each step of what you do into this notebook.
  - Make a careful note of what you do first, then what you do second etc. It is highly likely that you will implement your data tidying in R using the same sequence of steps like you did in Excel.
  - If possible, condense these steps into a map/checklist that is true for all datasets you typically analyse.
  - Put this checklist as a poster close to your workspace. You can make it pretty.
- Strive to transition this process gradually into R.
  - Take the different steps involved and analyse carefully for each step which functions from the tidyverse and from other packages might be helpful.
  - You can either implement parts of this process in R, keeping other parts in Excel at first, or transition in one go.
  - You may also want to gradually built a file of cheat sheets and notes/instructions for yourself of solutions you have found for common data tidying problems involved for your data.

Major packages used in tidying

- This heavily relies on the melt() and cast() functions in the reshape2 package.
- The reshape2 package is now retired.
- -> https://rdocumentation.org/packages/reshape2/versions/1.4.4

- Still sometimes a nice package to use because the syntax is very clear.
- See: https://seananderson.ca/2013/10/19/reshape/

- Has been replaced by the tidyr package
  - pivot_wider()
  - pivot_longer()

reshape2

Status

reshape2 is retired: only changes necessary to keep it on CRAN will be made. We recommend using tidyr instead.

Introduction

Reshape2 is a reboot of the reshape package. It's been over five years since the first release of reshape, and in that time I've learned a tremendous amount about R programming, and how to work with data in R. Reshape2 uses that knowledge to make a new package for reshaping data that is much more focused and much much faster.

This version improves speed at the cost of functionality, so I have renamed it to `reshape2` to avoid causing problems for existing users. Based on user feedback I may reintroduce some of these features.

What's new in `reshape2`:

- considerably faster and more memory efficient thanks to a much better underlying algorithm that uses the power and speed of subsetting to the fullest extent, in most cases only making a single copy of the data.

Tidyr package

- https://tidyr.tidyverse.org/

- tidyr function fall into 5 categories, all useful for tidying
  - Pivotting (essentially melt() and cast() from reshape2)
  - Rectangling (flattening lists into data frames, mainly helpful when interacting with APIs)
  - Nesting (converting grouped data so that each group becomes a single row containing a nested data frame) or unnesting (the opposite)
  - Splitting and combining character columns (separate(), extract(), unite()) – an easier implementation of stringr functions
  - Functions for making NAs explicit

tidyr

Overview

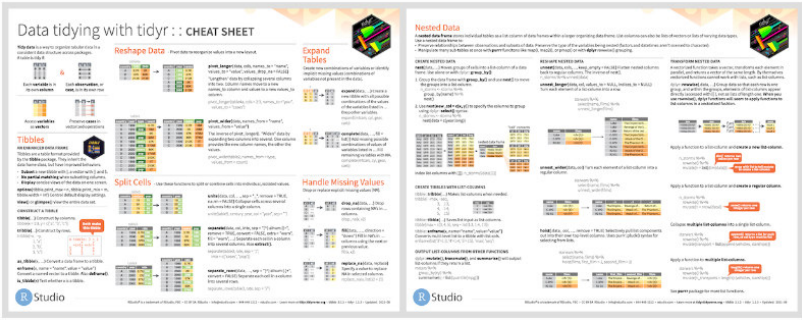
The goal of tidyr is to help you create **tidy data**. Tidy data is data where:

1. Every column is variable.
2. Every row is an observation.
3. Every cell is a single value.

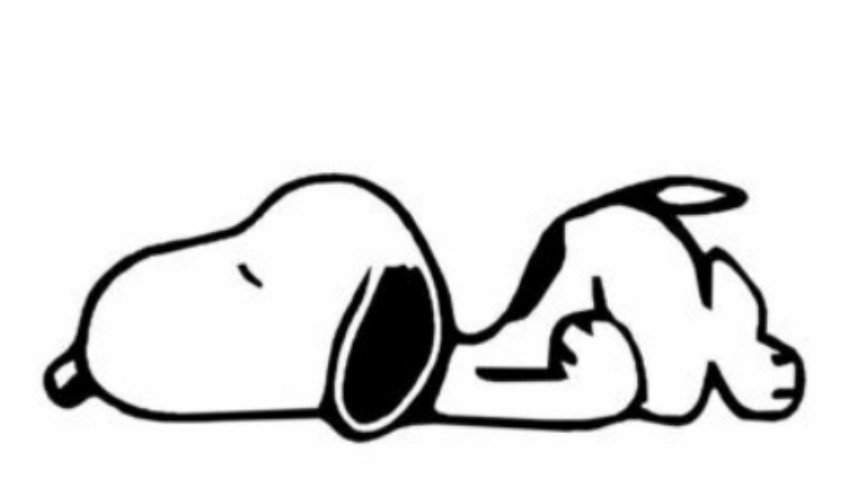
Tidy data describes a standard way of storing data that is used wherever possible throughout the tidyverse. If you ensure that your data is tidy, you'll spend less time

Cheatsheet

out tidy



Short break



“The“ system, aka Working with data in R projects workflow

Phases in any data project

Aims & objectives of the project
Study protocol
Statistical analysis plan

- Never fly without a protocol
- Aims & objectives dictate types of data to be collected
- SAP always useful for specifying which variables are important and which are not, which analyses are a priori and which are post-hoc

↓

Collecting data

- If possible, collect clean data.
- If possible, set up procedures that support clean data (i.e., data dictionary, double data entry etc).
- Put datafile metainformation into place.

↓

Importing and cleaning data to arrive at masterfile in .rds format.

- Either importing a pre-cleaned file or doing all the tidying in R.
- Cleaning also involves setting up all calculated variables.
- .Rds format keeps dataframe class & mode of each variable intact.

↓

Analysing the masterfile using different methods

- Using the split-apply-combine paradigm: Taking vars & | obs from masterfile, applying functions, combining results into different formats

↓

Communicating results

- Different formats from saved graphs to full Shiny dashboard.

The container around all this: The R project

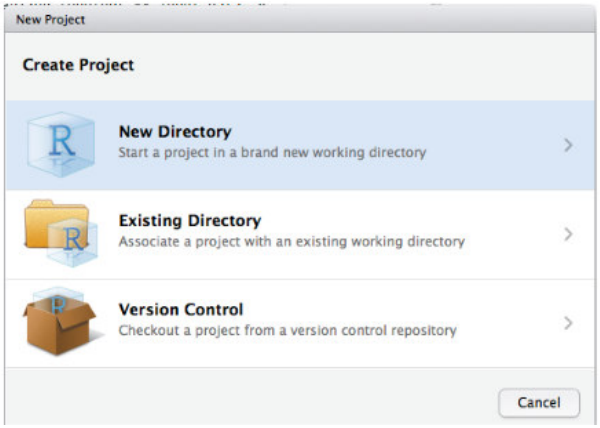
- https://r4ds.had.co.nz/workflow-projects.html

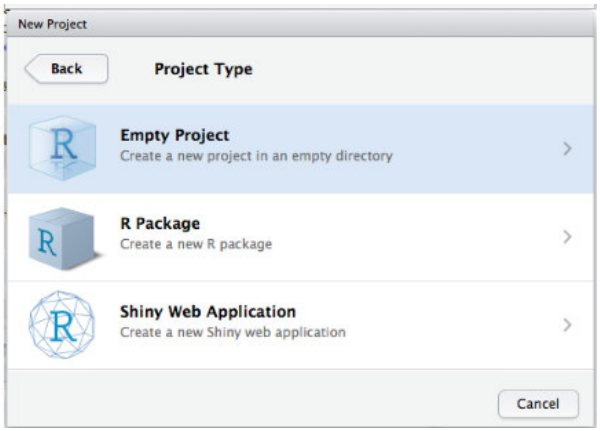
8.4 RStudio projects

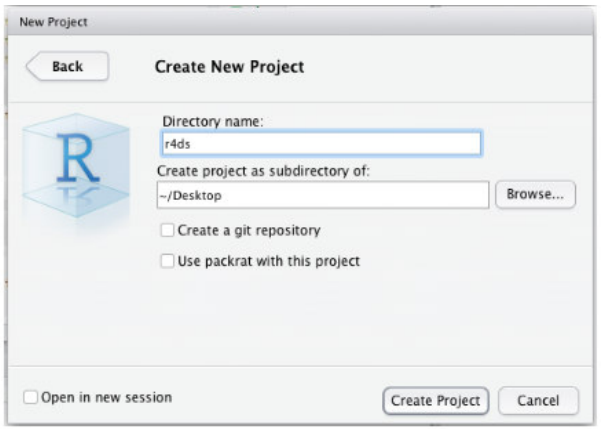
R experts keep all the files associated with a project together — input data, R scripts, analytical results, figures. This is such a wise and common practice that RStudio has built-in support for this via **projects**.

Let's make a project for you to use while you're working through the rest of this book. Click File > New Project, then:

- Create an RStudio project for each data analysis project
- Keep data files there
- Keep scripts there; edit them, run them in bits or as a whole.
- Save your outputs (plots and cleaned data) there.
- Only ever use **relative paths**, not absolute paths.







What goes into R projects?

data_nr	sort_id	fall_id	patient_id	episode_n	sex
1	1	1001	1001	1	2
1	2	1002	1002	1	2
1	3	1003	1003	1	2
1	4	1004	1004	1	2
1	5	1005	1005	1	2
1	6	1006	1006	1	1
1	7	1007	1007	1	1

Raw data files

Datafile metadata

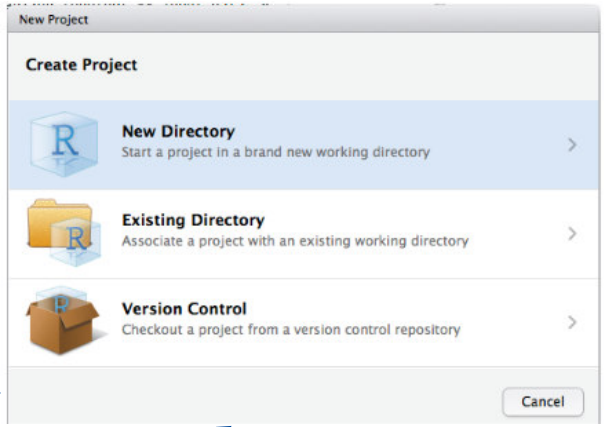
Codebook Dataset HF Pallcare Pilot data

Vrbl no	Variable name	Instrument	Label	
1	id	-	ID number	continuc
2	age	-	Age in years	continuc
3	sex	-	Gender	0 = male 1 = fema
4	length_care	-	Length of Care (referral date to end in days)	continuc
5	time_referral	-	Time referral to first	continuc

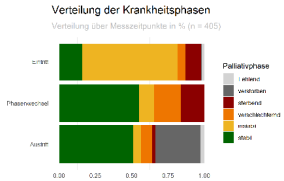
Notes / notebooks

	id	date	tmax	tmin
1	MX17004	2010-01-31	27.8	14.5
2	MX17004	2010-02-02	27.3	14.4
3	MX17004	2010-02-03	24.1	14.4

Data masterfiles



Graphs / tables




```
### Barthel-Item: Treppe steigen
Im Barthel-Item *Treppe steigen* zeigt sich e
mit 75%, die pflegeabhängig sind.

