Good Statistical Practice (GSP): guidence for using R for scientific publication

Caroline X. Gao [1,2,3], Matthew Hamilton [3], Sue Cotten [1,3]

a

Contents

Preface

This short practice guidance is designed based on a few guidelines and practice principles, books and jounal articles, including:

- ASA "Ethical Guidelines for Statistical Practice"
- Reproducible Research with R and R Studio
- Efficient R programming

The aim of this guidance is to promote accountability, reproducibility and integrity of statistical practice in Orygen health service and outcome research team. The guidance is divided into four parts: 1) planning for analysis, 2) data cleaning, 3) analysis, and 4) reporting.

Important note: **the authors do not give permission for you to print this resource on papers, unless you are experimenting magical ink that disappears after 3 months (you are considered to have net-zero carbon emissions in this case).** If you are thinking about reading this on paper before going to sleep, you are too opportunistic for your study plan : P

Have fun ~~

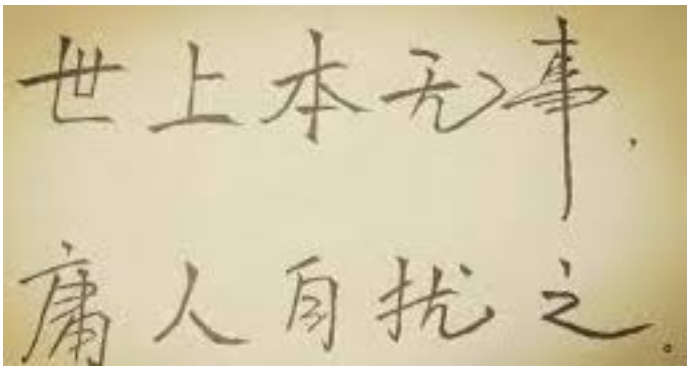
Planning for anlaysis

Dafting the anlaysis plan

The analysis will need to be planed before you touch the data. Your analysis may deviate from the analysis plan to some degrees, which may relate to data integrity of the variable selected, model fitting factors (i.e. multicollinearity, heteroscedasticity etc) and change of research questions. However, unless your aim is purely exploratory (i.e. identify latent clusters) or predictive, your analysis should be guided by the analysis plan to prevent "fishing" results. Remember "If you torture the data long enough, it will confess to anything - Ronald H. Coase", which is really against the scientific principle. The scientific evidence is based on a collection of studies and findings rather than a single paper. If you have a negative finding, you are obligated to publish it !!!

Less is more

Be aware of the "Complexity Bias". "Life is really simple, but we insist on making it complicated".



The famous quote is not by Confucius (it is from the New Book of Tang published about 1500 years after his death), and it is not well translated, but you get the idea. We often find it easier to face a complex problem than a simple one, and often feel that a complex solution must be better than an easier one. However this is often not true in applied statistics.

More often than not, ingenious statistical designs are surprisingly simple. An famous example is the Cox model for survival analysis, which simplifies the needs to describe the baseline hazard function with proportional hazard assumption. Another example, Dijkstra's algorithm (algorithm for finding the shortest paths between nodes in a graph), the author, Edsger Dijkstra, once said "One of the reasons that it is so nice was that I designed it without pencil and paper. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities."

This is the same with your analysis, always force yourself to avoid complexities if you could achieve what you need with a simpler model. There are numerous benefits for simpler models, i.e. less likely to over-fitting with your model, easier to communicate with your audience, less likely to make mistakes etc. If a logistic regression works, there is no need to use a structure equation model.

Officially certified plans

Get your analysis plan approved or reviewed. I think we all do this to some degrees. Some studies have strict protocols on who and when the analysis plan will need to be approved together with other ethical requirements. Other studies will only require you to discuss with your supervisors. Regardless, it will be better to have a written analysis plan with review and/or approval and avoids confusions down the track. Sometimes you might want or need to preregister your study, which is considered as a part of the open science practice.

Use R

Why R?

The essential skill set in the modern analytical community is the ability to use one or more script languages, SPSS doesn't count here. It's often critical for the success of your publications. More often then not, you might face challenges from the reviewer to do something that SPSS is not capable of. If you have already started your analysis in SPSS, please abandoned it ASAP. If R is in its early adolescent years, SPSS is a toddler and it suffers from the Peter Pan Syndrome (it will never grow up).

Source: unknown

Sorry for being a bit offensive, but the reality is the industry and scientific community are gradually moving away from SPSS to other languages for many reasons, cost, capacity, flexibility, reporducibility etc. So to avoid the long term pain, change to R (Stata is also good, since this is a practice guide for R, I won't touch too much on Stata).

Learn to use R

"R being hard to learn" mostly exists in our fears. Once you get on to it the challenges are mostly "feeling a bit confused", there are so many packages, so many ways to do the same thing, so much typing needed, so many materials everywhere… So the best way to learn R is to learn by using it. You can start with a short intensive introduction course (one or two days) to build

up your confidence. Start using it in a small project, and then gradually roll out to more parts of your analysis tasks. You can also start by learning your collaborator's code (communicate with the author and create a separate environment so that you won't break anything).

There are lots of good online training materials for R:

- R for Reproducible Scientific Analysis from Software Carpentry
- R Programming on Coursera by Roger Peng,
- An Introduction to R
- R for Data Science.

Almost all R users are still learning R!! So you have no excuse not to ~~

R setup and project managment

Setup R on your computer

One thing that you have to remember: R is a fast evolving language. It has a new version every few months (sometimes two versions in one months), with funny names :)

```
library(rversions)
tail(r_versions())
```

```
##     version                date                 nickname
## 116   3.6.3 2020-02-29 08:05:16    Holding the Windsock
## 117   4.0.0 2020-04-24 07:05:34                Arbor Day
## 118   4.0.1 2020-06-06 07:05:16          See Things Now
## 119   4.0.2 2020-06-22 07:05:19        Taking Off Again
## 120   4.0.3 2020-10-10 07:05:24 Bunny-Wunnies Freak Out
## 121   4.0.4 2021-02-15 08:05:13       Lost Library Book
```

Although R kept on updating, you do not need to re-install R all the time. But I tend to update my R half-yearly. It doesn't take a long time to update everything now a days. The first thing that you need to do after installing R is to install your commonly used packages. A good way to install + load packages is to use the *pacman* package, which is much faster than typing install.package() and library(). I normally store the names of my commonly used packages somewhere. So when I need to re-install R, I will call *pacman* to install all of those packages for me at once (only takes about 10 minutes).

```
# load libraries
library(pacman)
p_load("dplyr", "tidyr", "ggplot2")
```

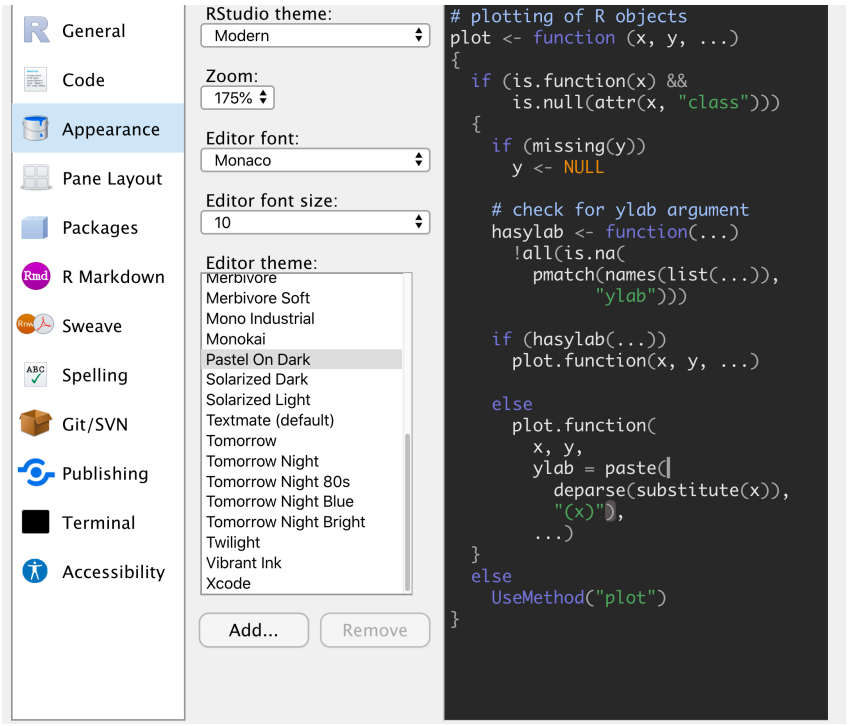
install.package() and *pacman* install packages from R CRAN. However, the authors may have additional update on Github which is not loaded to CRAN. You can install the most resent package from Github using *devtools*.

```
library(devtools)
install_github("rstudio/bookdown")
```

To get start with things, just install packages from R CRAN. If you come across with a problem which seems to be a software "bug", you can update your package using *install_github*. You can also check whether your issue was being reported on Github (i.e. https://github.com/rstudio/bookdown/issues).

Customizing RStudio

The modern use of R is almost always via RStudio. After installing the RStudio, you might want to customize it based on your own preferences. See the guide here. I like the "Pastel On Dark" editor theme (green and blue dominated dark theme).



I also like to turn on a few diagnostics including "Check usage of <- in function call" and "Provide R style diagnostics" (make sure you have a good R coding style, see style guide here

and a tidyverse style guide here). There is no need to strictly follow these guidelines, but you need to be aware of what is good, acceptable or bad.

Keep tracking of R and package versions

For all your analysis, you need to keep track of the R environment, this includes R version, platform and all package versions that were loaded in your current analysis environment. R environment information can be easily summarised using *sessionInfo*.

```
sessionInfo()
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Catalina 10.15.7
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_AU.UTF-8/en_AU.UTF-8/en_AU.UTF-8/C/en_AU.UTF-8/en_AU.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] rversions_2.0.2  kableExtra_1.3.1 knitr_1.30       forcats_0.5.0
##  [5] stringr_1.4.0    dplyr_1.0.2      purrr_0.3.4      readr_1.4.0
##  [9] tidyr_1.1.2      tibble_3.0.4     ggplot2_3.3.2    tidyverse_1.3.0
## [13] pacman_0.5.1
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.1.0  xfun_0.20         haven_2.3.1       colorspace_2.0-0
##  [5] vctrs_0.3.5       generics_0.1.0    htmltools_0.5.1.1 viridisLite_0.3.0
##  [9] yaml_2.2.1        rlang_0.4.10      pillar_1.4.7      glue_1.4.2
## [13] withr_2.3.0       DBI_1.1.0         dbplyr_2.0.0      modelr_0.1.8
```

```
## [17] readxl_1.3.1      jpeg_0.1-8.1      lifecycle_0.2.0   munsell_0.5.0
## [21] gtable_0.3.0      cellranger_1.1.0  rvest_0.3.6       memoise_1.1.0
## [25] evaluate_0.14     curl_4.3          fansi_0.4.1       broom_0.7.2
## [29] Rcpp_1.0.5        formatR_1.7       scales_1.1.1      backports_1.2.1
## [33] webshot_0.5.2     jsonlite_1.7.2    fs_1.5.0          conflicted_1.0.4
## [37] hms_0.5.3         digest_0.6.27     stringi_1.5.3     bookdown_0.21
## [41] rprojroot_2.0.2   grid_4.0.2        here_0.1          cli_2.2.0
## [45] tools_4.0.2       magrittr_2.0.1    crayon_1.3.4      pkgconfig_2.0.3
## [49] ellipsis_0.3.1    xml2_1.3.2        reprex_0.3.0      lubridate_1.7.9.2
## [53] assertthat_0.2.1  rmarkdown_2.6.6   httr_1.4.2        rstudioapi_0.13
## [57] R6_2.5.0          compiler_4.0.2
```

In fact, it might also be a good idea to include a R package version table in the appendix of your paper.

```
my_session_info <- sessionInfo()
# extract name, version and date
Version <- lapply(my_session_info$otherPkgs, function(x) x$Version)
Version <- unlist(lapply(Version, function(x) ifelse(identical(x,
    character(0)), " ", x)))
Date <- lapply(my_session_info$otherPkgs, function(x) as.character(as.Date(x$Date)))
Date <- unlist(lapply(Date, function(x) ifelse(identical(x,
    character(0)), " ", x)))

# extract session info in to data frame
info_table <- data.frame(Package = names(my_session_info$otherPkgs),
    Version = Version, Date = Date)

# keep in two columns as your list gets too
# long
if (nrow(info_table)%%2 == 0) {
    info_table <- cbind(info_table[c(TRUE, FALSE),
        ], info_table[c(FALSE, TRUE), ])
} else {
    info_table <- cbind(info_table[c(TRUE, FALSE),
        ], rbind(info_table[c(FALSE, TRUE), ],
```

```
        c("", "", "")))
}


# display table
rownames(info_table) <- NULL
knitr::kable(info_table, booktabs = TRUE, linesep = "") %>%
    kable_styling(bootstrap_options = "striped")
```

Package	Version	Date	Package	Version	Date
rversions	2.0.2	2020-05-25	kableExtra	1.3.1	2020-10-22
knitr	1.30	2020-09-22	forcats	0.5.0	2020-03-01
stringr	1.4.0	2019-02-10	dplyr	1.0.2	2020-08-18
purrr	0.3.4	2020-04-17	readr	1.4.0	2020-10-05
tidyr	1.1.2	2020-08-27	tibble	3.0.4	2020-10-12
ggplot2	3.3.2	2020-06-19	tidyverse	1.3.0	2019-11-21
pacman	0.5.1	2019-03-11			

This is crucial as packages or R updates may make your code fail or your results differ. This sounds scary than it really is. All analysis packages will have this issue. It's slightly complected with R because updates of R and affiliated packages are not synchronized. For my 13 years of using R, I haven't found this particularly challenging for a few reasons. The packages update rates were not as fast as we are seeing today. Most of my projects were in isolated environments so I rarely need to re-run the analysis a few years later and will need to produce exactly the same results. If I need to re-run the analysis, I will mostly update the analysis code and make sure it adhere to the current best practice. Occasionally, some of my package updates will cause problems with my current analysis code. As I keep track of my R environment, I will be able to figure out the problem quickly and re-install the older version back.

```
require(devtools)
install_version("bookdown", version = "0.20",
    repos = "http://cran.us.r-project.org")
```

If you are in the area that requires 100% reproducibility all the time, you can use **Microsoft R open** together with *checkpoint* which lock down the set of packages in your project environment, so package updates will not impact your code. There are a few other alternatives, but you can get other benefits from **Microsoft R open** without too much trouble, so it seems to be a wise choice at the current stage. If you need to install lots of things from Github and work on package development, it may not be a good choice.