```{r treppe, echo=FALSE, out.width = '70%'}
knitr::include_graphics("dt_treppe.png")
```
```

R Markdown for reports

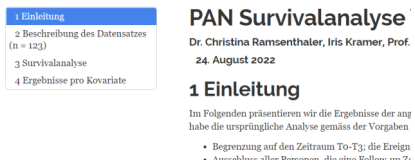
R scripts

```
################
## 3.4 Multiple
################

library(lubrida

raw <- read.csv
View(raw)
```

R Markdown websites



(Shiny) dashboards

Supporting information, for instance:
Manuals of how to calculate scores for outcome measures (calculated values)

Where do R projects live on your machine?

Option 1: They simply are nested within a parent folder for the whole project.

- Pro: All your files are within the same parent structure. Whenever you want to access this project, this will be in place.
- Con: Slightly harder to use with Git version control as commits need to involve the correct file structure.
- May make version control harder if projects go through iterations repeatedly.

Option 2: You have a dedicated space for all R projects.

- Pro: Absolves you from searching for files within large parent directories.
- May make version control easier (particularly with the luxury of dedicating one harddrive for R projects).
- Con: Makes it harder to find stuff after a long time. May also lead to breaking links between analyses and reports or papers. Essentially creates a duplicated structure of content-related files in one folder and analysis files in another place.

Option 3: You mix and match.

- If you work with many different partners or want to follow a different structure for work and for private projects, you may do so anytime.
- You might not have a choice when you need to work on different machines or are forced into certain structures by your workplace.
- Con: Gets messy quickly and you will inevitably forget what lives where without documentation.

Where do R projects live on your machine?

- R projects should never live in the cloud!
- This may become a major headache in case your workplace forces you to work with sharepoint/onedrive.
- It is always safest to have an R project implemented on a harddrive.
- For folks from the university hospital:
  - If you are prohibited from using USB drives or upload to the cloud from your work computer, you could try to ask nicely whether they would be willing to install Git. You could then push changes to Github and have a backup there.
  - Cave: Always make sure to never violate data security. Check with your department head what is allowed and possible. Sometimes you need to ignore files from your project (or whole subfolders, e.g. the raw data folder when pushing to Git – specify .gitignore).
  - If data protection is difficult to reconcile with version control or file management, then consider removing all identifiable information from your data files. Most often, you will not need this anyway. Strive to have a master file that is anonymised so that you can use version control.
- If you use Option 3 (mix of systems), strive to use the same system for one group of projects.

What about naming stuff?

What makes the following examples good or bad names?
How could we improve the bad names?
Do you have a consistent naming system?

```
fit models.R
fit-models.R
foo.r
stuff.r
get_data.R
Manuscript version 10.docx
manuscript.docx
new version of analysis.R
trying.something.here.R
plotting-regression.R
utility_functions.R
code.R
```


Naming conventions

- Good file management should involve a consistent naming system.
- It doesn‘t matter whether you use _01, _02, _03 or prepend each file with a date.
  - analysis_SPC_01.R
  - 2022-10-15_analysis_SPC.R
- If you use dates, make sure to be consistent.
  - Either use international date standard of YYYY-MM-DD or use European style DD-MM-YYYY
  - Never ever switch to a different system in between.
- Naming conventions for files will majorily depend on whether you:
  - Implement a version control system via your file naming.
  - Implement a version control system via Git.
- I tend to do both (but I am often in a situation where I work without an internet connection which prohibits committing changes via Git. I am kind of old school and that‘s alright).
- Pro Tip: Write a small Style Guide document for yourself. Put it above your desk.

My naming conventions

- Files:
  - munge_IRR_01.R (underscore, usually NOT with date unless it‘s the only version control)
- Vars: use _
  - id
  - i01_t1 : i17_t1 and i01_t2 : i17_t2 (be careful with baseline – always gets a 0)
  - I try to give very short variable names (8 characters); what‘s in a variable is described in Codebook
  - Notice the leading ‚0‘ when I deal with max numbers > 9
- Object names:
  - ALM list results use the . -> model.01 to model.02
  - Again, I use an external documentation system to keep track of what‘s in model 1, model 2…
  - object names for vectors or stuff used in function calls are prepended with a leading .
- Graphs and tables:
  - When exported, get consecutive numbering as abb01 … abbn.png (abb being graph in German) or table01.png … tablen.png

R Style Guide

- Hadley Wickham's style guide:
- https://style.tidyverse.org/index.html
- Google's style guide:
- https://google.github.io/styleguide/Rguide.html
- R bloggers good R programming style:
- https://www.r-bloggers.com/2019/01/%f0%9f%96%8a-r-coding-style-guide/
- Kruschke's chapter in Doing Bayesian Analysis:
- https://rpruim.github.io/Kruschke-Notes/some-useful-bits-of-r.html#style-guide

1 Files

1.1 Names

File names should be meaningful and end in `.R`. Avoid using special characters in file names - stick with numbers, letters, `-`, and `_`.

```
# Good
fit_models.R
utility_functions.R

# Bad
fit models.R
foo.r
stuff.r
```

If files should be run in a particular order, prefix them with numbers. If it seems likely you'll have more than 10 files, left pad with zero:

```
00_download.R
01_explore.R
...
09_model.R
10_visualize.R
```

If you later realise that you've missed some steps, it's tempting to use `02a`, `02b`, etc. However, I think it's generally better to bite the bullet and rename all files.

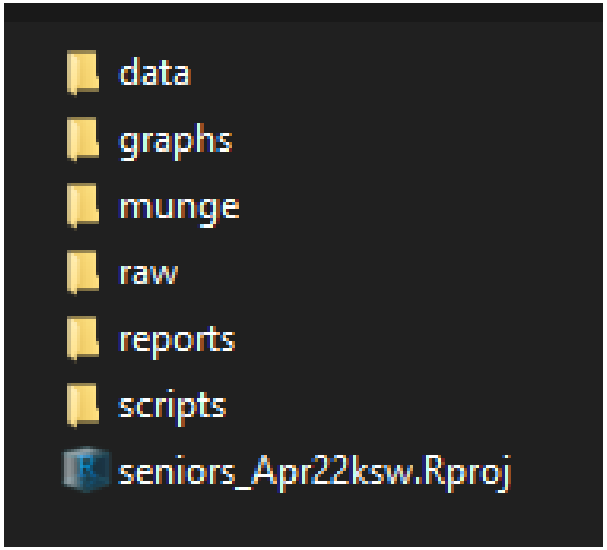
Why are R projects helpful in managing your project?

- Organise all R scripts and files in the same folder (also called "directory") so it is more *self-contained* (doesn't rely on other components in your computer).
- Use a common and consistent folder and file structure for your projects.
- Use version control to track changes to your files.
- Make raw data "read-only" (don't edit it directly) and use code to show what was done.
- Whenever possible, use code to create output (figures, tables) rather than manually creating or editing them.
- Think of your code and project like you do of your manuscript (or any thesis): that other people will eventually look at it and review it, and that it will likely also be published or archived online.

Brain-friendly working involves:

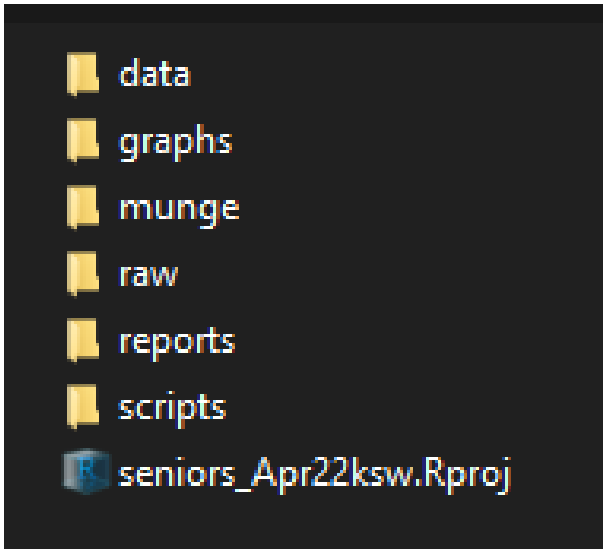
- Having a consistent system in place. The exact components do not matter. The discipline does.
- Understanding that your brain after one night of sleep is a new brain. Treat everything you do for yourself as if you would be doing this for a different person. Document all procedures involved in your research project so that a person not familiar with the project could do the project.

And how do I organise the R project folder itself?



- Example of my most common R project folder structure.
- Features:
  - Have all files except the .Rproj file in subfolders.
  - **data**: Contains masterfiles (result after munging) or csv files of tables or data exporting after analysis
  - **graphs**: Contain all graph and table/visualisation output
  - **munge**: Contains R scripts and metadata in text files and notebooks or pdfs to help with preparing data for analysis
  - **raw**: Contains all the raw data files
  - **reports**: Contains markdown files and pdf, html, or word etc. files containing the results of analysis (basically my analysis communication folder)
  - **scripts**: Contains the analysis.R scripts
- I put this folder structure there myself. However, it is a condensed version of a fuller subfolder structure coming from the ProjectTemplate package.

What goes into my subfolders?