Project managment with R

Over the years of being a programmer and biostatistician, I have learned many things in a painful way. One of them is the importance of good project management with your data, analysis, documentation and drafts. I had to spent a long time to remember what I did, where I save things and also lots of detective work on whether anyone have touched my 'cheese' (with files stored on network drives). When I am lucky I can request IT to restore a backup at specific time to find the lost treasure. But in many cases, this doesn't work as either I was stupid enough not to save things on the network drive, or the network drive deletes backups automatically.

Project management became more critical when I had to change my work between computers, laptops and servers as well as swap frequently between Windows and Mac. Managing file versions, backups and file paths became additional burdens. However, all of these things became much more easier with RStudio + Github. It's also more user friendly with people who are not familiar with Git.

Essentially the idea is that you can create an independent project environment for isolated task or tasks that shares common data source. This can be a specific analysis for a paper or a range of analysis base on the same dataset. When there are multiple users, it's always the best to use separate project environment for individual analysis, so users do not load the same project file on the network drive.

The method for setting up projects are described here. There are many benefits using projects:

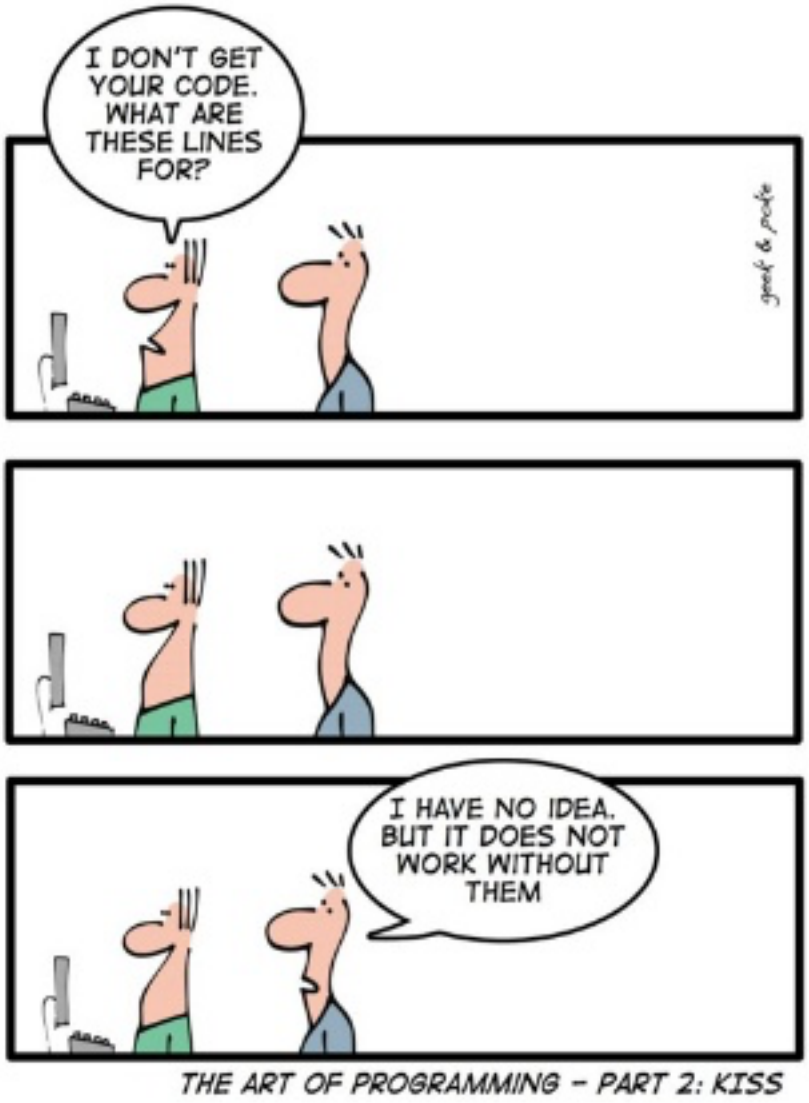
- No need to type the long path directory in the project environment because it is using relative file path.
- You can use here package when working with Rmarkdown files within the project environment. The default file path in Rmarkdown is the Rmarkdown file location, using here package, it always point to the project directory. So you have your path directory consistent when using console and Knit (see Rmarkdown in the next session).
- Access all files in the project environment easily
- Easy to communicate with Github

You can setup a rule for organising sub-folders and files. I like to separate code, data and results. However, it is slightly complicated to split analysis code and results when you are using Rmarkdown. Regardless, it would be better to have sub-folder structure in placed so you can find files easily.

State-of-art Rmarkdown/Rnotebook paractice

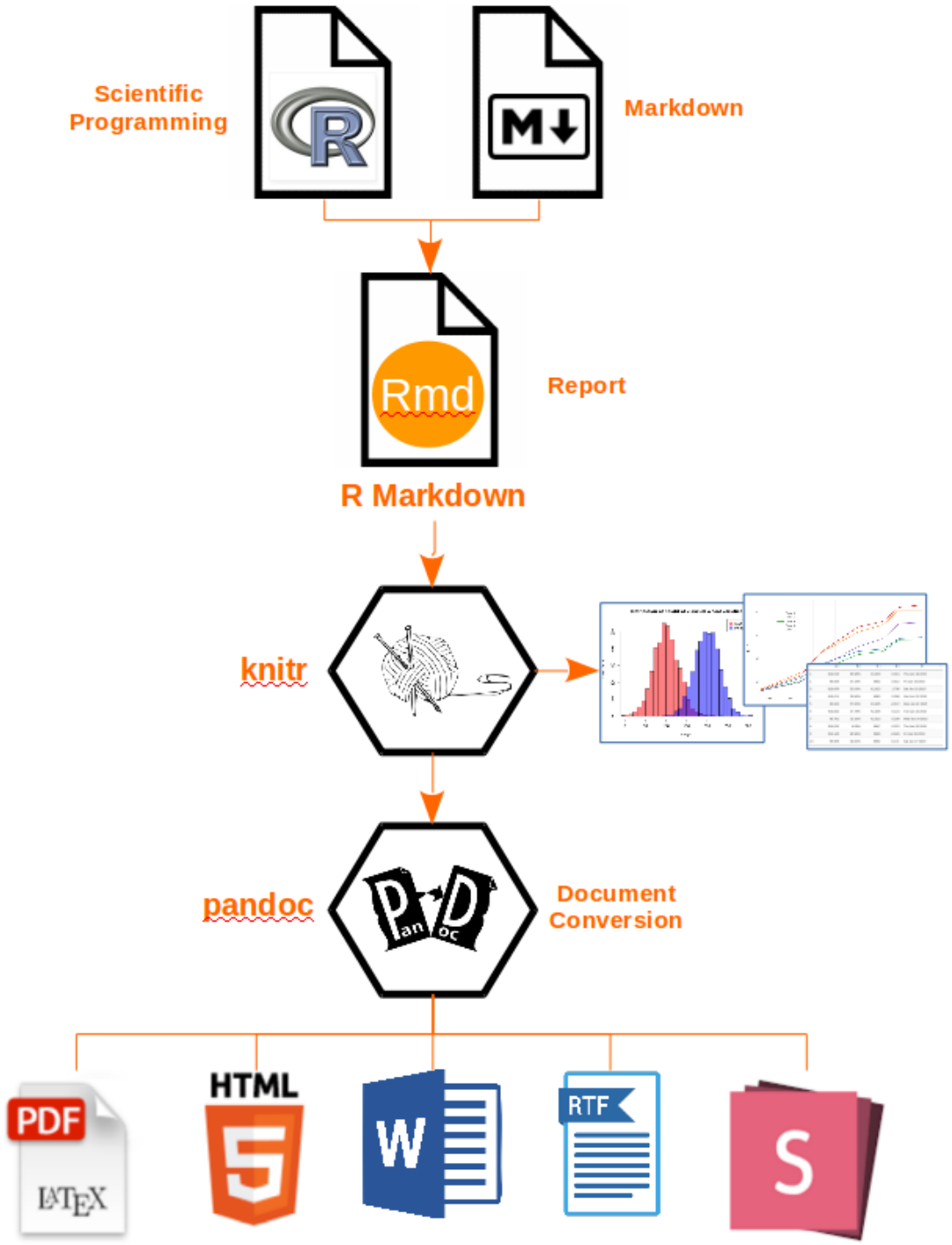
The idea of Markdown practice was originated from Donald Knuth. Donald believed that programmers should think of programs as works of literature and introduced a new programming paradigm called "Literate programming". He then developed TeX to facilitate this paradigm of coding, which is the typesetting system that LaTex is based. The idea behind it is simple, combining the text and code for human to read not computers. In the end, scientific discoveries are for humans to interpret, replicate and extend, and not for computers.

Although this is a fascinating idea, it had been hard to achieve for many reasons. Academic publications mostly only allow words, tables, figures but not computer code, so there is no obvious momentum from the academic communities. Software developers and computer science engineers mostly focused on the end product and not so much on displaying the process in between. Technically it is also difficult to achieve as this approach will need to be implemented within or closely connected to one or multiple programming environments.



Source: http://geek-and-poke.com/geekandpoke/2009/7/25/the-art-of-programming-part-2.html

However, the sudden boom of data science brought in new challenges. As software engineers move towards data science, the community gradually realised that the traditional coding style (i.e. modular programming, black-box approach) became barriers instead of advantages. Each step of the programming code, from cleaning variables to running algorithm, suddenly became all important. Hence there is an immediate need to communicate regarding every step of the process. This needs drove the birth of many interactive computational notebooks (IPython, SageMath, Beaker, Jupyter, Apache Zeppelin and Rmarkdown). R markdown became the most successful computational notebooks for R users. Why? Probably for the same reason as why R has been successful, simplicity, flexibility, open source etc.



Source: http://applied-r.com/project-reporting-template/

R markdown is a system that integrate narrative, analysis, code and output to create a production quality document. The code and text were implemented via R in a *.Rmd file, which will be knit to Markdown file and then convert to file type of choice via Pandoc. There are a few important features that makes it one of R's "killer" feature

- Reproducibility. You are no longer required to live in the copying and pasting world. All the analysis, results can be easily reproduced with changes in source data, cleaning routine and analyses. There are different degrees of implementing R markdown, from basic including notes for your data cleaning process to write a whole paper in one markdown file. Regardless of where you are at, using R markdown will significantly reduced you workload of having to find and copying and pasting results to the final paper. So you won't need to panic when your cohorts said one person is missing from the data set, and the 12 large tables in your paper will need to be updated.
- Easy progress tracking and debugging. Before the age of R markdown, R users will need to store all the code into a plain text file or R file and execute the code one by one to see the results. When the code file is getting very long, it gets harder to maintain the code and changes. Then coders will use split system to break long code into modules to make the progress tracking and debugging easier. However this will no longer be the case with R markdown, because you will see the code and results together and you can also tag the section of your code and use hyperlinks to quickly find things.
- Communication. You can integrate narrative, code and results together. This is quite crucial in the communication in data analysis, because it avoid all the efforts of reading segregated code comments, switching between documents to find results and maintaining separate documentations. All this process can be integrated into one system.
- Very nice for equations. If you need to type lots equations, then R markdown is also your friend. The standard LaTeX equations works with all types of outputs (when your equations are getting complicated, sometimes word doesn't work)
- Extension. R markdown system provides you will limitless extension capacities, such as integrating multiple languages (I can combine python, Stan and R in the same file now), producing multiple types of documents (word, html, pdf, slides, books, shiny app etc), displaying interactive plots

J.J. Allaire gave an excellent introduction presentation, Notebooks with R Markdown, in useR conference in 2016. I hope I have convinced you to move to R markdown.

Here are some practical point.