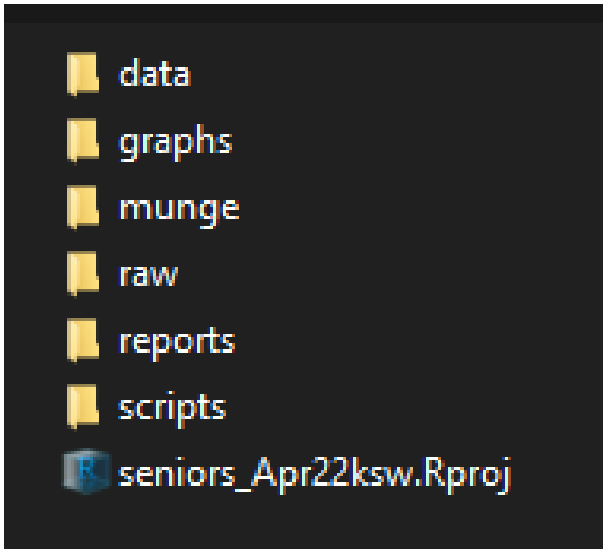


Example for logfile.txt:
name: 20220323_data_seniorsi.csv
date of download: 23/03/2022
des: All routine data from hospital XYZ palliative care ward of all patients on ward from 01/01/2022 until 23/03/2022 1800.

1. Raw data folder raw

- Raw data files to clean or munge in R.
- If more than one: consider putting a date
- Construct a README.txt or Notes.txt/Excel file log containing:
  - Name of each file
  - Date of download/receipt
  - Short description of what it contains (and whether it represents an interim state of a dataset)
- If your data analysis project involves the repeated cleaning of frequently updated files, consider installing an archive subfolder and consistently naming all raw files with dates.
- Only have the most up-to-date raw datafile on this first sublevel.
- The logfile.txt will come in very handy once the project is archived and you need to return to it.

What goes into my subfolders?



Example of contents of a munge subfolder:
- Codebook_seniorsi_01.docx
- Codebook_seniorsi_02.docx
- Codebook_seniorsi_03.docx
- Codebook_seniorsi_04.docx
- Log_Datacleaning.txt
- munge_seniorsi_01.R
- Steps_munge_seniorsi.docx

2. munge

- munge_projectname_01.R: The R script containing everything I need to do in terms of data tidying & munging to get to the masterfile in .rds format.
- Any metadata or supporting documents to help with data cleaning:
  - A document detailing the steps (pre-planned) and checklist for data cleaning
  - A log_munge.txt file detailing all the problems etc I run into during cleaning (might be done using an R Notebook format)
  - At least one iteration of the Codebook for the data
  - Any supporting documents needed for data munging. Most often this involves data manuals when working with data that involves calculation of scores.
  - May contain .xlsx or other files for calculation of scores.
  - May contain an archive folder to only have most up-to-date versions of all munge docs on this sublevel.

Data munging – the Data Codebook

- **A codebook for your data does the following:**
- For the masterfile(s), it details the exact structure of variables for that masterfile.
- This usually involves the following information which should be provided in a table structure:
  - 1. Number of the variable — Useful for absolute indexing in R []
  - 2. Name of the variable — Use consistent naming scheme
  - 3. Variable label — Provide a full & intelligible label to what is measured
  - 4. (Measure) — If consecutive variables belong to a measure (i.e. questionnaire)
  - 5. Coding (for factor vars) — Categorical data (factors): factor levels; date format for date vars, 0/1 vars not factors: levels, integer/numerical: left blank
  - 6. Scale level/mode — integer, numerical, factor, character, logical, date
  - 7. Missing values — whether present y/n, NA or specific code like '999'?

- The codebook is one of the most important elements (actually the most important element) for understanding the structure of a dataset. You will always need one, no matter how small the project. It saves you hours when returning to datasets. I have mine printed out next to the computer for analysis.

Example of a data codebook

- I do mine in Word. You may use any programme that allows a table format (i.e., Excel).
- There is a way of getting automated data codebooks via R & R Markdown with LaTeX integration. I don‘t bother.

Vrbl no	Variable name	Instrument	Label	Coding	Missing
1	id	-	ID number	continuous	-
2	age	-	Age in years	continuous	no missing
3	sex	-	Gender	0 = male 1 = female	no missing
4	length_care	-	Length of Care (referral date to end in days)	continuous	no missing
5	time_referral	-	Time referral to first telephone call (days)	continuous	no missing
6	time_assess	-	Time referral to 1st assessment F2F	continuous	no missing
7	died	-	Died (censored 31-aug-18)	0 = alive 1 = dead	no missing
8	died_aug20	-	Died (censored 31-aug-20)	0 = alive 1 = dead	no missing
9	time_death	-	Time referral to death (censored aug18)	continuous	no missing
10	time_deathaug20	-	Time referral to death (censored aug20)	continuous	no missing
11	pod	-	Place of death	1 = home 2 = nursing home 3 = hospital 4 = hospice	5 = info missing at aug 20 follow-up sysmis = patient not dead

And another data codebook…

Kodierbuch SENIORS-I

Nr.	Name	Label	Kodierung	Skalen-niveau	Fehlende Werte
1	data_nr	Nummer des ursprünglichen Datensatzes	1= Datensatz Nr. 1 2= Datensatz Nr. 2 3= Datensatz Nr. 3	factor	Keine fW
2	sort_id	Sortierungs-ID	-	integer	keine fW
3	fall_id	Fallnummer	Neue distinkte Fallnummer pro Episode	integer	keine fW
4	patient_id	Patientennummer	Distinkte Patientennummer	integer	keine fW
5	episode_nr	Anzahl der Episode pro Patient	-	integer	keine fW
6	sex	Geschlecht	1 = m 2 = w	factor	keine fW
7	yob	Geburtsjahr	yyyy-mm-dd	date	keine fW
8	age	Alter, Aufnahmedatum – Geburtsjahr	-	integer	keine fW
9	age_cat	Altersgruppe	1 = <40 2 = 40-49 3 = 50-59 4 = 60-69 5 = 70-79 6 = 80-89 7 = 90+	factor	keine fW
10	hdiagc	Hauptdiagnose	1 = Krebs	factor	NA

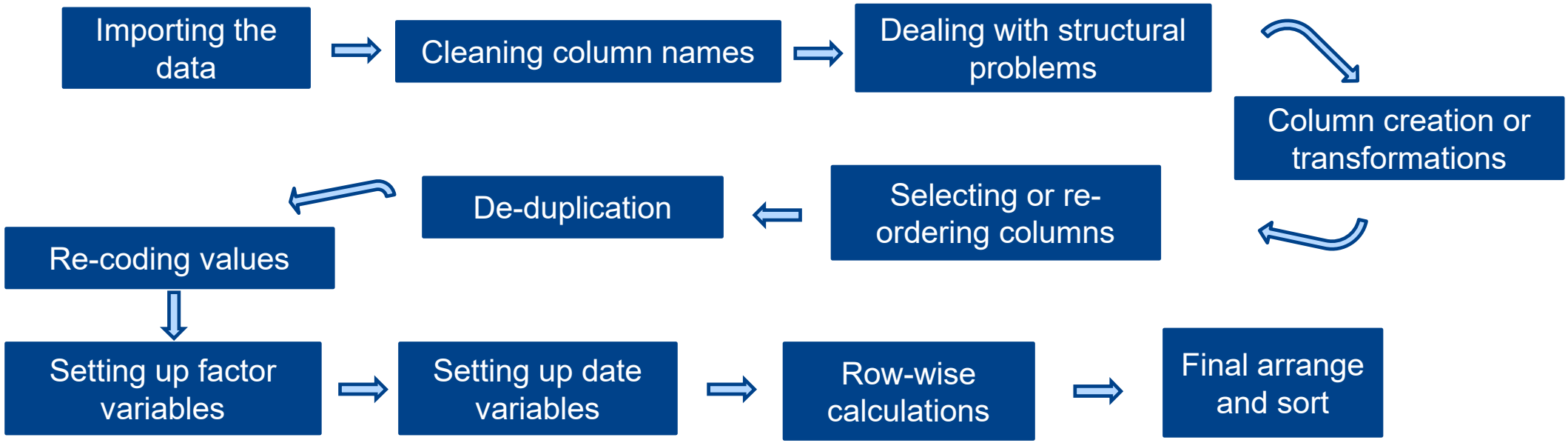
- **You need to keep them up to date!**
- Construct a new version every time you change something on the master file.
- Some projects start with the data codebook, particularly in projects involving the construction of databases or in projects working with REDCap for data collection (or any survey-based project).
- Ideally, the codebook should make it easy to understand which variables are
  - independent variables
  - effect modifiers (moderators)
  - dependent variables (at end of data file)
- Early construction of the codebook may help with understanding cleaning steps to get from the raw to the master file.

Data munging – the checklist

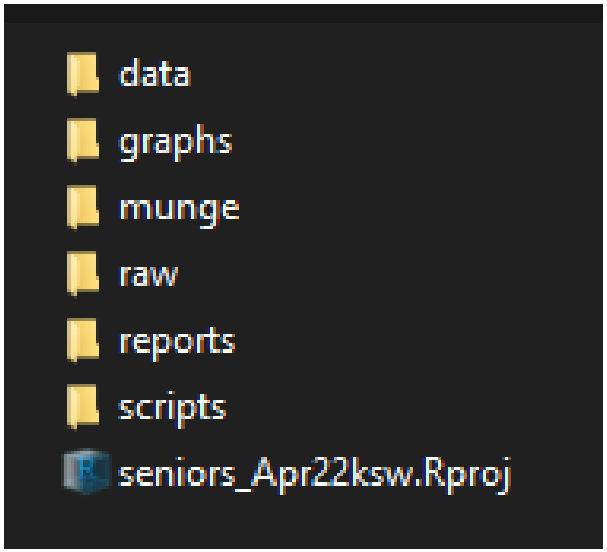
Date of Onset	Sex	Age
1/1/1965	M	24 years
15 March 1994	*NA*	16 months
13 Dec. 1989	Fem	29
25/6/2001	F	3

date_onset	sex	age_years
1965-01-01	Male	24.00
1994-03-15	Missing	1.33
1989-12-13	Female	29.00
2001-06-25	Female	3.00

- It is advisable to write yourself a checklist or a recipe for data cleaning. This should involve all the steps needed to arrive at the data master file.
- You can start this document before you get your raw data and add to it during the process of cleaning.
- Ideally, you list the functions & packages which you will use (can be skipped with experience).
- Data munging usually follows the following steps (we will look into this process in-depth):



What goes into my subfolders?



3. data

- The data folder should contain
  - all master files for a project stored as .rds data format
  - potentially any exported .csv files of tables or data that need to be included as output
- Consider using an archive subfolder for old masterfiles no longer in use.
- The most up-to-date version of the Codebook in the munge folder MUST correspond with the masterfile(s).

4. scripts

- The scripts folder contains analysis scripts in which analysis happens.
  - These can be .R files
  - These can be .Rmd R notebook files
  - These can be .Rmd report files

Scripts: Where do you do the analysis?

- R offers maximum flexibility when it comes to its structures. You need to test and find the workflow that best suits you and your needs. The file type you end up using may also depend on the project itself.
- I would opt for a process that allows you to have a safe playground to test stuff and then have a file containing the final analysis.

Example 1:

- Project only involved analyses with results being fed back via a Powerpoint brochure for analysis recipients.
- Analysis_project_01.R contained all analyses.
- Analysis output was done via graphs.png and tables.png files that were given to project recipients for inclusion into a Powerpoint.

Example 2:

- Project involved writing an academic paper.
- Analysis_project_01.R contained all analyses (also dead ends) and was treated like a notebook/process document.
- Final analyses plus visualisations were put together into .Rmd file for publication.

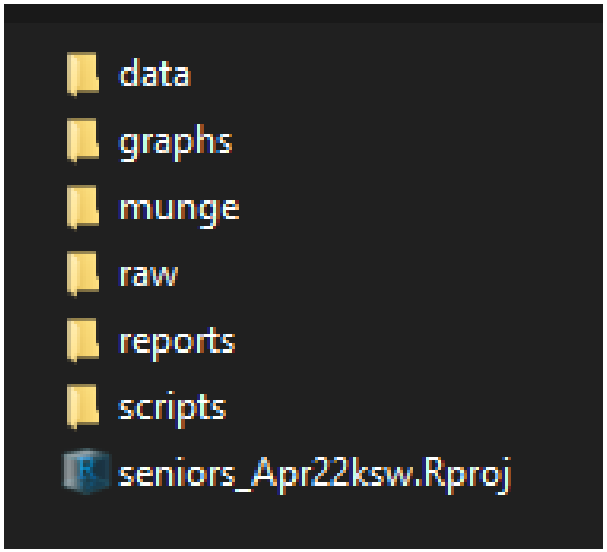
Example 3:

- Project only involved a data dashboard as output.
- Analysis_project_01.Rmd was an R notebook file and kept like a normal lab notebook with complete documentation of the process.
- Analysis output involved building a flexdashboard hosted as a webpage.

Scripts: Where do you do the analysis?

- Much of this boils down to personal taste and how you were socialised (e.g., lab-based environment, social science environment, familiarity with Markdown/LaTeX).
- It doesn‘t matter which way you choose.
- For reproducibility in science, you will need the following ingredients:
  - A playground where you can try out stuff. Ideally, this should still have structure. Also document dead-ends, but also have a system of knowing what the final, implemented version of code was.
    - Implementation in R: Use .R scripts with sumptuous commenting or use .Rmd R Notebooks (work similarly to a lab notebook), see here: https://r4ds.had.co.nz/r-markdown-workflow.html and https://colinpurrington.com/tips/lab-notebooks/
  - One final file which contains the final analysis. This will most often be an R Markdown document.
  - A logbuch system to record your thoughts.
    - Can be incorporated via comments or markdown into R files.
    - May be done in a paper notebook which allows you to jot down ideas quickly.
- To help the creative juices flowing, prefer a format that allows iterative thinking (not linear thinking).

What goes into my subfolders?



5. graphs

- Contains all visualisations (graphs, tables, maps).
- For a reproducible workflow try to use R's built-in tools to save tables directly into either .png or .txt/.docx/.pdf
- Even if the output is an .Rmd document, I tend to save all visualisations into this folder to be able to use them without having to compile/render the Markdown file.
- Pro tip: Name them in the order in which they appear in your doc.
- For hex colour codes, I keep a .txt file containing my codings.

6. reports

- May contain .Rmd files and their output.
- May contain further file types for documenting and disseminating the analysis.
- For academic papers, I may have all files here or transfer output file into my paper folder.

Graphs

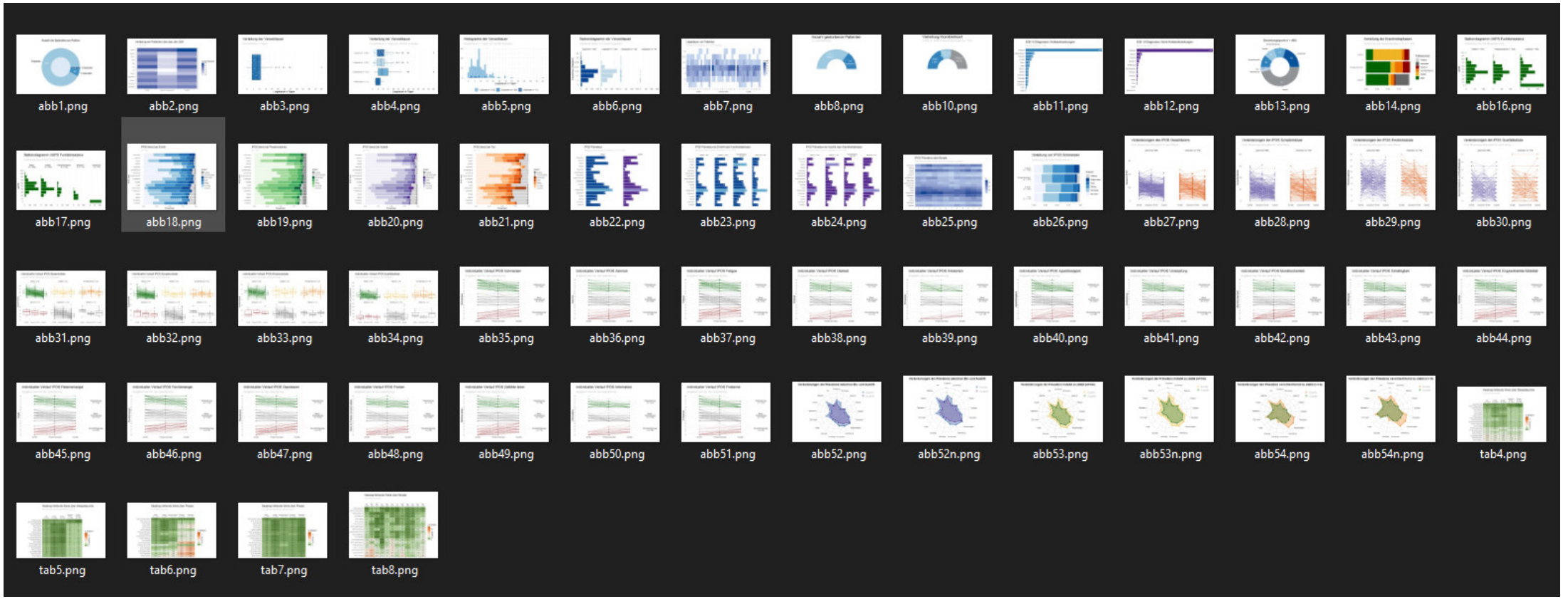


abb1.png
abb2.png
abb3.png
abb4.png
abb5.png
abb6.png
abb7.png
abb8.png
abb10.png
abb11.png
abb12.png
abb13.png
abb14.png
abb16.png
abb17.png
abb18.png
abb19.png
abb20.png
abb21.png
abb22.png
abb23.png
abb24.png
abb25.png
abb26.png
abb27.png
abb28.png
abb29.png
abb30.png
abb31.png
abb32.png
abb33.png
abb34.png
abb35.png
abb36.png
abb37.png
abb38.png
abb39.png
abb40.png
abb41.png
abb42.png
abb43.png
abb44.png
abb45.png
abb46.png
abb47.png
abb48.png
abb49.png
abb50.png
abb51.png
abb52.png
abb52n.png
abb53.png
abb53n.png
abb54.png
abb54n.png
tab4.png
tab5.png
tab6.png
tab7.png
tab8.png

Examples for visualisation tables via gtsummary

QUALIDEM Item	Nie	Sehr selten	Selten	Manchmal	Oft	Häufig	Sehr häufig	Median	Unteres Quartil	Oberes Quartil	MW	SD
A: Pflegebeziehung	19	3	6	13	2	10	3	73.8	69.1	82.2	76.2	12.7
4. Weist Hilfe der Pflegenden ab	2	1	3	2	0	0	0	66.7	62.5	87.6	72.9	19.8
7. Ist verärgert	2	1	1	4	0	0	0	58.4	50.0	87.6	68.8	22.6
14. Hat Konflikte mit den Pflegenden	4	1	0	1	0	2	0	91.7	41.7	100.0	70.8	37.5
17. Beschuldigt andere	5	0	1	1	1	0	0	100.0	62.5	100.0	81.2	27.4
24. Schätzt Hilfe, die er/sie bekommt	0	0	0	2	0	3	3	83.4	75.1	100.0	81.3	20.8
31. Nimmt Hilfe an	0	0	1	1	1	5	0	83.4	62.5	83.4	70.9	19.5
33. Hat an den Routineabläufen etwas auszusetzen	6	0	0	2	0	0	0	100.0	87.5	100.0	87.5	23.1

Tabelle. Deskriptivstatistik Zeit (in Tagen) bis zum letzten MZP

Messzeitpunkt	Distribution	Mean	SD	Median	P25	P75
T0 (n = 1)		63.00		63.00	63.00	63.00
T1 (n = 74)		47.38	43.00	36.50	25.00	47.75
T2 (n = 35)		224.29	100.05	206.00	162.00	260.00
T3 (n = 23)		397.83	116.61	357.00	306.00	519.50
T4 (n = 10)		466.80	125.57	434.50	364.75	516.25
T5 (n = 47)		503.64	79.18	506.00	426.50	557.50

Survival analysis, ZHAW, PAN

Characteristic	N = 200[1]
Chemotherapy Treatment	
Drug A	98 (49%)
Drug B	102 (51%)
Age	47 (38, 57)
Unknown	11
Grade	
I	68 (34%)
II	68 (34%)
III	64 (32%)

[1] n (%); Median (IQR)