Learn to use R markdown

There are many good online introductions on using R markdown:

- R Markdown from R Studio
- R Markdown Basics by Andy Lin from the famous IDRE, UCLA stats resource website.
- R Markdown cheatsheet
- R Markdown Reference Guide
- R Markdown: The Definitive Guide by Yihui Xie
- bookdown: Authoring Books and Technical Documents with R Markdown (for advanced users) by Yihui Xie

YAML header

R Markdown files (*.Rmd) starts with YAML headers, which specify document parameters (or pandoc parameters), such as title, author, type of document, parameters etc.

```
---
title: "Good Statistical Practice (GSP)"
output: html_document
---
```

One thing you have to remember is that indentation has its meaning in YAML header. See the following example, having a table of content (toc), is a sub-option for the HTML document, hence is it needs to be indented.

```
---
title: "Good Statistical Practice (GSP)"
output:
  html_document:
    toc: TRUE
---
```

There is no comprehensive YAML guide exist. However, there is a nice package called ymlthis, which provides an addin for choosing YAML specifications. I normally use the bookdown output styles which makes, references tables and figures citation slightly easier for both pdf and word documents. The YAML header for this document looks like this:

```
---
title: "Good Statistical Practice (GSP)"
author:
- name: Caroline X. Gao [1,2], Matthew Hamilton [3]
output:
```

```
   bookdown::pdf_document2:
     toc: yes
     number_sections: false
     latex_engine: xelatex
     pandoc_args:
      - --template=template.tex
   bookdown::word_document2:
     toc: yes
     number_sections: false
     reference_docx: "template.docx"
geometry: margin=1in
fontsize: 11pt
bibliography: GSP.bib
tables: yes
header-includes:
   \usepackage{float}
   \floatplacement{figure}{H}
   \newcommand{\beginsupplement}{\setcounter{table}{0}  \renewcommand{\thetable}{S\arabic{ta
   \usepackage{lscape}
   \newcommand{\blandscape}{\begin{landscape}}
   \newcommand{\elandscape}{\end{landscape}}
---
```

You may notice that I have used template documents, "template.tex" and "template.docx". The LaTex template was inherited from Prof Rob hyndman's MonashEBSTemplates with minor modifications. "template.docx" is simply a Word document with defined fonts. "Header-includes" contains the additional latex command that I want to use in the documents, i.e. "lscape" package is for the landscape pdf page and "beginsupplement" is for setting supplementary table and figure captions (starts from Table S1 and Figure S1) .

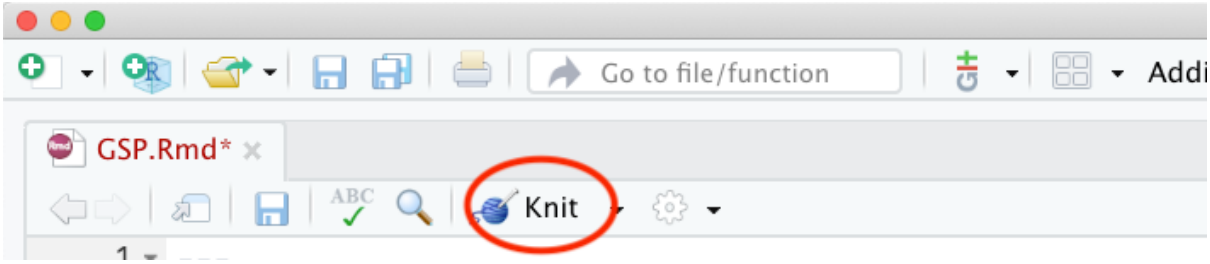
Using R markdown

R markdown is very easy to use. Code will need to be included as code chunk and everthing else outside of code chunks will be treated as formatted text, which includes, headings, lists, standard text and equations, see R Markdown cheatsheet for details. Code chunk include varies of definitions, such as code type (R, Python, Stan ...), chunk name, whether to display code,

figure size, table options etc. * R Markdown Reference Guide provides very detailed information about this.

Knit document

To create html, word or pdf file requires to Knit the Rmd document.



You also need to install LaTex (https://www.latex-project.org/get/). However, LaTex is really a large monster to work with. So Yihui Xie has developed TinyTeX for R users, which is much smaller. You can install additional LaTex packages easily if needed.

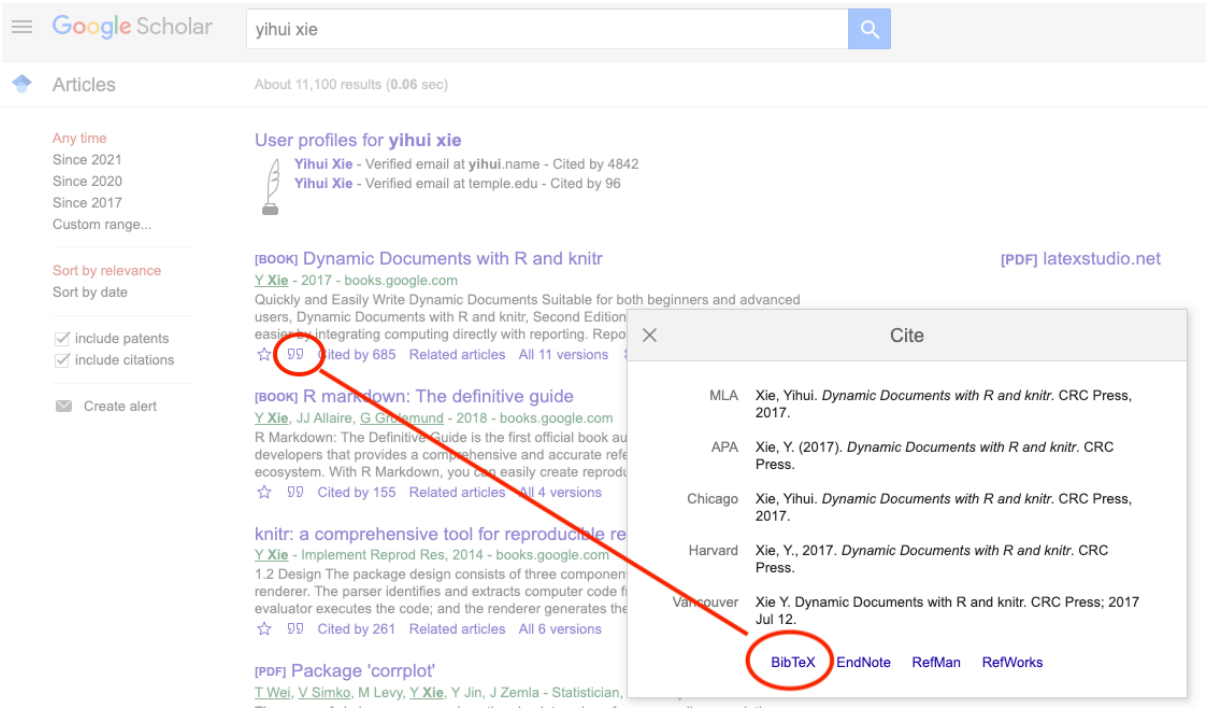
Citation in R markdown.

Another advantage of using R markdown is that you can finally give up using Endnote (the monster crashes your computer and word all the time). R markdown can work with the many citation management file types. The commonly used one is the LaTeX bibliography management system BibTeX, in which the references were stored as plain text like this:

```
@article{Xu_2020,
  title={Socioeconomic inequality in vulnerability to
    all-cause and cause-specific hospitalisation associated
    with temperature variability: a time-series study
    in 1814 Brazilian cities},
  author={Xu, Rongbin and Zhao, Qi and Coelho,
    Micheline SZS and Saldiva, Paulo HN and
    Abramson, Michael J and Li, Shanshan and Guo, Yuming},
  journal={The Lancet Planetary Health},
  volume={4},
  number={12},
  pages={e566--e576},
  year={2020},
```

```
  publisher={Elsevier}
}
```

Most journals provide direct download of the BibTex of articles. You can also export the Endnote library to a bib file. What I like to use is the Google Scholar BibTex Citation. However, it does not include doi of the paper.



When using the BibTeX, you need to first store the reference in a *.bib file and specify the name of the file in the YAML header. Then you can reference all your code in the text using @ followed by the id of the reference with either @Xu_2020 which generates author (year) or [@Xu_2020] which generates (author, year). By default Pandoc uses Chicago author-date CSL format for citations and references. You can specify the style according to the journal's requirements. Most of journals' csl styles can be found here. All features will be automatic including whether the numbers will be included before or after punctuation.

```
---
title: "Good Statistical Practice (GSP)"
output:
  html_document:
    toc: TRUE
bibliography: GSP.bib
csl: biomed-central.csl
---
```

Data pre-processing

A range of names were used to refer to the pre-processing stage of your data analysis: data cleaning, data cleansing, data wrangling, data mungling… This is a stage that you organize, validate, and prepare data for further analysis.

Importing data to R

R can import different types of source data (csv, excel, SPSS, Stata, SAS. SQL…). See the comprehensive guide here. There are three packages frequently used for processing common external data: foreign , haven, Hmisc. *foreign* includes most of the importing and exporting functions, however *spss.get* from *Hmisc* provides additional enhanced features including applying proper labels, compress data etc. *haven* is also easy to work with which allows you to use the numeric values instead of directly import as factor variables.

```
dta <- haven::read_sav(here::here("Data", "testdata.sav"))
dta
```

```
## # A tibble: 5 x 4
##   numeric                factor_numeric  factor_n_coded_miss date
##     <dbl>                     <dbl+lbl>            <dbl+lbl> <dttm>
## 1       1  1 [strongly disagree]         1 [strongly disagr~ 1983-12-11 00:00:00
## 2       2  2 [disagree]                  2 [disagree]        2018-07-01 00:00:00
## 3       3  3 [neither agree nor disagr~ NA                  2017-10-23 00:00:00
## 4      NA NA                             5 [strongly agree]  NA
## 5       3 NA                            NA                  NA
```

```
dta <- as_factor(dta)
dta
```

```
## # A tibble: 5 x 4
##   numeric factor_numeric            factor_n_coded_miss date
##     <dbl> <fct>                     <fct>               <dttm>
## 1       1 strongly disagree         strongly disagree   1983-12-11 00:00:00
## 2       2 disagree                  disagree            2018-07-01 00:00:00
## 3       3 neither agree nor disagree <NA>               2017-10-23 00:00:00
## 4      NA <NA>                      strongly agree      NA
## 5       3 <NA>                      <NA>                NA
```

```
dta <- foreign::read.spss(here::here("Data", "testdata.sav"),
    to.data.frame = TRUE)
```

```
## re-encoding from CP1252
```

```
head(dta)
```

```
##   numeric             factor_numeric factor_n_coded_miss        date
## 1       1          strongly disagree   strongly disagree 12659328000
## 2       2                   disagree            disagree 13749782400
## 3       3 neither agree nor disagree                <NA> 13728096000
## 4      NA                       <NA>      strongly agree          NA
## 5       3                       <NA>                <NA>          NA
```

```
dta <- Hmisc::spss.get(here::here("Data", "testdata.sav"),
    datevars = c("date"))
```

```
## re-encoding from CP1252
```

```
dta
```

```
##   numeric             factor.numeric factor.n.coded.miss       date
## 1       1          strongly disagree   strongly disagree 1983-12-11
## 2       2                   disagree            disagree 2018-07-01
## 3       3 neither agree nor disagree                <NA> 2017-10-23
## 4      NA                       <NA>      strongly agree       <NA>
## 5       3                       <NA>                <NA>       <NA>
```

Working with data.frame, tibble and data.table

Data frame is an easier understandable term, representing a flat data file with different columns having different variables with different format. However you might also hear tibbles. It was designed as a modern version of data frame, it allows better print visualization, flexible variable names, no row names etc. Normally it is better to use tibble, however sometimes it gets into trouble, then you need to change back to data frame.

```
as_tibble(dta)
```

```
## # A tibble: 5 x 4
##   numeric    factor.numeric             factor.n.coded.miss date
##   <labelled> <fct>                      <fct>               <date>
```

```
## 1  1         strongly disagree          strongly disagree   1983-12-11
## 2  2         disagree                   disagree            2018-07-01
## 3  3         neither agree nor disagree <NA>                2017-10-23
## 4 NA         <NA>                       strongly agree      NA
## 5  3         <NA>                       <NA>                NA
```

```
data.frame(dta)
```

```
##   numeric             factor.numeric factor.n.coded.miss       date
## 1       1          strongly disagree   strongly disagree 1983-12-11
## 2       2                   disagree            disagree 2018-07-01
## 3       3 neither agree nor disagree                <NA> 2017-10-23
## 4      NA                       <NA>      strongly agree       <NA>
## 5       3                       <NA>                <NA>       <NA>
```

If you are using large datasets, you can also use data.table. The coding style follows SQL. Although it takes some time to learn the coding style, it is generally over 10 times faster than operating with data.frame. I never had to move to data.table because most of my time consuming code/functions require data.frame as input : P

Name you variables properly!!!!!!!!

Every time when I saw variable names such as "X1", "V2", "12345", "WWF111", "GGG222", "Don't use", "?", "Gender", "Gender1", "Gender2", "Gender3", "Gsp11229371", "never_ending_name_but_do_not_know_what_it_means", or "theyusenamesthatjustruntogetherwithnocapitalsorunderscores", I would turn into Jeff Atwood.

Variable names have to be proper because they will make both you and your collaborator's work much easier. You will be able to find variables easily and communicate well with your code. So

when you import your data, make sure that all variables have proper names (actually variable names have to be proper in the database). There are a many common naming systems:

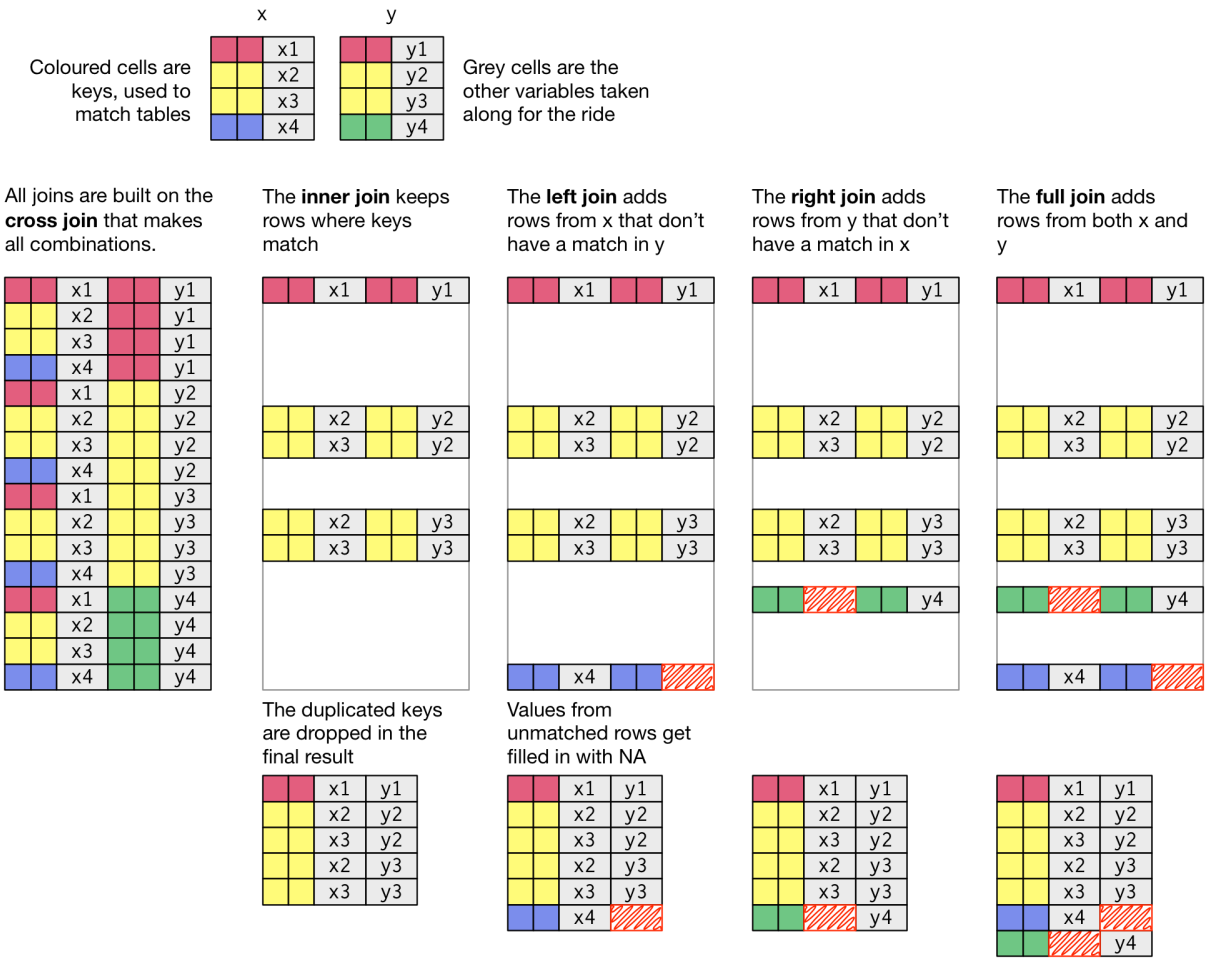
Analysis	Function and package
twoWords	lower camel uase
TwoWords	upper uamel uase
two_words	snake case
TWO_WORDS	screaming snake case
two_Words	camel snake case
Two_Words	pascal snake_case

Although snake case is mostly recommended, you can use any/all systems as long as you use them consistently. R allows you to use other special characters such as "." in variable names, but some other programming language may not allow you to do so (i.e. Stata do not allow "."). So stick with snake, camel or combined systems.

Generally, variable names should be nouns and function names should be verbs. You also need to name clusters of variables with similar yet specific names so they are easier to be referred to using *dplyr::starts_with()* or *dplyr::ends_with()*. For example, If you use "IESR_q1", "IESR_q2"…"IESR_q22" as IESR individual item names and IESR_total as IESR total scores. Then you can use *dplyr::starts_with("IESR_q")* to select individual items without the total items. If you name them as "IESR1", "IESR2"…"IESR22", "IESRTotal", *dplyr::starts_with("IESR")* will select all of them.

Data orgnizing

In this stage, you will need to combine data from different sources to one or more datasets. In many cases, raw data were extracted from relational database into multiple flat data files, which will need to be merged first. R has many packages for merging the datasets. The most popular functions are mutating joins from *tidyverse*. An important thing to check is whether the *by* variable(s) can uniquely define one record before merging the data. If not, joins will create all possible pairs, so you might double up your records. A easier way of checking is to count the number of records before and other *join*.



Source: https://twitter.com/hadleywickham/status/684407629259526144

One important benefit of using R is that it allows you to open many datasets and process them simultaneously. This is one of major differences compared to SPSS and Stata, which allow one data file at a time.

R also allows you to store many datasets in to one list. The benefit of list + data frame may not be trivial at this stage. But it is extremely powerful to make you code simple and efficient we you work with repetitive operations with datasets having the same structure.

```
data1 <- tibble(a = 1:4, b = 2:5)
data2 <- tibble(c = 3:6, d = 4:7)
ManyDatasets <- list(data1, data2)
ManyDatasets
```

```
## [[1]]
## # A tibble: 4 x 2
##       a     b
```

```
##   <int> <int>
## 1     1     2
## 2     2     3
## 3     3     4
## 4     4     5
##
## [[2]]
## # A tibble: 4 x 2
##       c     d
##   <int> <int>
## 1     3     4
## 2     4     5
## 3     5     6
## 4     6     7
```

Inspection

An important stage of pre-processing is to have a global understanding of your data:

- Is the number of observations of your data correct? (sometimes it takes some efforts to finalize the total sample size particularly for longitudinal studies as participants may withdraw the study and request their data to be removed).
- Are factor variables stored as character or numeric ?
- Is the data file long or wide when there are multiple observations for one participant?
- How is missing data coded?
- What's the proportion of missing in different variables? Is there any variable with substantial missing data? If so why it is the case?
- Are the nested variables correctly entered? i.e. Drinking frequency vs whether drinks or not.
- Are date variables correct?
- Is the raw data matching the definitions in the data dictionary?

There are a few functions that are quite useful, including *summary()*, *str()*, *psych::describe()*, *tibble::glimpse()* … The function I used the most is *describe()* from *Hmisc* which is similar with the codebook command in Stata.

```
Hmisc::describe(iris)
```

```
## iris
##
##  5  Variables      150  Observations
## ---------------------------------------------------------------------------
## Sepal.Length
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      150        0       35    0.998    5.843   0.9462    4.600    4.800
##      .25      .50      .75      .90      .95
##    5.100    5.800    6.400    6.900    7.255
##
## lowest : 4.3 4.4 4.5 4.6 4.7, highest: 7.3 7.4 7.6 7.7 7.9
## ---------------------------------------------------------------------------
## Sepal.Width
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      150        0       23    0.992    3.057   0.4872    2.345    2.500
##      .25      .50      .75      .90      .95
##    2.800    3.000    3.300    3.610    3.800
##
## lowest : 2.0 2.2 2.3 2.4 2.5, highest: 3.9 4.0 4.1 4.2 4.4
## ---------------------------------------------------------------------------
## Petal.Length
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      150        0       43    0.998    3.758    1.979     1.30     1.40
##      .25      .50      .75      .90      .95
##     1.60     4.35     5.10     5.80     6.10
##
## lowest : 1.0 1.1 1.2 1.3 1.4, highest: 6.3 6.4 6.6 6.7 6.9
## ---------------------------------------------------------------------------
## Petal.Width
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##      150        0       22     0.99    1.199   0.8676      0.2      0.2
##      .25      .50      .75      .90      .95
##      0.3      1.3      1.8      2.2      2.3
```

```
##
## lowest : 0.1 0.2 0.3 0.4 0.5, highest: 2.1 2.2 2.3 2.4 2.5
## ---------------------------------------------------------------------------
## Species
##        n  missing distinct
##      150        0        3
##
## Value          setosa versicolor  virginica
## Frequency          50         50         50
## Proportion      0.333      0.333      0.333
## ---------------------------------------------------------------------------
```

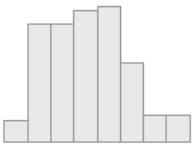
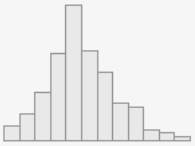
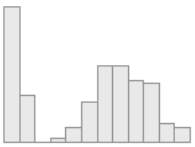
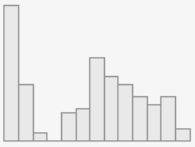
dfSummary from *summarytools* is a new function being developed which provides more comprehensive evaluations including distributions, distinct values and missing data.

```
summarytools::view(summarytools::dfSummary(iris))
```

Data Frame Summary

iris

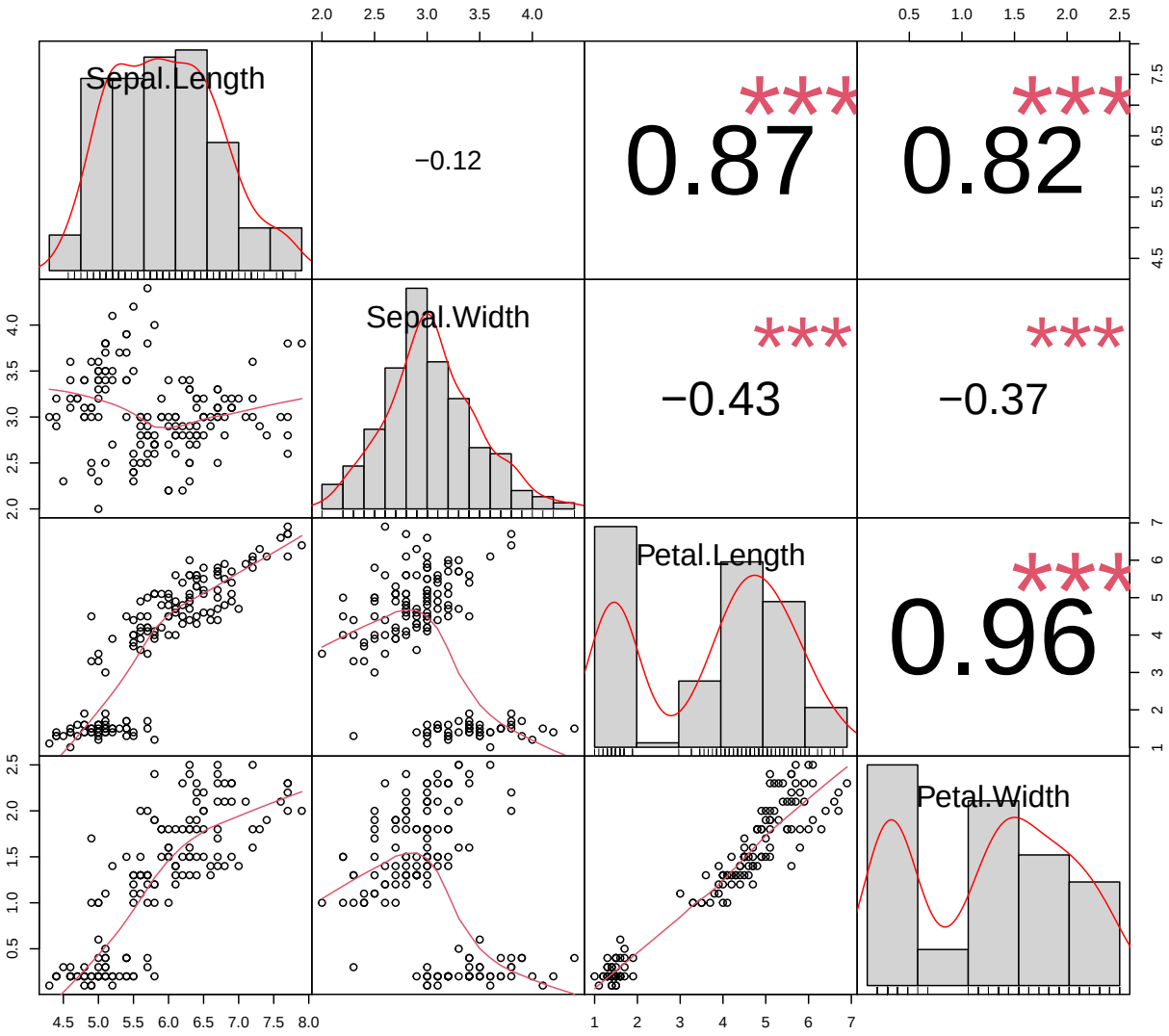
Dimensions: 150 x 5
Duplicates: 1

No	Variable	Stats / Values	Freqs (% of Valid)	Graph	Valid	Missing
1	Sepal.Length [numeric]	Mean (sd) : 5.8 (0.8) min < med < max: 4.3 < 5.8 < 7.9 IQR (CV) : 1.3 (0.1)	35 distinct values		150 (100.0%)	0 (0.0%)
2	Sepal.Width [numeric]	Mean (sd) : 3.1 (0.4) min < med < max: 2 < 3 < 4.4 IQR (CV) : 0.5 (0.1)	23 distinct values		150 (100.0%)	0 (0.0%)
3	Petal.Length [numeric]	Mean (sd) : 3.8 (1.8) min < med < max: 1 < 4.3 < 6.9 IQR (CV) : 3.5 (0.5)	43 distinct values		150 (100.0%)	0 (0.0%)
4	Petal.Width [numeric]	Mean (sd) : 1.2 (0.8) min < med < max: 0.1 < 1.3 < 2.5 IQR (CV) : 1.5 (0.6)	22 distinct values		150 (100.0%)	0 (0.0%)
5	Species [factor]	1. setosa 2. versicolor 3. virginica	50 (33.3%) 50 (33.3%) 50 (33.3%)		150 (100.0%)	0 (0.0%)