Summary: How I move through my folder structure

raw: storing raw data

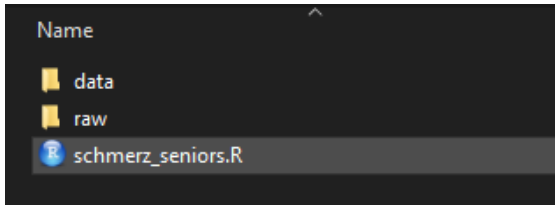
munge: cleaning & munging

data: masterfile

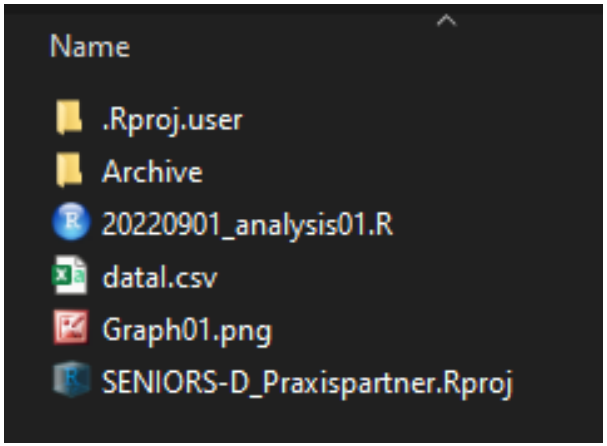
scripts: analysis

graphs: visualisation outputs and reports: .RMD

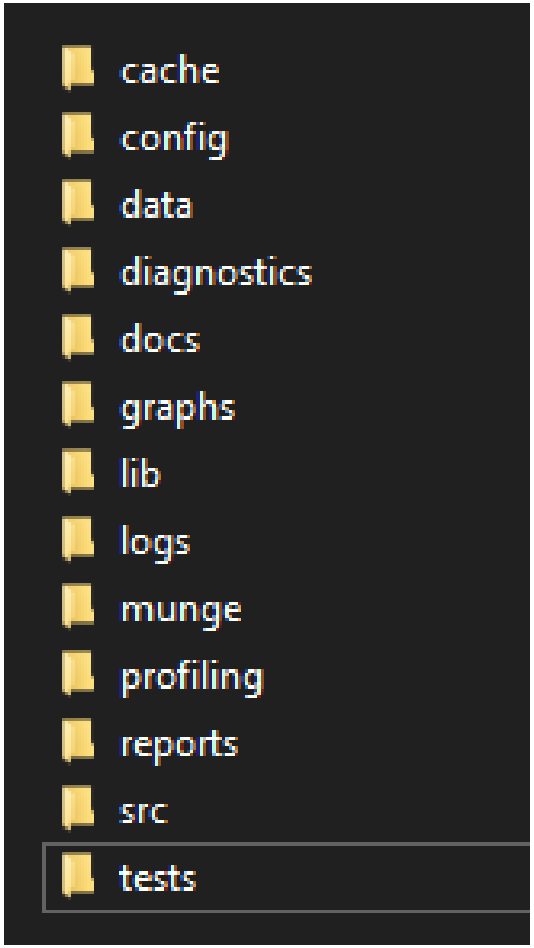
Different subfolder structures depending on need



- Example of a quick & dirty analysis of one symptom.
- Project involved two raw data files
- One master file
- One table (in folder data)
- One analysis script.



- Example of a small project involving a cleaned input file (master file) with data munging & analysis done in one script.
- One output: Graph01.png

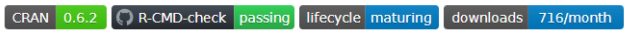


- Example of a full subfolder structure necessary for package development.

A package for subfolder structure in R projects

https://cran.r-project.org/web/packages/prodigenr/readme/README.html

Project creation with prodigenr: A component of reproducible and open scientific projects



This R package is part of a series of (planned) packages that are aimed at creating a toolkit for doing reproducible and open science. Many researchers (especially in biomedicine, medicine, or health, which is my area c little to no knowledge on what open science is or what reproducibility is, let alone how to do it. Our goal is create an (opinionated) toolkit to automate and simplify the process of doing open and reproducible science.

This package has a simple aim of being a *pro*ject *di*rectory *gen*erator (prodigenr), with a simple focus on:

1. Creating a standardized project folder structure with a few template files needed for beginning a data analysis project
2. Following a "one project, one (main) output" principle, such as a report/manuscript in the case of scientific output or a book or website for things like courses or workshops
3. Adhering to established best practices that make it easier for the project to be reproducible and open

This standardized approach to how a scientific project is structured helps ensure that the final code and documents are fairly modular, self-contained, easy to share and make public, and be as reproducible as possible. T makes use of the existing and established applications and workflows (RStudio, devtools, and usethis).

Installation

You can install the released version of prodigenr from CRAN with:

```
install.packages("prodigenr")
```

And the development version from GitHub with:

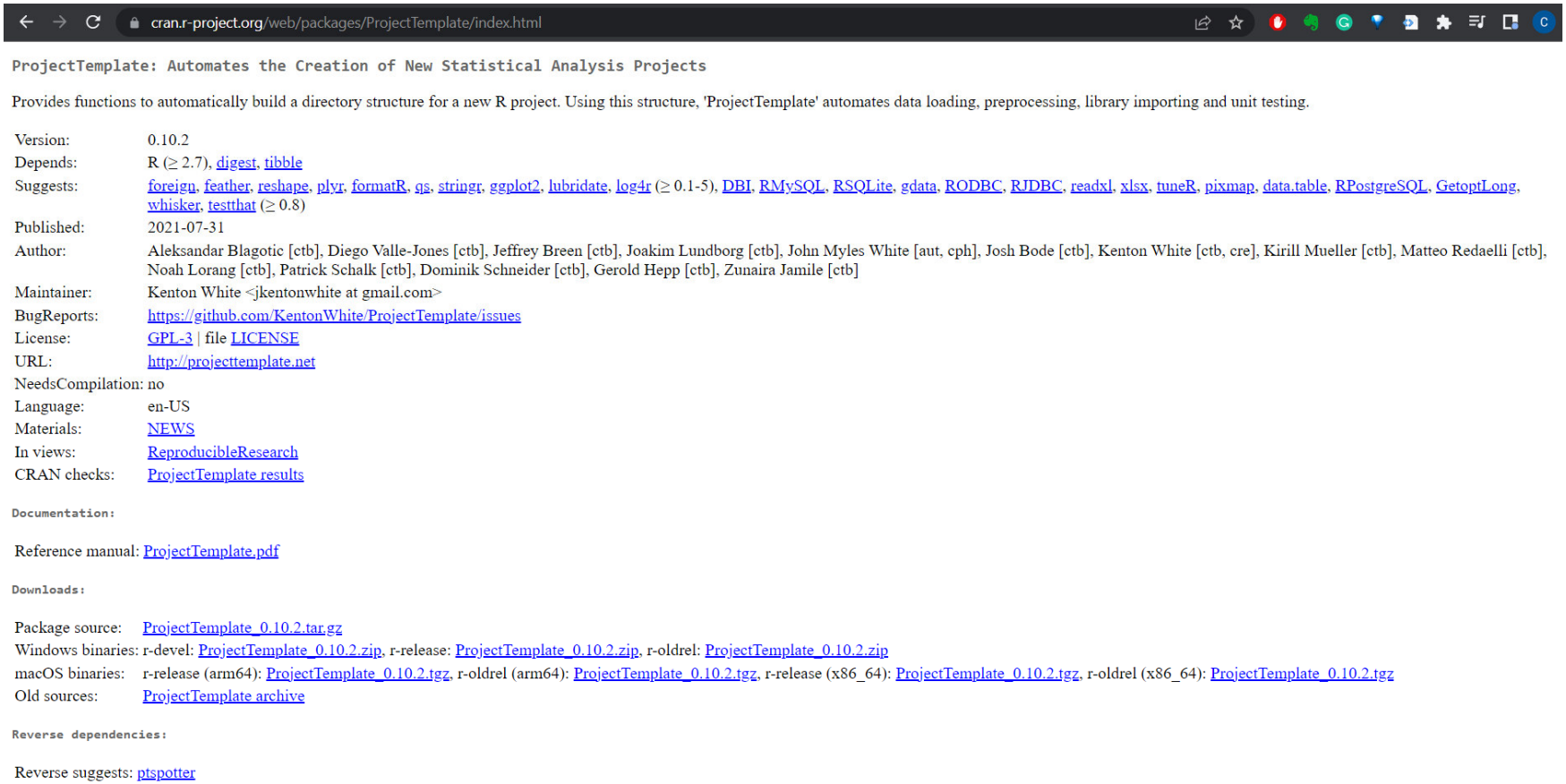
```
# install.packages("remotes")
remotes::install_github("rostools/prodigenr")
```

Usage

A very nice bookdown on reproducible science practices: https://r-cubed.rostools.org/r-project-management.html#r-project-management

Another package for subfolder structure in R projects

https://cran.r-project.org/web/packages/ProjectTemplate/index.html



cran.r-project.org/web/packages/ProjectTemplate/index.html

ProjectTemplate: Automates the Creation of New Statistical Analysis Projects

Provides functions to automatically build a directory structure for a new R project. Using this structure, 'ProjectTemplate' automates data loading, preprocessing, library importing and unit testing.