Generated by summarytools 0.9.8 (R version 4.0.2)
2021-01-22

You might also need to check multivariate associations briefly to check if there were any variables were coded in the wrong direction (i.e. some psychological instruments requires reverse coding). This can be achieved via simply checking, scatter plots and correlations.

```
library("PerformanceAnalytics")
PerformanceAnalytics::chart.Correlation(iris[1:4],
    histogram = TRUE, pch = 19)
```

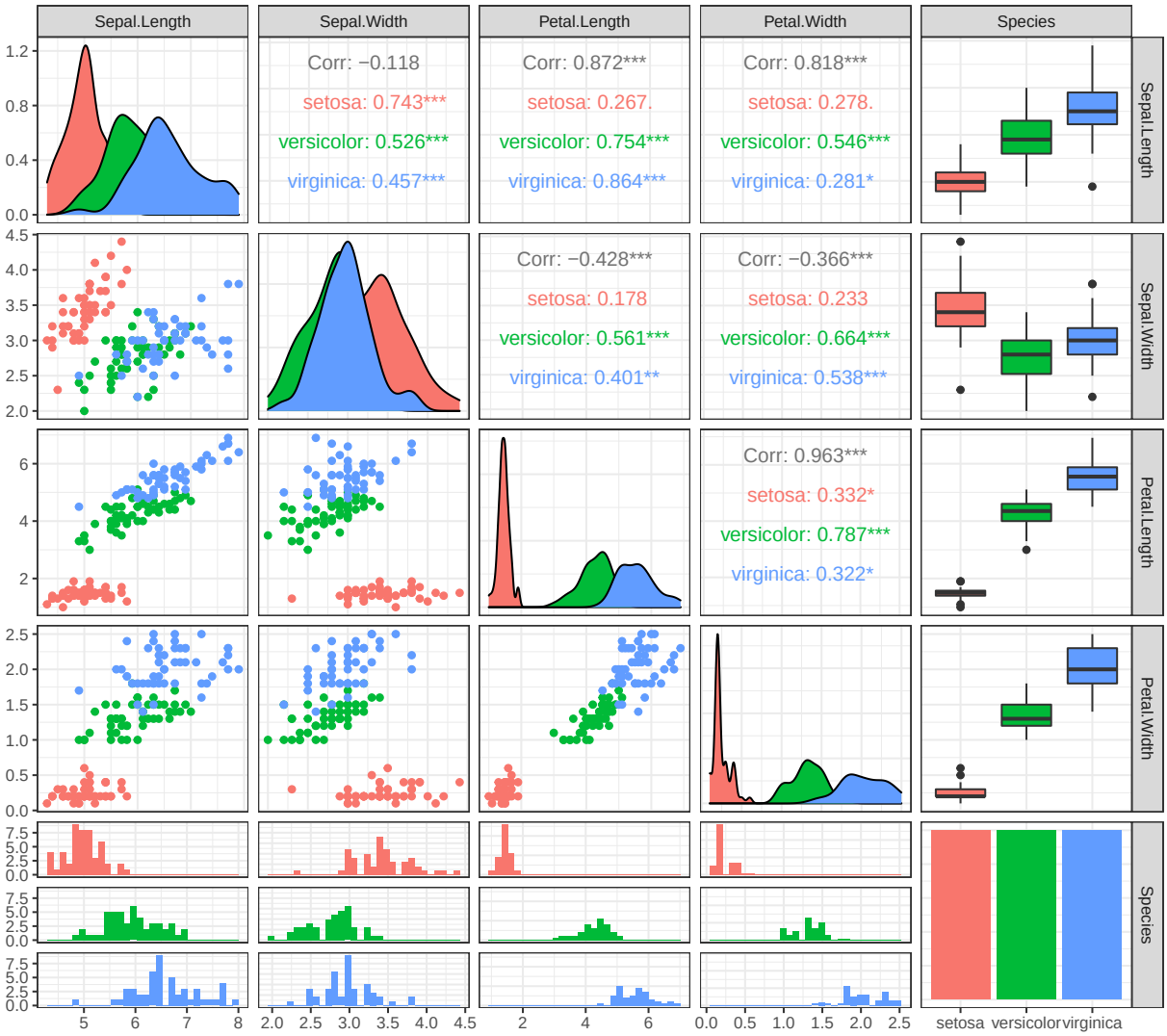


ggpairs is an interesting library I found recently, which provides comined multivariate plots.

```
library(GGally)

GGally::ggpairs(iris, aes(color = Species)) +

    theme_bw()
```



There are many other types of plots displaying multivariate associations, such as heatmaps, mosaic plots, flow diagrams etc. Interested readers can find more examples in Data Visualization with R by Rob Kabacoff.

Data cleaning

After the data is properly organised, you will need to start the data cleaning phase. The total workload depends on the type of source data received. If you are lucky that the data is collected using a more advanced tool such as Redcap (designed by a competent data manager), the process may not be substantially tedious. Still, generally data scientists and statisticians spend 60%-80% of their time in data cleaning and pre-processing. So be prepared!!

Another issue with data cleaning is that there is no clear definition of what is a "cleaned" data. Hadley Wickham provided more general definition of tidy vs messy data https://vita.had.co.nz/papers/tidy-data.pdf. This is my definition of cleaned data related to health studies:

A cleaned data is one dataset with (i) variables validated, ordered, clearly named and labeled, (ii) categorical variables properly categorized with clear labels, (iii) derived and transformed variables properly included, (iv) and the required format (by the analysis) established.

"Validated" refers to a range of expects including: consistency (male participants should not reported menstrual cramps in PHQ15), validity (poor quality tests sometimes needed to be excluded, should be coded as missing, data out of range should not be included), variable type (numeric variables should not include strings i.e. <5), uniformity (using the same (unit of measure)

It will be better to store related variables closer in the dataset so it is easier check. Sometimes variable naming can be problematic and confusing. For example, when individual items in K10 is named as K1, K2… K10, then the last item name is overlapping with the general understanding of K10, which represents the summary score of K10.

Categorical variables are sometimes stored as numeric variables (1,2,3,999), which would need to be labeled as factor variables. Sometimes there is a "other" category which affiliate with a free-text field. Often what was entered will need to be re-evaluated and code back to existing categories. Categorical variables from source data are often includes much more detailed categories than needed for analysis. It's better to create consistent categorisations. For example, gender identity sometime can be collected as male, female, transgender, gender neutral, agender, pangender, genderqueer etc. Most of these categories would include very small numbers that cannot be use in the analysis. Then it will be a good practice to generate another shorter version of the gender identity variable in the cleaned data (i.e. male, female and non-binary). Then this variable can be used consistently across different analysis.

In psychology, lots of measures were conducted using instruments with multiple items. Hence derived variables (i.e. total scores, weighted scores) will need to be generated in the cleaned data.

The "format of the data" refers to the format of data required for different types of analysis, i.e. cross-sectional analysis requires wide version of data (one row per person); longitudinal analysis requires long version of data (multiple rows per person with each row representing one observation with unique identifiers for each person or a higher level of clustering group); survival analysis requires censored data (one or multiple rows per person depending on whether evaluating single outcome or recurrent outcomes with time scale and censoring status defined) . Sometimes you will need to have both the wide and long version of cleaned data stored.

More often your final "cleaned" data will change over time for most of larger studies due to issues identified when the study evolves, i.e. additional errors identified, more data collected etc. Hence you will need to create a version control system to capture these changes. A common practice is to store the "cleaned" versions of data with a time stamp, i.e. "A_brilliant_study_cleaned_V1_03032012.rds".

Tidyverse data cleaning routine

The data cleaning code pre Hadley Wickham era normally look like this:

```
data$group <- as.factor(data$group)
data$time <- as.factor(data$time)
data$gender <- as.factor(data$gender)
data$work_status <- as.factor(data$work_status)
data$diagnosis <- as.factor(data$diagnosis)
data$therapy <- as.factor(data$therapy)
```

The operation this chunk of code was trying to achieve was simply converting a few variables from string to factor. When there are only a few variables to edit, this type of code is still readable. As the number of variables increase and number of operations increase, you code will soon be too hard to read and amend. Hadley introduced a game changing new philosophy: **tidyverse**. In this new philosophy defines not only what the tidy data should look like, but also what tidy code should look like. Using tidyverse grammar (using *dplyr* package), the above data cleaning code became:

```
data1 <- data %>% mutate_at(vars(group, time,
    gender, work_status, diagnosis, therapy),
    as.factor)
```

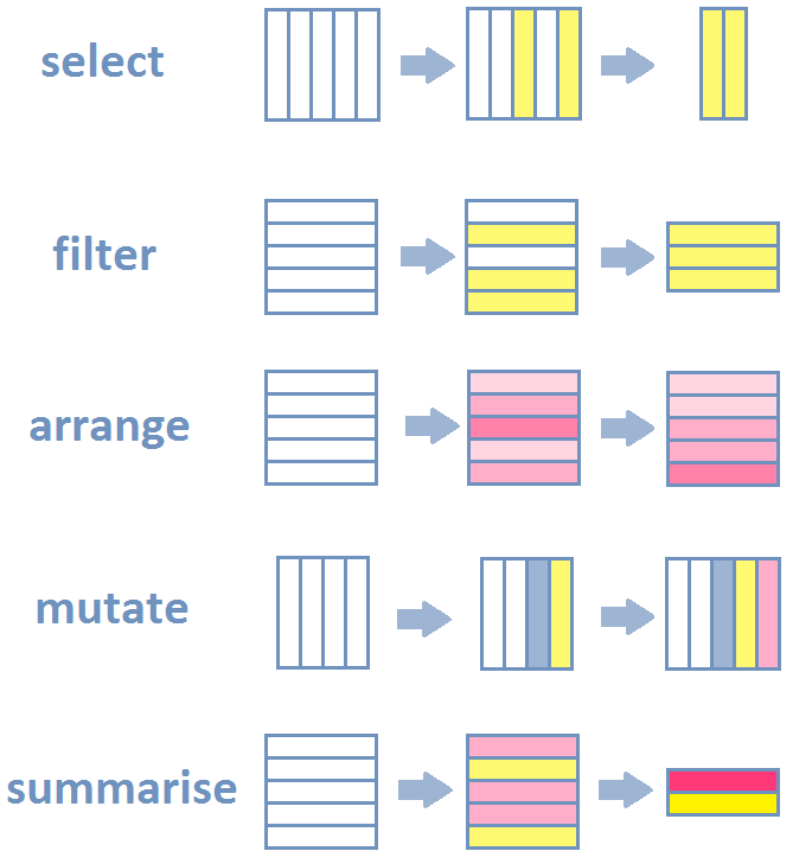
How to read this is simple, I am manipulating 6 variables (changing them to factor variables) in the *data* dataset and store the new dataset in *data1*. The code is now much more concise and readable. The beauty of this new style also lie on the use of pipe operation %>% in building a series of operations on the same dataset.

```
data1 <- data %>% mutate_at(vars(group, time,
    gender, work_status, diagnosis, therapy),
    as.factor) %>% mutate_at(vars(starts_with("k10_q")),
```

```
    as.numeric) %>% mutate(k10_total = rowSums(select(.,
    k10_q1:k10_q10)))
```

The above code change 6 variables to factor variables, change individual K10 items to numeric and then calculate total scores of K10 as the sum of items from k10_q1 to k10_q10.

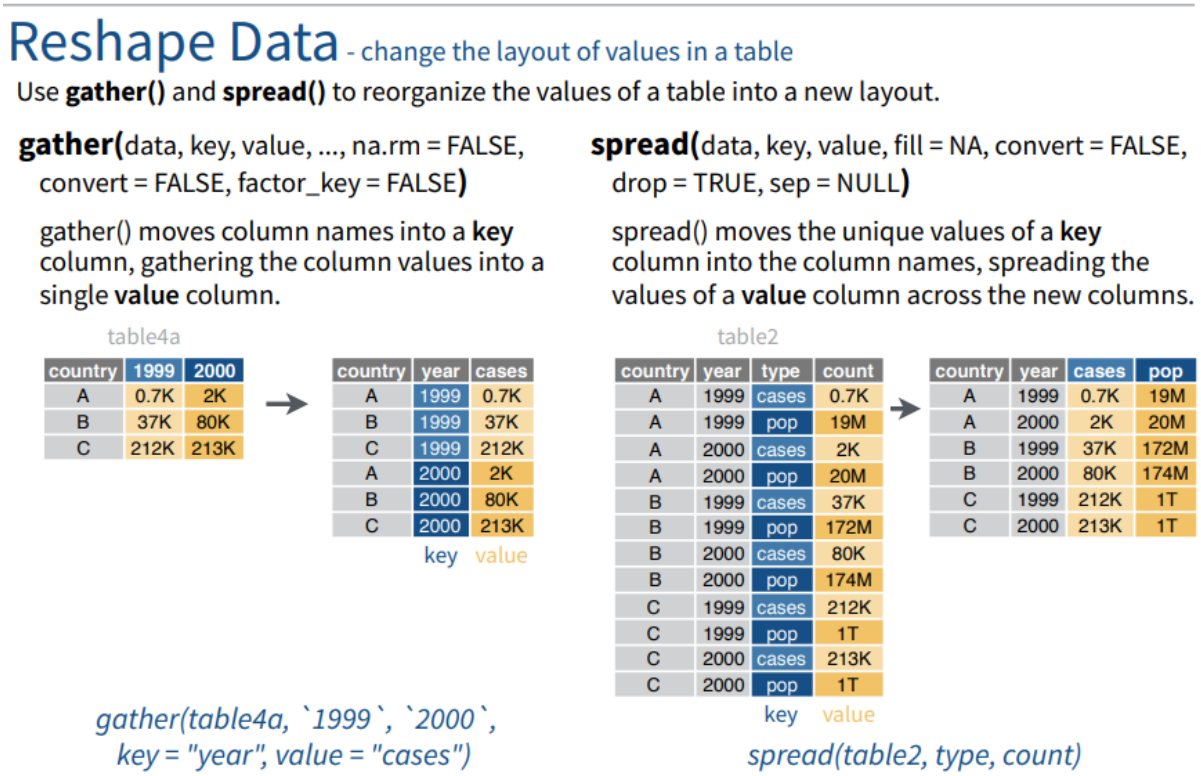
There are a range of useful functions in dplyr such as, group_by(), filter(), select(), summarize()… The detailed introduction on *dplyr* can be found in [Chapter 4 of Introduction to Data Science] (https://rafalab.github.io/dsbook/tidyverse.html) by Hadley and *dplyr* cheat sheet If you are still using the traditional "hard" coding style, it is time to upgrade!



source: http://perso.ens-lyon.fr/lise.vaudor/dplyr/

Re-shaping the data

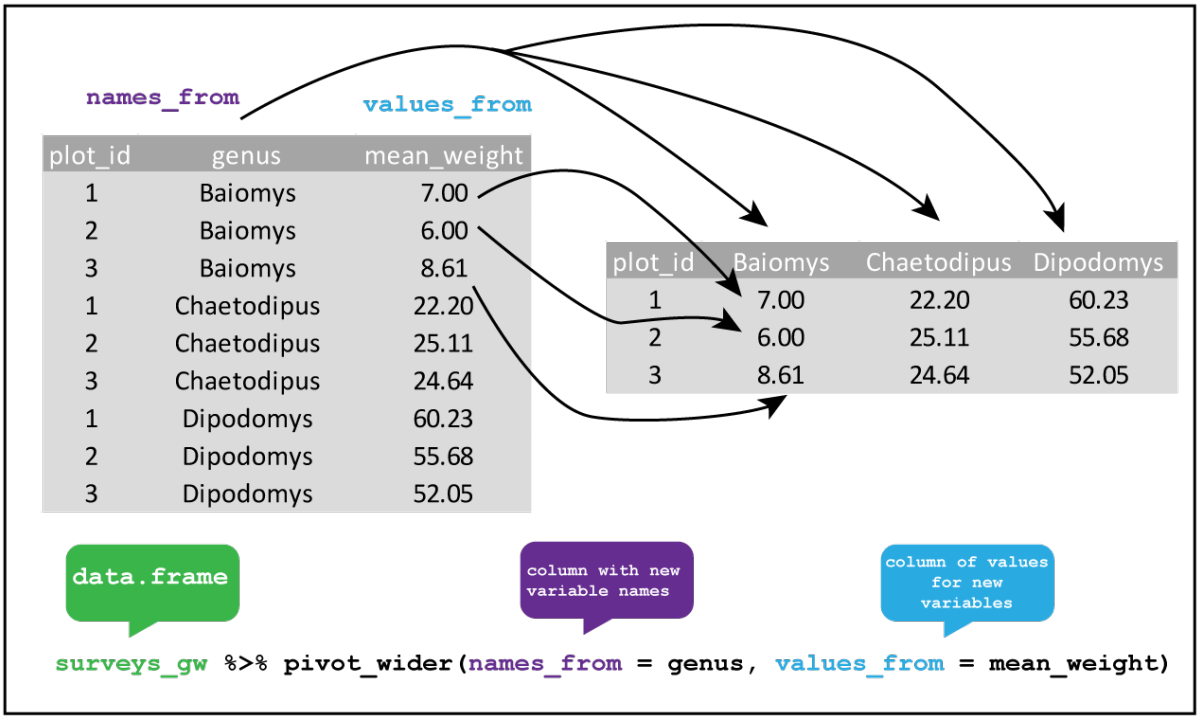
It is common to have to work with both wide(or wider) and long(or longer) format of the data during your data cleaning process. You are also frequently required to change data between these two types, see example of reshaping using *gather()* and *spread()* from *tidyr* package below:

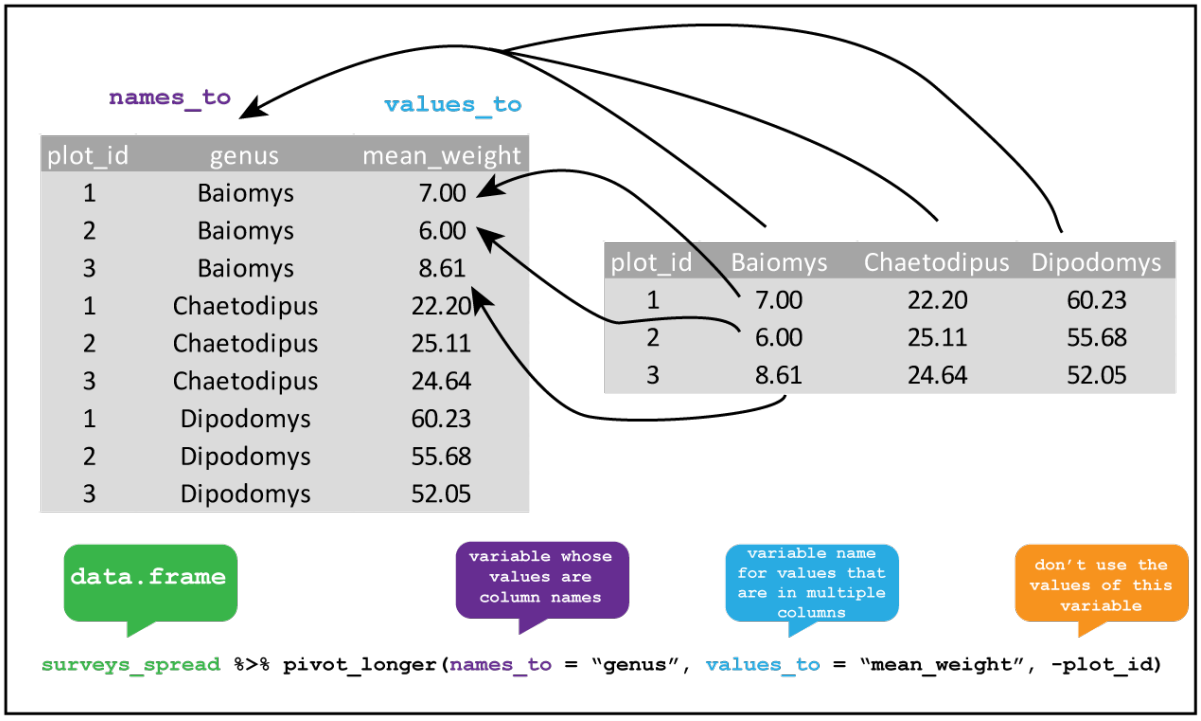


source: https://dannytabach.com/connecting-r-and-google-sheets.html

You can also use the *reshape* function from *stats* package which can work with multiple columns.

Two new functions in *tidyr* can also work with mutiple columns and rows

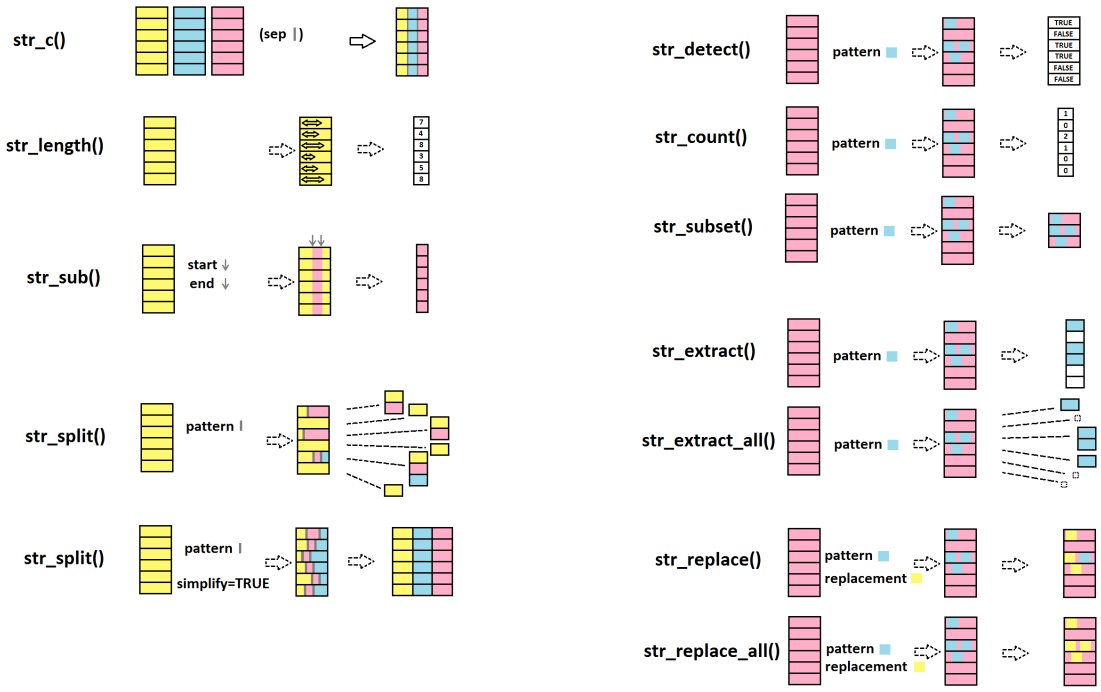




source: https://ab604.github.io/docs/coding-together-2019/data-wrangle-2.html

Other useful packages

String operation package *stringr*



Package working with date and time variables *lubridate*.

Apply family, see tutiral here

```
apply(iris, 2, table)
```

```
## $Sepal.Length
##
## 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.0 5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9 6.0 6.1 6.2
##   1   3   1   4   2   5   6  10   9   4   1   6   7   6   8   7   3   6   6   4
## 6.3 6.4 6.5 6.6 6.7 6.8 6.9 7.0 7.1 7.2 7.3 7.4 7.6 7.7 7.9
##   9   7   5   2   8   3   4   1   1   3   1   1   1   4   1
##
## $Sepal.Width
##
## 2.0 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.0
##   1   3   4   3   8   5   9  14  10  26  11  13   6  12   6   4   3   6   2   1
## 4.1 4.2 4.4
##   1   1   1
##
## $Petal.Length
##
## 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.9 3.0 3.3 3.5 3.6 3.7 3.8 3.9 4.0 4.1 4.2 4.3
##   1   1   2   7  13  13   7   4   2   1   2   2   1   1   1   3   5   3   4   2
## 4.4 4.5 4.6 4.7 4.8 4.9 5.0 5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9 6.0 6.1 6.3 6.4
##   4   8   3   5   4   5   4   8   2   2   2   3   6   3   3   2   2   3   1   1
## 6.6 6.7 6.9
##   1   2   1
##
## $Petal.Width
##
## 0.1 0.2 0.3 0.4 0.5 0.6 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3
##   5  29   7   7   1   1   7   3   5  13   8  12   4   2  12   5   6   6   3   8
## 2.4 2.5
##   3   3
##
## $Species
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

purrr for functional programming

Source: *purrr* tutorial by Rebecca Barter

Notes for yourself and others

There is a famous quote about writing programming code

"When I wrote this code, only God and I understood what it did. Now only God knows"

The quote was originally from *The Barretts of Wimpole Street* when Robert Browning was trying to work out what he meant in a poem. However it is the reality for a vast majority of programmers, data scientists and statisticians.

I have developed a respiratory infection transmission model during my PhD time as an early R user. It was only published in my PhD thesis not in scientific publications due to lack of interest in airborne disease transmission. Then COVID happened, and we thought it will be important to publish this work in a reproducible format. However, my code back then was so terrible that it took me half a year to figure out what I did in 3 months. I had excuses back then, i.e. stress to finish the work on time, difficulties in including equations in code comments… Now my code is slightly better with my following rules:

- Always use R markdown (or R notebook) file for data cleaning
- Document decisions in text field of the R markdown file
- Include specific code comments alongside the code

The modern version of my data cleaning code looks like this:

Calculate total scores and obtain OASIS categories using cut-off value of >=8 following Moore et al 2015.

```
OASIS <- OASIS %>% # calculate total score
mutate(oasis_total = rowSums(select(., OASISq1:OASISq5))) %>%
    # define groups
mutate(oasis_total_group = cut(oasis_total, breaks = c(-1,
    7, Inf), labels = c("Absence of anxiety disorder ",
    "Presence of anxiety disorder"))) %>% # remove useless columns
select(-c(AxDate, status)) %>% # rename variables for consistency
setnames(old = paste0("OASISq", 1:5), new = paste0("oasis_q",
    1:5))


# check
table(OASIS$oasis_total, OASIS$oasis_total_group,
    exclude = NULL)
```

Code checking

It's important to cross check you data cleaning code. The common self-checking points are

- When a new variable is generated based on old variable, check whether it is correct via cross-tabulate it with the original variable.
- When performing merging datasets, always check if you have correctly matched records, weather the total number of rows is correct
- When calculate total scores, choose to eyeball a few records to see if it is correct.
- Cross check you final data with raw data (look at the Hmisc::describe() of the raw data and the updated data side by side)
- When change the data between long and wide data type, manually check if the number of rows are as expected (i.e. 1000 participants completed baseline survey and 500 completed follow-up, the long data should have 1500 rows).

- Whether nested variables are coded correctly. If you have a drinking quantity variable which was only asked when people reported drinking. Then you should code the quantity as 0 if they did not drink (unless you have a good reason not to). This is because sometimes you might accidentally exclude observations if these nested variables were coded as missing. Also if you use multiple imputation, coding them to missing (intentionally missing) will also cause problems.

Common pitfalls

Missing value

Missing values are coded as *NA* in R data. See instructions on dealing with R missing values here and here When tabulate data always use *exclude=NULL*, or you can use tidyverse style function *tabyl* from *janitor* package which can also store cross-tabulate results in a data frame.