Version:	0.10.2
Depends:	R (≥ 2.7), digest, tibble
Suggests:	foreign, feather, reshape, plyr, formatR, gs, stringr, ggplot2, lubridate, log4r (≥ 0.1-5), DBI, RMySQL, RSQLite, gdata, RODBC, RJDBC, readxl, xlsx, tuneR, pixmap, data.table, RPostgreSQL, GetoptLong, whisker, testthat (≥ 0.8)
Published:	2021-07-31
Author:	Aleksandar Blagotic [ctb], Diego Valle-Jones [ctb], Jeffrey Breen [ctb], Joakim Lundborg [ctb], John Myles White [aut, cph], Josh Bode [ctb], Kenton White [ctb, cre], Kirill Mueller [ctb], Matteo Redaelli [ctb], Noah Lorang [ctb], Patrick Schalk [ctb], Dominik Schneider [ctb], Gerold Hepp [ctb], Zunaira Jamile [ctb]
Maintainer:	Kenton White <jkentonwhite at gmail.com>
BugReports:	https://github.com/KentonWhite/ProjectTemplate/issues
License:	GPL-3 | file LICENSE
URL:	http://projecttemplate.net
NeedsCompilation:	no
Language:	en-US
Materials:	NEWS
In views:	ReproducibleResearch
CRAN checks:	ProjectTemplate results

Documentation:

Reference manual: ProjectTemplate.pdf

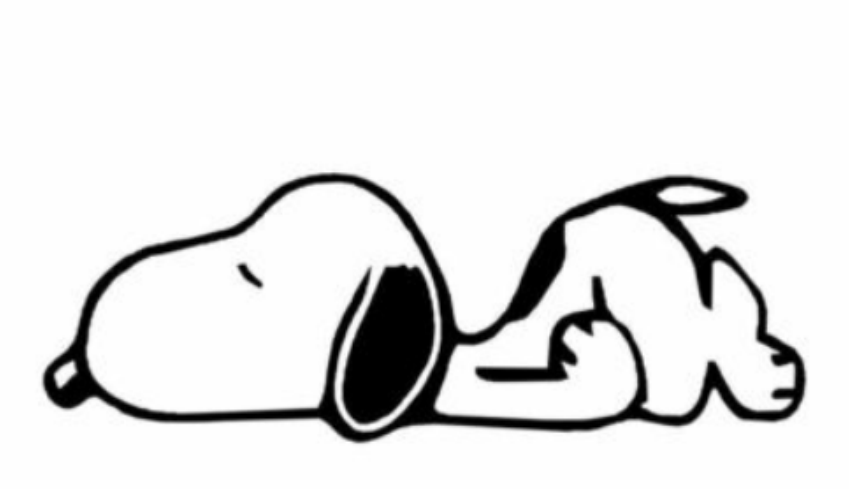
Downloads:

Package source:	ProjectTemplate_0.10.2.tar.gz
Windows binaries:	r-devel: ProjectTemplate_0.10.2.zip, r-release: ProjectTemplate_0.10.2.zip, r-oldrel: ProjectTemplate_0.10.2.zip
macOS binaries:	r-release (arm64): ProjectTemplate_0.10.2.tgz, r-oldrel (arm64): ProjectTemplate_0.10.2.tgz, r-release (x86_64): ProjectTemplate_0.10.2.tgz, r-oldrel (x86_64): ProjectTemplate_0.10.2.tgz
Old sources:	ProjectTemplate archive

Reverse dependencies:

Reverse suggests: ptspotter

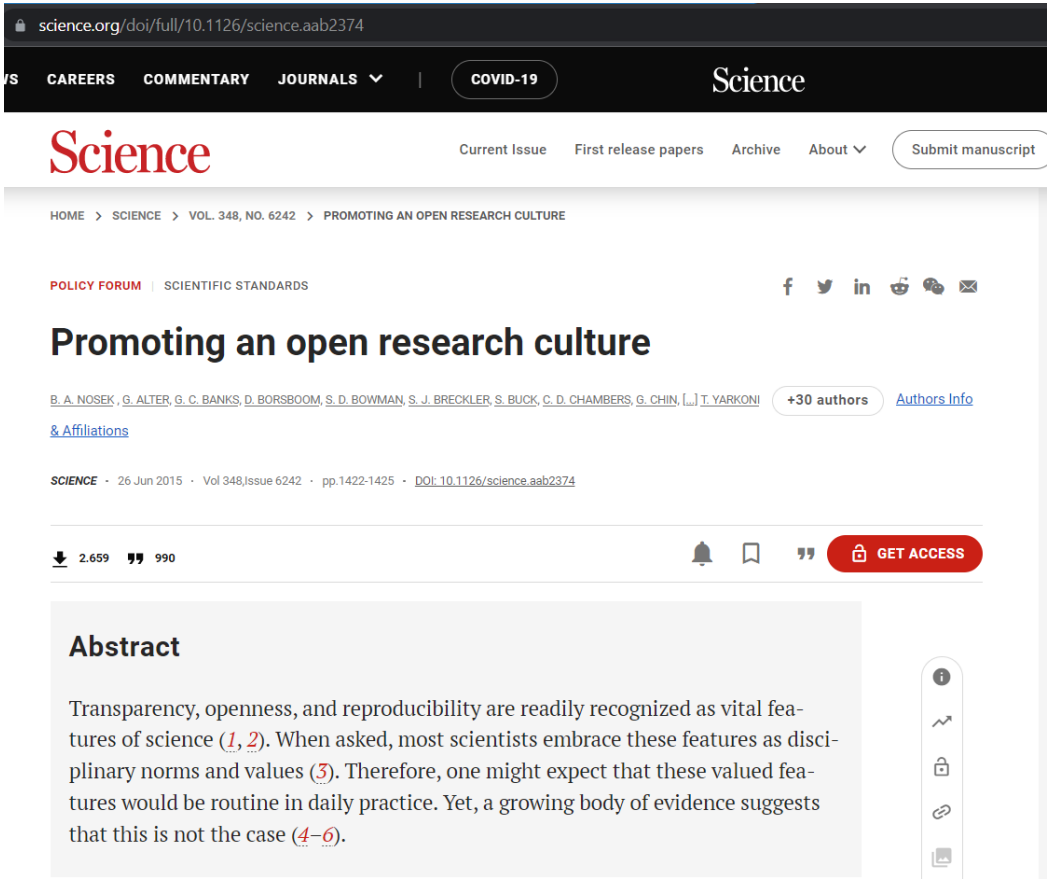
Long break



Science: Promoting an open research culture

Nosek B. A., Alter G., Banks G. C., Borsboom D., Bowman S. D., Breckler S. J., . . . Yarkoni T. (2015). Promoting an open research culture. *Science*, *348*(6242), 1422–1425. https://doi.org/10.1126/science.aab2374

- Transparency
- Openness
- Reproducibility of every single step
- Full documentation



Prof John Ioannidis: Research – increasing value, reducing waste

Published in final edited form as:
Lancet. 2014 January 11; 383(9912): 166–175. doi:10.1016/S0140-6736(13)62227-8.

Research: increasing value, reducing waste 2:

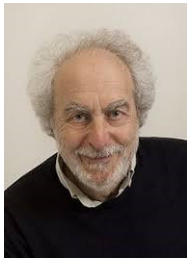
Increasing value and reducing waste in research design, conduct, and analysis

Prof. John P A Ioannidis, MD, **Prof. Sander Greenland, DrPH**, **Prof. Mark A Hlatky, MD**, **Muin J Khoury, MD**, **Prof. Malcolm R Macleod, PhD**, **Prof. David Moher, PhD**, **Prof. Kenneth F Schulz, PhD**, and **Prof. Robert Tibshirani, PhD**

Stanford Prevention Research Center (Prof J P A Ioannidis Medicine (Prof M A Hlatky MD), Department of Medicine, S Stanford, CA, USA; Division of Epidemiology, School of M CA, USA (Prof J P A Ioannidis); Division of Health Service Department of Health Research and Policy (Prof R Tibshir CA, USA; Department of Statistics, School of Humanities a Stanford, CA, USA (Prof J P A Ioannidis, Prof R Tibshirani Stanford (METRICS), Stanford University, Stanford, CA, U of Epidemiology and Department of Statistics, UCLA Scho USA (Prof S Greenland DrPH); Office of Public Health Ger and Prevention, Atlanta, GA, USA (M J Khoury MD); Epide Program, National Cancer Institute, Rockville, MD, USA (M Neurosciences, University of Edinburgh School of Medicin PhD); Clinical Epidemiology Program, Ottawa Hospital Re and Department of Epidemiology and Community Medicin University of Ottawa, Ottawa, ON, Canada; FHI 360, Durh and Department of Obstetrics and Gynecology, University Chapel Hill, NC, USA (Prof K F Schulz)

Abstract

Correctable weaknesses in the design, conduct, and analysis of biomedical and public health research studies can produce misleading results and waste valuable resources. Small effects can be difficult to distinguish from bias introduced by study design and analyses. An absence of detailed written protocols and poor documentation of research is common. Information obtained might not be useful or important, and statistical precision or power is often too low or used in a misleading way. Insufficient consideration might be given to both previous and continuing studies. Arbitrary choice of analyses and an overemphasis on random extremes might affect the reported findings. Several problems relate to the research workforce, including failure to involve experienced statisticians and methodologists, failure to train clinical researchers and laboratory scientists in research methods and design, and the involvement of stakeholders with conflicts of interest. Inadequate emphasis is placed on recording of research decisions and on reproducibility of research. Finally, reward systems incentivise quantity more than quality, and novelty more than reliability. We propose potential solutions for these problems, including improvements in protocols and documentation, consideration of evidence from studies in progress, standardisation of research efforts, optimisation and training of an experienced and non-conflicted scientific workforce, and reconsideration of scientific reward systems.



Douglas Altman
(† 2018)

John Ioannidis

Sander Greenland

Robert Tibshirani

David Moher

Transparency and Openness Promotion (TOP) guidelines

- 1) Standards for reporting/ Citation
- 2) to 4) data analytical methods (Code) and transparency regarding materials / methods
- 5) Transparency in design and analysis
- 6) Preregistration of studies
- 7) Preregistrations of the statistical analysis plan
- 8) Replication

TOP Guidelines: 1) Standards for reporting

- *Summary:* Citation of articles is routine and well-formulated. **Similar standards can be applied to citation of data, code, and materials to recognize and credit these as original intellectual contributions**. Level 1 recommends citation standards, Level 2 requires adherence to citation standards, and Level 3 requires and enforces adherence to citation standards.

- All **data**, **program code** and **other methods** should be appropriately cited. Such materials should be recognized as ***original intellectual contributions and afforded recognition through citation***.
- All **data sets** and **program code** used in a publication should be **cited in the text and listed in the reference section**.
- References for data sets and program code should include a persistent identifier, such as a **Digital Object Identifier** (DOI). Persistent identifiers ensure future access to unique published digital objects, such as a text or data set. Persistent identifiers are assigned to data sets by digital archives, such as **institutional repositories** and partners in the Data Preservation Alliance for the Social Sciences (Data-PASS).
- *Data set citation examples:*
  - Campbell, Angus, and Kahn, Robert L. ANES 1948 Time Series Study. Ann Arbor, MI: Inter-university Consortium for Political and Social Research [distributor], 2015-11-10. https://doi.org/10.3886/ICPSR07218.v4
  - Kidwell, M., Lazarevic, L. B., Baranski, E., Hardwicke, T. E., Piechowski, S., Falkenberg, L.-S., … Nosek, B. A. (2016, August 18). Badges to Acknowledge Open Practices: A Simple, Low Cost, Effective Method for Increasing Transparency. Retrieved from osf.io/rfgdw

TOP Guidelines: 2) to 4) Data analytic transparency

- Summary: Transparency guidelines for data, analytic methods, and research materials are conceptually distinct.
- To publish papers **only if the data, methods used in the analysis, and materials used to conduct the research are clearly and precisely documented and are maximally available to any researcher for purposes of reproducing the results or replicating the procedure**.

Authors using original data must:

a) make the **data available** at a **trusted digital repository** (Note: If all data required to reproduce the reported analyses appears in the article text, tables, and figures then it does not also need to be posted to a repository.)

b) include **all variables, treatment conditions, and observations** described in the manuscript

c) provide **a full account of the procedures used to collect, preprocess, clean, or generate the data**

d) provide **program code, scripts, codebooks, and other documentation** sufficient to precisely reproduce all published results

e) provide **research materials** and **description** of procedures necessary to conduct an **independent replication** of the research

- All should be made available through a trusted digital repository

TOP Giudelines: 5) und 6)

5) Transparency in design and analysis

- Using appropriate **standards/guidelines** for
  - **Reporting** a specific study design (medicine/health sciences: EQUATOR network)
  - **Analysing** a specific study design (i.e., PROGRESS guidelines for statistical modelling of prognostic/diagnostic models (Steyerberg et al., 2013), guidelines for treating missing values in clinical studies (Little et al., 2012))

Guidelines for reporting a specific design are distinct from statistical analysis guidelines!!!

6) Preregistration of studies

- Ideally, every study (no matter the design) should be registered. Helps with the file-drawer problem in science

TOP Guidelines: 7) Registration of the SAP

- To publish papers where authors indicate whether or not the conducted research was preregistered with an analysis plan in an independent, institutional registry (e.g., http://clinicaltrials.gov/, http://socialscienceregistry.org/, http://osf.io/, http://egap.org/design-registration/, http://ridie.3ieimpact.org/).
- Preregistration of studies involves registering the study design, variables, and treatment conditions. **Including an analysis plan involves specification of sequence of analyses or the statistical model that will be reported.**

1. Authors must, in acknowledgments or the first footnote, indicate if they did or did not preregister the research with or without an analysis plan in an independent, institutional registry.
2. If an author did preregister the research with an analysis plan, the author must:

   a. confirm in the text that the study was registered prior to conducting the research **with links to the time-stamped preregistrations at the institutional registry**, and that the preregistration adheres to the disclosure requirements of the institutional registry or those required for the preregistered badge with analysis plans maintained by the Center for Open Science.

   b. **report all pre-registered analyses in the text, or, if there were changes in the analysis plan following preregistration, those changes must be disclosed with explanation for the changes**.

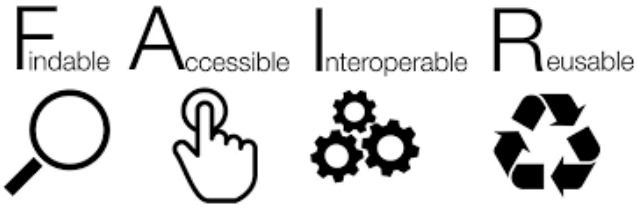
   c. **Clearly distinguish in text analyses that were preregistered from those that were not**, such as having separate sections in the results for confirmatory and exploratory analyses.

TOP Guidelines: 8) Reproducibility/replication

- *Summary: The transparency standards above account for reproducibility of the reported results based on the originating data, and for sharing sufficient information to conduct an independent replication. While not formally a transparency standard for authors, this section addresses journal guidelines for consideration of independent replications for publication.*

- Springer journals (includes BMJ journals via Springer Nature) adopt the TOP guidelines and FAIR principles
- Many national (and international) third-party funding bodies make the TOP and FAIR guidelines/principles mandatory for submission. Data Management Plans and Registration of Studies are mandatory for the BMBF, DFG and other large funders.
- I am associate editor of 2 journals and peer reviewer for journals and funding bodies. I reject 60-70% of submissions due to non conformance to the standards.

FAIR Principles



FAIR Principles | Compliance

Findability

Resource and its metadata are easy to find by both, humans and computer systems. Basic machine readable descriptive metadata allows the discovery of interesting data sets and services.

- **F1**. Resource is uploaded to a public repository.
- **F2**. Metadata are assigned a globally unique and persistent identifier.

Accessibility

Resource and metadata are stored for the long term such that they can be easily accessed and downloaded or locally used by humans and ideally also machines using standard communication protocols.

- **A1**. Resource is accessible for download or manipulation by humans and is ideally also machine readable.
- **A2**. Publications and data repositories have contingency plans to assure that metadata remain accessible, even when the resource or the repository are no longer available.

Interoperability

Metadata should be ready to be exchanged, interpreted and combined in a (semi)automated way with other data sets by humans as well as computer systems.

- **I1**. Resource is uploaded to a repository that is interoperable with other platforms.
- **I2**. Repository meta- data schema maps to or implements the CG Core metadata schema.
- **I3**. Metadata use standard vocabularies and/or ontologies.

Reusability

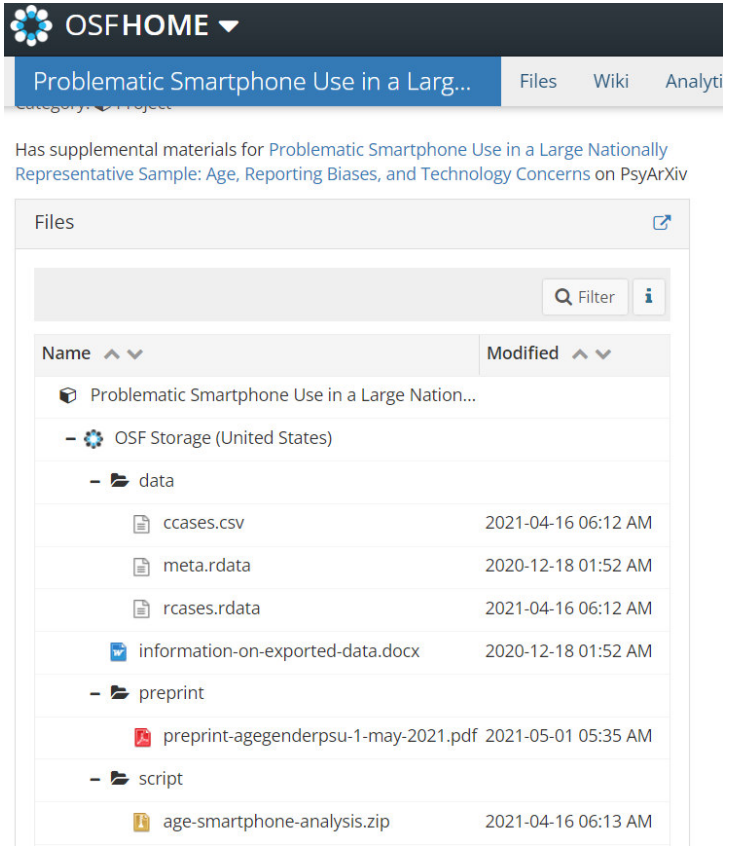
Data and metadata are sufficiently well-described to allow data to be reused in future research, allowing for integration with other compatible data sources. Proper citation must be facilitated, and the conditions under which the data can be used should be clear to machines

- **R1.** Metadata are released with a clear and accessible usage license.
- **R2.** Metadata about data and datasets are richly described with a plurality of accurate and relevant attributes.

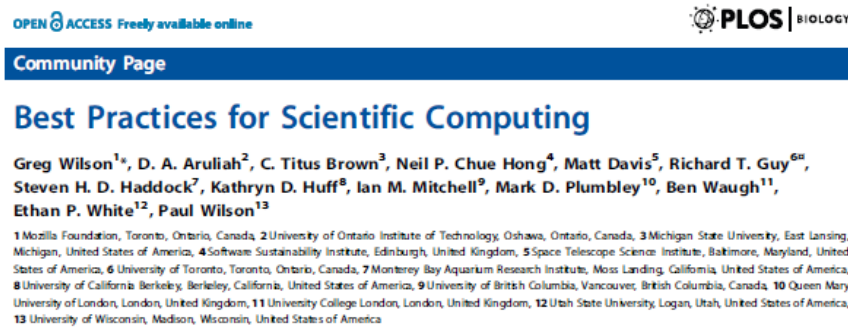
Wilkinson, M., Dumontier, M., Aalbersberg, I. *et al.* (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data, 3,* 160018. https://doi.org/10.1038/sdata.2016.18

How can we use R to help with TOP and FAIR?