```
data <- tibble(a = c(1, 2, 3, 4), b = c(4, NA,
    5, 6), c = c(7, 8, 9, NA))
table(data$a, data$b, exclude = NULL)
```

```
##
##     4 5 6 <NA>
##   1 1 0 0    0
##   2 0 0 0    1
##   3 0 1 0    0
##   4 0 0 1    0
```

```
library(janitor)
data %>% tabyl(a, b)
```

```
##  a 4 5 6 NA_
##  1 1 0 0   0
##  2 0 0 0   1
##  3 0 1 0   0
##  4 0 0 1   0
```

When using rowSums and rowMeans functions, you have to be mindfull of missing data, see following example.

```
data %>% mutate(sum = rowSums(., na.rm = FALSE),
    mean = rowMeans(., na.rm = FALSE))
```

```
## # A tibble: 4 x 5
##       a     b     c   sum  mean
##   <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1     4     7    12  4
## 2     2    NA     8    NA NA
## 3     3     5     9    17  5.67
## 4     4     6    NA    NA NA
```

```
data %>% mutate(sum = rowSums(., na.rm = TRUE),
    mean = rowMeans(., na.rm = TRUE))
```

```
## # A tibble: 4 x 5
##       a     b     c   sum  mean
##   <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1     4     7    12  4
## 2     2    NA     8    10  5
## 3     3     5     9    17  5.67
## 4     4     6    NA    10  5
```

Duplication

Sometimes there are duplicated records in the database or being created in the data pre-processing stage. This is perticularly problematic when you are working with large datasets from secondary source. You have to be mindful to keep track of number of observations through out the data cleaning code.

The trap of white space

R string fields sometimes include leading and trailing white spaces, which you cannot see when *View()* the data. However, this will cause issues when you are trying to categories data or merging the data, see following example

```
data1 <- tibble(key = c("a", "b", "c ", "d"),
    number_1 = c(1, 2, 3, 4))
data2 <- tibble(key = c("a", "b", "c"), number_2 = c(5,
```

```
    6, 7))
data1 %>% left_join(data2)
```

```
## Joining, by = "key"
```

```
## # A tibble: 4 x 3
##   key   number_1 number_2
##   <chr>    <dbl>    <dbl>
## 1 "a"          1        5
## 2 "b"          2        6
## 3 "c "         3       NA
## 4 "d"          4       NA
```

The *str_trim* function removes the leading and trailing white space, which will make sure the data can be joined correctly.

```
data1 %>% mutate(key = str_trim(key)) %>% left_join(data2)
```

```
## Joining, by = "key"
```

```
## # A tibble: 4 x 3
##   key   number_1 number_2
##   <chr>    <dbl>    <dbl>
## 1 a            1        5
## 2 b            2        6
## 3 c            3        7
## 4 d            4       NA
```

Be mindfull of which function you are calling

One of the issue with R is that a few functions can have the same name. Here are the list of conflicts in my attached libraries.

```
library(conflicted)
conflict_scout()
```

```
## 11 conflicts:
## * `as.Date`         : [zoo]
## * `as.Date.numeric`: [zoo]
## * `chisq.test`      : janitor, stats
```

```
## * `filter`          : [dplyr]
## * `first`           : xts, dplyr
## * `fisher.test`     : janitor, stats
## * `group_rows`      : kableExtra, dplyr
## * `lag`             : dplyr, stats
## * `last`            : xts, dplyr
## * `legend`          : PerformanceAnalytics, graphics
## * `Position`        : ggplot2, base
```

R's default conflict resolution system gives precedence to the most recently loaded package. So when calling the functions, it will be better to refer to the package name (*dplyr::summarize()* vs *Hmisc::summarize()*).

Type stability

This is referred as the "WAT" moment with R programming

```
# combining factors makes integers
c(factor("a"), factor("b"))
```

```
## [1] 1 1
```

```
# > [1] 1 1


# combing dates and date-times give incorrect
# values
dt <- as.Date("2020-01-1")
dttm <- as.POSIXct("2020-02-01 09:00", tz = "America/New_York")
c(dt, dttm)
```

```
## [1] "2020-01-01" "2020-02-01"
```

```
c(dttm, dt)
```

```
## [1] "2020-02-01 09:00:00 EST" "1970-01-01 00:04:22 EST"
```

Be careful with R's automatic coercion.

Documentation for the never ending data cleaning process

Documentation is the key to good data quality. Historically, studies are small and isolated. Once data cleaning is completed (mostly via eyeballing), you will conduct the analysis and the data files (on paper) will be sealed in a draw and no one will be looking at them. Nowadays, we often work with large and evolving datasets, i.e. longitudinal, administrative and registry data. Hence it is important to keep a good documentation that surmise important decisions, data errors identified, changes in data cleaning process, version history, and associated locations where datasets and historical data cleaning files are stored. I often use a separate word document to keep track of these issues. When there are multiple users using the "cleaned" data, it is also important to keep a data dictionary of the cleaned data, in which you note down, the time of the variable introduced to the dataset, any changes in coding and cleaning to the variable as well as other important notice that you would like your users to know.

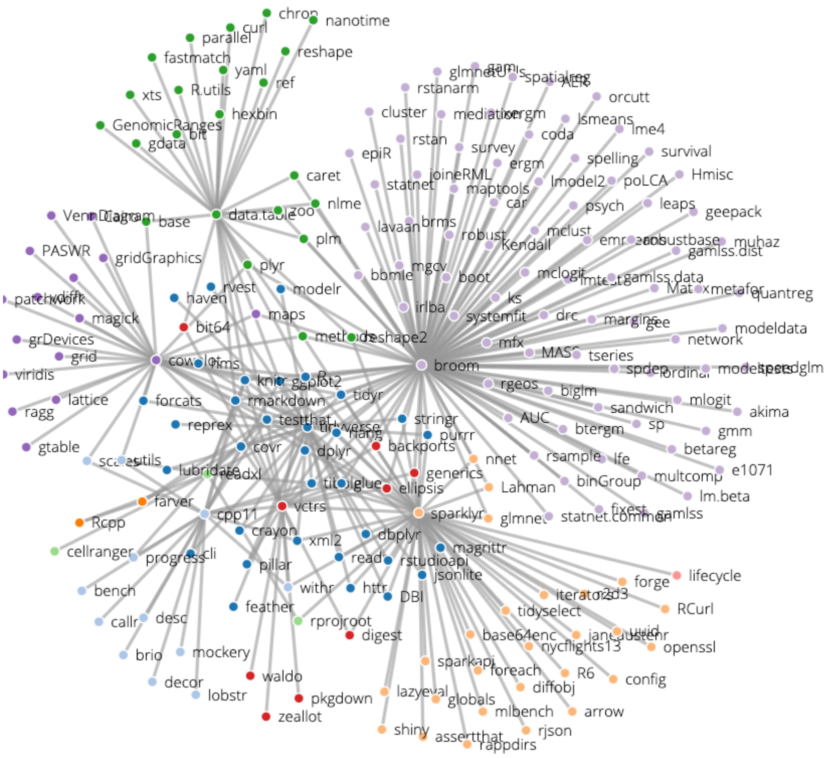
Final remarks

There are a few additional key points need to be noted for data pre-processing :

- Backup files (backup original data, intermediate "cleaned" versions of the data and cleaning files) frequently in reliable place(s) so you can uncover historical issues easily. You can zip the files to prevent people accidentally changing the files or store the backup files in locations with limited and authorised access.
- Establish a code auditing work flow. If possible, have all the code and final data reviewed by a second person.

Statistical analysis with R

R has over 20K indexed packages, over 3M indexed functions and over 3m users worldwide. So R has explosive level of capacity and flexibility as well as a tremendous workforce. R is now equipped with all or most of functionalities of many other statistical/ML packages: Stata, SPSS, SAS, Mplus... (Python is still irreplaceable in deep mining).



Source: https://www.rdocumentation.org/trends

A few common analysis and packages are listed here

Analysis	Function and package
Linear regression	stats::lm
Generlised linear regression	stats::glm
Generalized additive model	mgcv::gam
ARIMA (time series)	forecast::arima
Cox proportional hazards (survival analysis)	survival::coxph
Linear mixed effect	lme4::lmer
Generalised linear mixed effect	lme4::glmer
CFA/SEM	lavaan::cfa, lavaan::sem
Meta-regression	metafor::rma , meta::metareg
Clustering	stats::hclust, stats::kmeans, cluster::pam
Principal component analysis	stats::princomp

One of the best books for learning statistical methods using R is An Introduction to Statistical Learning by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. The advanced version of this book is The Elements of Statistical Learning by Trevor Hastie, Robert Tibshirani and Jerome Friedman. However, both of the books are from more machine learning (ML) perspectives. A similar free book for psychology students is Advanced Statistics Using R by Zhiyong Zhang and Lijuan Wang. Discovering Statistics Using R by Andy Field is also one of the best R statistical handbooks. There are many content specific books that are quite interesting and useful:

- An Introduction to Generalized Linear Models by by Annette J. Dobson, Adrian G. Barnett. R code were given in some examples
- Forecasting: Principles and Practice by Rob J Hyndman and George Athanasopoulos for time-series analysis
- Applied Survival Analysis Using R by Dirk F. Moore
- Statistical Rethinking A Bayesian Course with Examples in R and Stan by Richard McElreath. Associated lectures can be found here
- Geocomputation with R by Robin Lovelace, Jakub Nowosad, Jannes Muenchow
- Modern Psychometrics with R by Patrick Mair
- Multilevel Modeling Using R by W. Holmes Finch, Jocelyn E. Bolin, Ken Kelley

Exploratory data analysis (EDA)

Once your data is cleaned, you will normally start with the "exploratory" analysis phase. The "exploratory" here refers to evaluating variable properties, such as distribution, bivariate association and dimensionality, rather than explore "significant" results. More often exploratory results, such as distributions, will also need to be included in your report/thesis/paper. This process provides you with general ideas about your dataset, clues on appropriateness of models (Poisson regression vs negative binomial regression), check outliers, identify any inconsistencies with your understanding (I call this debugging your data).

Exploratory Data Analysis with R by Roger Peng includes lots of useful examples. The Chapter 7 of R for Data Science also includes a nice intro to EDA. Importantly, you should not use EDA to generate research questions. Your research questions should be pre-defined and your analysis will need to be planed ahead. EDA can help you refine models, but you should **not** look too hard in to it, which might turn your years of work into another piece of "fished" or "hacked" results out of null-association (winner's curse). See a few interesting publications here: The Extent and Consequences of P-Hacking in Science and The garden of forking paths: Why multiple comparisons can be a problem, even when there is no "fishing expedition" or "p-hacking" and the research hypothesis was posited ahead of time.

Table 1

Simple descriptive table, normally referred as the "Table 1", is where your data story start. In this table you will need to provide enough information about the data that was used in then analyses so your readers can have a over view of it's profiles. For most of health analysis, your reporting unit is participants and you would generally be required to report demographics characteristics and risk factors in this table. R has lots of existing packages that automatic generate table 1. *tableby* from *arsenal* is very easy to use when you directly display your results. However, it is difficult to be extracted to tibble/data frame for further manipulations.

```
library(arsenal)
my_controls <- tableby.control(test = T, total = F,
    digits = 1, digits.pct = 1, digits.p = 3,
    numeric.test = "anova", cat.test = "chisq",
    numeric.stats = c("meansd", "medianq1q3",
        "range", "Nmiss2"), cat.stats = c("countpct",
        "Nmiss2"), stats.labels = list(meansd = "Mean (SD)",
```

```
        medianq1q3 = "Median (Q1, Q3)", range = "Min - Max",
        Nmiss2 = "Missing"))
table <- tableby(Species ~ Sepal.Length + Sepal.Width +
    Petal.Length + Petal.Width, data = iris, control = my_controls)

summary(table, title = "Characteristics by species in iris data")
```

Table 1: *Characteristics by species in iris data*

	setosa (N=50)	versicolor (N=50)	virginica (N=50)	p value
Sepal.Length				< 0.001
Mean (SD)	5.0 (0.4)	5.9 (0.5)	6.6 (0.6)	
Median (Q1, Q3)	5.0 (4.8, 5.2)	5.9 (5.6, 6.3)	6.5 (6.2, 6.9)	
Min - Max	4.3 - 5.8	4.9 - 7.0	4.9 - 7.9	
Missing	0	0	0	
Sepal.Width				< 0.001
Mean (SD)	3.4 (0.4)	2.8 (0.3)	3.0 (0.3)	
Median (Q1, Q3)	3.4 (3.2, 3.7)	2.8 (2.5, 3.0)	3.0 (2.8, 3.2)	
Min - Max	2.3 - 4.4	2.0 - 3.4	2.2 - 3.8	
Missing	0	0	0	
Petal.Length				< 0.001
Mean (SD)	1.5 (0.2)	4.3 (0.5)	5.6 (0.6)	
Median (Q1, Q3)	1.5 (1.4, 1.6)	4.3 (4.0, 4.6)	5.5 (5.1, 5.9)	
Min - Max	1.0 - 1.9	3.0 - 5.1	4.5 - 6.9	
Missing	0	0	0	
Petal.Width				< 0.001
Mean (SD)	0.2 (0.1)	1.3 (0.2)	2.0 (0.3)	
Median (Q1, Q3)	0.2 (0.2, 0.3)	1.3 (1.2, 1.5)	2.0 (1.8, 2.3)	
Min - Max	0.1 - 0.6	1.0 - 1.8	1.4 - 2.5	
Missing	0	0	0	

Another useful Table 1 function is *tbl_summary* from *gtsummary* , which you can extract results as tibble.