- Git integration of R makes documentation of the full analysis possible
  - Version control helps with reproducibility anyway.
- R folder structure translates nicely into other repositories, i.e., the Open Science Framework
  - https://osf.io/
- R offers procedures that support saving the datasets as a minimal reproducible dataset or as a matrix structure that supports reproducibility even without sharing full data
  - For a var-cov matrix with means + sds for a dataset that could not be shared in full, please see here:
  - https://osf.io/ua75m/



Best/good enough practices in scientific computing



OPEN ACCESS Freely available online

PLOS | BIOLOGY

Community Page

Best Practices for Scientific Computing

Greg Wilson[1]*, D. A. Aruliah[2], C. Titus Brown[3], Neil P. Chue Hong[4], Matt Davis[5], Richard T. Guy[6¤], Steven H. D. Haddock[7], Kathryn D. Huff[8], Ian M. Mitchell[9], Mark D. Plumbley[10], Ben Waugh[11], Ethan P. White[12], Paul Wilson[13]

1 Mozilla Foundation, Toronto, Ontario, Canada, 2 University of Ontario Institute of Technology, Oshawa, Ontario, Canada, 3 Michigan State University, East Lansing, Michigan, United States of America, 4 Software Sustainability Institute, Edinburgh, United Kingdom, 5 Space Telescope Science Institute, Baltimore, Maryland, United States of America, 6 University of Toronto, Toronto, Ontario, Canada, 7 Monterey Bay Aquarium Research Institute, Moss Landing, California, United States of America, 8 University of California Berkeley, Berkeley, California, United States of America, 9 University of British Columbia, Vancouver, British Columbia, Canada, 10 Queen Mary University of London, London, United Kingdom, 11 University College London, London, United Kingdom, 12 Utah State University, Logan, Utah, United States of America, 13 University of Wisconsin, Madison, Wisconsin, United States of America

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. *PLoS computational biology*, *13*(6), e1005510. https://doi.org/10.1371/journal.pcbi.1005510

Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., Guy, R. T., Haddock, S. H., Huff, K. D., Mitchell, I. M., Plumbley, M. D., Waugh, B., White, E. P., & Wilson, P. (2014). Best practices for scientific computing. *PLoS biology*, *12*(1), e1001745. https://doi.org/10.1371/journal.pbio.1001745

Good Enough Practices in Scientific Computing

Greg Wilson[1,‡*], Jennifer Bryan[2,‡], Karen Cranston[3,‡], Justin Kitzes[4,‡], Lex Nederbragt[5,‡], Tracy K. Teal[6,‡]

1 Software Carpentry Foundation / gvwilson@software-carpentry.org
2 University of British Columbia / jenny@stat.ubc.ca
3 Duke University / karen.cranston@duke.edu
4 University of California, Berkeley / jkitzes@berkeley.edu
5 University of Oslo / lex.nederbragt@ibv.uio.no
6 Data Carpentry / tkteal@datacarpentry.org

‡ These authors contributed equally to this work.
* E-mail: Corresponding gvwilson@software-carpentry.org

Abstract

We present a set of computing tools and techniques that every researcher can and should adopt. These recommendations synthesize inspiration from our own work, from the experiences of the thousands of people who have taken part in Software Carpentry and Data Carpentry workshops over the past six years, and from a variety of other guides. Our recommendations are aimed specifically at people who are new to research computing.

Roadmap for the rest of today/tomorrow morning

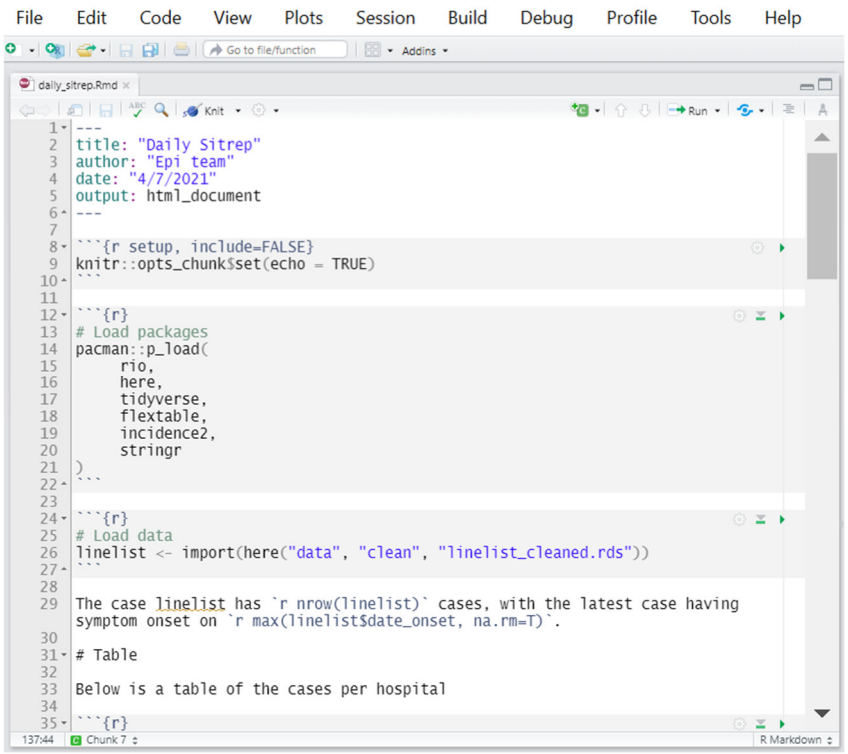
R processes and R workflow largely fall into two categories:

- **The wider architecture:**
  - Using an R project file
  - Where does the R project file sit in your overall folder structure?
  - How do you integrate your analysis files into your larger project files?
  - How do you organise your R project file folder?
  - How do you build in version control?
  - How do you integrate electronic systems with paper-based documentation?
  - How do you document what you do?
  - How do you achieve compatibility with a data repos/documentation system?

- **The focused architecture:**
  - How do I structure my R scripts so that I can find stuff?
  - What is the right flow of steps and functions within my R script file? How do I implement data cleaning pipelines/protocols and analysis protocols/guidelines?
  - How do I write code for data munging and data analysis that is durable?
  - How can I prevent workspace and package name conflicts? (Known as masking)
  - How can I write code that is self-contained and does not require for the whole script to run for one calculation to happen?
  - How can I write scripts that do not break when opened after a few months?

When do script files break?

- Hard coding versus soft coding
  - Hard coding: using an absolute file path (setwd()) vs soft coding: using a relative file path (via package here::here() function)
  - Hard coding: using subsetting with absolute row/cols positions vs soft coding: using relative row/cols positions via logical subsetting or c[, "var1"] name indexing
  - Hard coding: putting n = 28 into the .Rmd file versus soft coding: putting „Number of cases is 'r nrow(linelist)'.
  - Hard coding: Giving colour names as hex code values in ggplot versus soft coding: putting colour names as vectors, then using logical subsetting to detect number of colour names in ggplot2
- Sometimes, hard coding is perfectly alright. The moment you want your scripts to be durable, a certain amount of soft-coding is necessary. You decide how elaborate you need to be.



When do script files break?

- **Workspace problems:**
  - Managing the workspace in R is really difficult.
  - Later loaded packages may create masking conflicts by masking functions with the same names from earlier loaded packages.
  - Some packages should not be loaded at the same time because it may make one package to malfunction (unfortunately often the case with gtsummary and other packages)
  - Depending on the version of R and RStudio and package updates, old scripts break over time (i.e., reshape2 becoming obsolete with tidyr package).
  - Cluttered workspaces with many objects (ggplots/flextable objects) make rendering adventurous.
  - Windows is very good in sending updates that break Javascript and rtools, therefore making it impossible to run certain packages.
- **Solutions:**
  - Elaborate – setting up specific workspaces for projects and saving workspace setup files
  - Simple – keeping old R and RStudio installations for working with specific R script files
  - Relatively simple but laborious – before saving the final iteration of a script, put a .packages vector with lapply into the setup header, use lapply(), use package::function() for local package needs

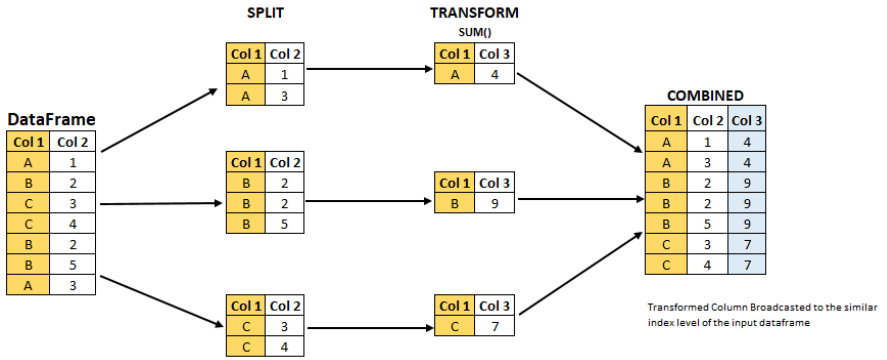
When do script files break?

- **Having no roadmap beforehand / forgetting to instigate a structure from the start**
  - Happens when iteratively finding your way through the dark with just a flashlight
  - Symptoms are: not following a style guide, not commenting, forgetting to use section headers, not being able to build the structural outer layer of an analysis based on a statistical analysis plan (SAP) beforehand, building scripts in a non-linear way which necessitates going back three lines to run the next line of code, always using the full dataset, only setting up factor variables on the go….
- Remedies:
  - Structure your analysis based on the aims/objectives/hypotheses beforehand. When starting your R script, put the structure in before you do anything else
  - Be consistent with naming stuff
  - Comment like you would comment if somebody else had to do your analysis
  - Always build script files so that they can be run line-by-line. Never write a script that contains for a step to be executed to go back three lines.
  - Try to use the split – apply – combine method in everything. Build scripts such that each step uses self-contained data files that are subfiles from the masterfile. If you have to change the coding of a variable (factor), do it in just this self-contained area.

When do script files break?

- **Doing stuff to the data files that overwrites columns directly.**
  - Most commonly this is a symptom of not having a master file.
  - Most commonly this problem is tied to overwriting the class of a variable, i.e., overwriting numeric variables as factor variables.

- **Remedies:**
  - Always treat the master file as “read-only”. Never ever change it.
  - Set up data transformations of any kind such that you create new variables instead of overwriting ol ones. This saves time later (for certain graphs it may be advisable to do transformation to factor only for the graph, not within the data file).
  - In general, use the split-combine-apply paradigm of programming in R:



When do script files break?

- **We are human. We forget. We also work with others that do not understand how data analysis is structured.**
  - Many of the symptoms of not optimal coding practices come from being rushed. In science, a lot of energy is spent on getting stuff ready on a deadline.
  - Building and maintaining code scripts requires time for carefully scripting and documenting the process so that you know when you come back to it.
  - 90% of my analysis time is spent on maintenance/documentation, 10% on actual analysis.

Remedies:

- See building and documenting your analysis as integral steps. They are allowed to take time.
- Explain about the quality processes involved in data cleaning/munging and analysis. Try to normalise these procedures towards superiors.
- Link into the Open Science Community. We can help.
- If you are really rushed, try to use the dictate function on your mobile to note your thoughts. Make time to document after getting the paper finished and for tidying up mandatory and non-negotiable.
- Consider delegating subprocesses (i.e., certain cleaning procedures) to students under careful supervision. Data teams in industry work such that everybody has tightly assigned roles.

Data munging in-depth

- General process : Looking at a data munging script → Over to R.

- Then: https://www.epirhandbook.com/en/index.html
  - Chapters 7 Import, export
  - Chapter 8 Cleaning data and core functions (with option to go into chapters 9-16 as necessary)
  - See .Rmd R Notebook or html files for following along

  - After material: Exercises (either provided dataset or own dataset)