```
library(gtsummary)
```

```
## #BlackLivesMatter
```

```
tbl <- iris %>% tbl_summary(missing = "ifany",
    statistic = where(is.numeric) ~ "{mean} ({sd})",
    by = Species) %>% bold_labels() %>% add_p(test = list(all_numeric() ~
    "aov"), pvalue_fun = ~style_pvalue(.x, digits = 3))

tbl
```

```
## Table printed with `knitr::kable()`, not {gt}. Learn why at
## http://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
## To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**setosa**, N = 50[1]	**versicolor**, N = 50[1]	**virginica**, N = 50[1]	**p-value**[2]
Sepal.Length	5.01 (0.35)	5.94 (0.52)	6.59 (0.64)	<0.001
Sepal.Width	3.43 (0.38)	2.77 (0.31)	2.97 (0.32)	<0.001
Petal.Length	1.46 (0.17)	4.26 (0.47)	5.55 (0.55)	<0.001
Petal.Width	0.25 (0.11)	1.33 (0.20)	2.03 (0.27)	<0.001

[1]Statistics presented: Mean (SD)

[2]Statistical tests performed: One-way ANOVA

gtsummary object can be extracted as tibble and further edited.

```
tbl <- tbl %>% as_tibble()

# remove style
names(tbl) <- str_replace_all(names(tbl), "\\**",
    "")
indent <- which(str_count(tbl$Characteristic,
    "__") == 0)
tbl$Characteristic <- str_replace_all(tbl$Characteristic,
    "__", "")

# change first row
tbl$Characteristic <- str_remove(tbl$Characteristic,
```

```
    "Sepal.")
tbl$Characteristic <- str_remove(tbl$Characteristic,
    "Petal.")
tbl <- tbl %>% add_row(Characteristic = "Sepal",
    .before = 1) %>% add_row(Characteristic = "Petal",
    .before = 4)


options(knitr.kable.NA = "")
knitr::kable(tbl, booktabs = TRUE, linesep = "") %>%
    kable_styling(bootstrap_options = "striped") %>%
    add_indent(c(2, 3, 5, 6))
```

Characteristic	setosa, N = 50	versicolor, N = 50	virginica, N = 50	p-value
Sepal				
Length	5.01 (0.35)	5.94 (0.52)	6.59 (0.64)	<0.001
Width	3.43 (0.38)	2.77 (0.31)	2.97 (0.32)	<0.001
Petal				
Length	1.46 (0.17)	4.26 (0.47)	5.55 (0.55)	<0.001
Width	0.25 (0.11)	1.33 (0.20)	2.03 (0.27)	<0.001

Regression models

Regression model follows very similar style of coding:

function(y~ x1 + x2 + x3, data=data, additional specifications)

```
mod1 <- lm(Petal.Length ~ Species + Sepal.Width *
    Sepal.Length, data = iris)
summary(mod1)
```

```
##
## Call:
## lm(formula = Petal.Length ~ Species + Sepal.Width * Sepal.Length,
##     data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.74506 -0.18607  0.00386  0.16942  0.79286
##
```

```
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -2.14865    1.44426  -1.488  0.13901
## Speciesversicolor          2.18252    0.11214  19.462  < 2e-16 ***
## Speciesvirginica           3.06260    0.12854  23.825  < 2e-16 ***
## Sepal.Width                0.13127    0.48106   0.273  0.78535
## Sepal.Length               0.73177    0.24183   3.026  0.00294 **
## Sepal.Width:Sepal.Length -0.02912    0.08035  -0.362  0.71754
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2841 on 144 degrees of freedom
## Multiple R-squared:  0.975,  Adjusted R-squared:  0.9741
## F-statistic:  1121 on 5 and 144 DF,  p-value: < 2.2e-16
```

"Sepal.Width*Sepal.Length" is the interaction term. Most of regression models are easy to fit. The following lists are some of known issues and concerns for common models (some are not specific to R):

- Some GLMs, particularly Relative Risk regression also known as log-binomial regression (GLM with log link and binomial distribution model), have convergence problem (Williamson, Eliasziw, and Fick 2013). This is relate to parameter constraints at boundaries. For log-binomial regression, a package *logbin* was developed with advance convergence ability (use combinatorial EM to cycles through parameter space). For other GLMs, Bayesian models (i.e. using *brms*) can be used to overcome some of the convergence problems.
- Linear mixed effect models sometimes have convergence issues too. You can test performance of different optimizers, see example here.
- Multinomial logistic regression model *nnet::multinom* requires predictors to be roughly scaled to [0,1] or the fit will be slow or may not converge at all. Normally this is fine, however when you have some large values (i.e. IRSAD scores which is around 1000), the model will stop working. Hence it will be better to standardise those predictors.
- Sometimes there will be an error message "Error: no valid set of coefficients has been found: please supply starting values", this means the model is having trouble to initialize the optimization and require you to supply a set of starting values for intercept and

predictors. I normally run a model with reduced number of predictors and use the coefs from the reduced model as starting values.

- Set seed for models require random sampling, i.e. bootstrap.
- It will be better to use splines (i.e. *ns*) instead of square and cube terms to evaluate non-linear associations, see a review by Perperoglou et al. (2019).
- Different package will not necessary produce identical results, but they should generally agree with each other. If not something is wrong: either a bug in your code or in the existing code libraries.

```
library(logbin)
iris$Versicolor = as.numeric(iris$Species == "versicolor")
mod1 <- glm(Versicolor ~ Sepal.Length, family = binomial(link = "log"),
    data = iris)
mod2 <- logbin::logbin(formula(mod1), data = iris,
    method = "cem", trace = TRUE)
```

```
## logbin parameterisation 1/2
## Deviance = 190.9543 Iterations - 432
## logbin parameterisation 2/2
## Deviance = 190.2446 Iterations - 613
```

```
coefficients(summary(mod1))
```

```
##                Estimate Std. Error    z value   Pr(>|z|)
## (Intercept)  -1.6925062  0.8234556 -2.0553703 0.03984325
## Sepal.Length  0.1008402  0.1370450  0.7358181 0.46184139
```

```
coefficients(summary(mod2))
```

```
##                Estimate Std. Error    z value   Pr(>|z|)
## (Intercept)  -1.692716  0.8234646 -2.0556024 0.03982085
## Sepal.Length  0.100876  0.1370491  0.7360579 0.46169549
```

Extract results

Although R can print analysis results, it often will not present results that can be directly reportable. You often required to programe your own code to store and process results and then assemble proper tables and figures. Luckily there are many third party packages available

to extract results from common models. *sjPlot* is a useful package which can process many common model results.

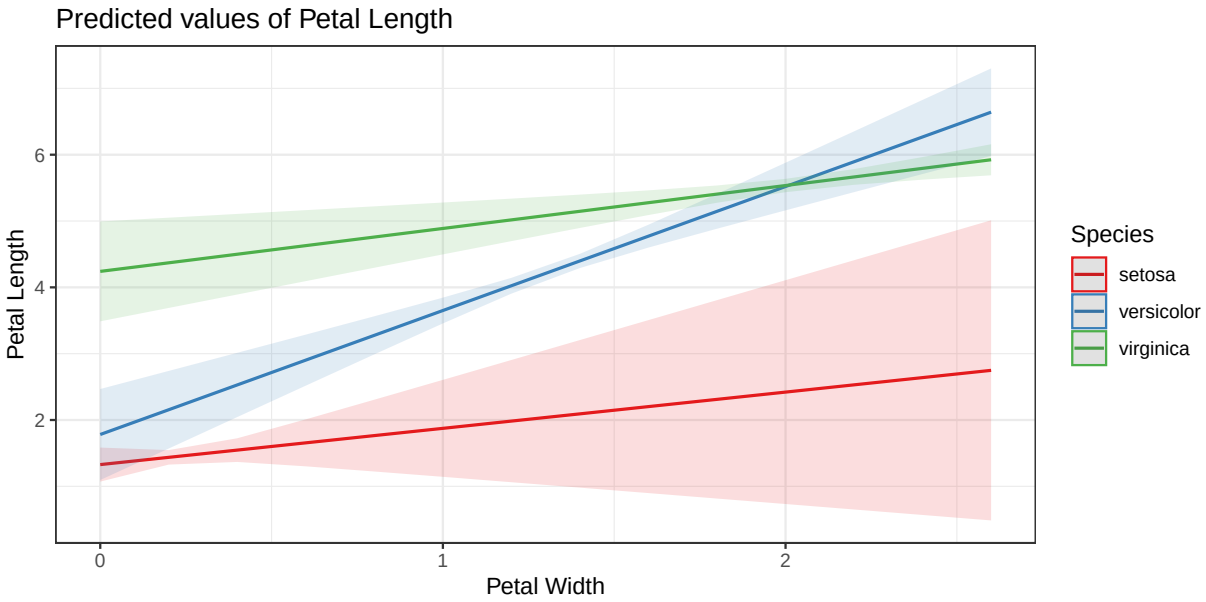
```
library(sjPlot)
mod1 <- lm(Petal.Length ~ Species, data = iris)
mod2 <- lm(Petal.Length ~ Species + Petal.Width,
    data = iris)
mod3 <- lm(Petal.Length ~ Species + Petal.Width +
    Species:Petal.Width, data = iris)
```

```
tab_model(mod1, mod2, mod3)
```

	Petal Length			Petal Length			Petal Length		
Predictors	*Estimates*	*CI*	*p*	*Estimates*	*CI*	*p*	*Estimates*	*CI*	*p*
(Intercept)	1.46	1.34 – 1.58	**<0.001**	1.21	1.08 – 1.34	**<0.001**	1.33	1.07 – 1.59	**<0.001**
Species [versicolor]	2.80	2.63 – 2.97	**<0.001**	1.70	1.34 – 2.06	**<0.001**	0.45	-0.28 – 1.19	0.227
Species [virginica]	4.09	3.92 – 4.26	**<0.001**	2.28	1.72 – 2.83	**<0.001**	2.91	2.11 – 3.72	**<0.001**
Petal.Width				1.02	0.72 – 1.32	**<0.001**	0.55	-0.42 – 1.52	0.267
Species [versicolor] * Petal.Width							1.32	0.23 – 2.42	**0.019**
Species [virginica] * Petal.Width							0.10	-0.94 – 1.14	0.848
Observations	150			150			150		
R^2 / R^2 adjusted	0.941 / 0.941			0.955 / 0.954			0.959 / 0.958		

Currently *tab_model* only support html output, which hopefully can be extended to latex in the future. *plot_model* from *jsplot* can be use to plot marginal effects of interaction models. More examples can be found here.

```
plot_model(mod3, type = "pred", terms = c("Petal.Width",
    "Species")) + theme_bw()
```



There are many factors you need to consider when displaying results, a few of them were listed here:

(1) Include point estimate, 95% CI and p-value (not starts) for regression models. The new recommendation is to report 3 digits if p-value < 0.05 otherwise 2 digits.

(2) Do not report p-value when hypothesis testing is meaningless. A perfect example is a correlation coefficient $\rho = 0.08$ with p-value< 0.001 (see example below).

```
set.seed(123)
a <- rnorm(1000)
b <- rnorm(1000)
cor.test(a, b)
```

```
##
##  Pearson's product-moment correlation
##
## data:  a and b
## t = 2.7423, df = 998, p-value = 0.006211
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.02461834 0.14768079
## sample estimates:
##       cor
## 0.08647944
```

(3) Include reference values. *tab_model* is currently having trouble to print out reference values. It is also difficult to configure custom settings and print LaTex table. The function below prints out linear regression model results and can be configured for different types of models and output requiements.

```
reg_table <- function(model, dta) {
    # Summarise coef
    Coef <- data.frame(coefficients(summary(model))) %>%
        mutate(Variables = rownames(.)) %>% as_tibble() %>%
        filter(Variables != "(Intercept)") %>%
        mutate(Coef = format(round(Estimate, digits = 2),
            nsmall = 2)) %>% mutate(`95% CI` = paste0(format(round(Estimate -
        1.96 * Std..Error, digits = 2), nsmall = 2),
        ", ", format(round(Estimate + 1.96 * Std..Error,
            digits = 2), nsmall = 2))) %>% mutate(`p-value` = Pr...t..) %>%
        mutate(`p-value` = ifelse(`p-value` <
            0.001, "<0.001", ifelse(`p-value` <
            0.05, format(round(`p-value`, digits = 3),
            nsmall = 3), format(round(`p-value`,
            digits = 2), nsmall = 2)))) %>% select(Variables,
        Coef, `95% CI`, `p-value`)

    # categorical variables for left col
    if (length(model$xlevels) > 0) {
        levels <- reshape2::melt(model$xlevels)
        levels <- levels %>% mutate(Variables = paste0(L1,
            value)) %>% group_by(L1) %>% mutate(level = seq(n()))
    } else {
        levels <- data.frame(value = NA, L1 = "",
            Variables = "", level = NA)
    }

    # extract variable labels
    varlabels <- as.data.frame(Hmisc::label(dta[,
        attr(model$terms, "term.labels")]))
```

```
    names(varlabels) <- "label"
    varlabels$L1 <- rownames(varlabels)
    varlabels <- varlabels %>% mutate(label = ifelse(is.na(label) |
        as.character(label) == "", L1, as.character(label))) %>%
        mutate(listvar = seq(n()))

    # prepare first column
    tablelabel <- data.frame(Variables = names(model$coefficients)) %>%
        filter(Variables != "(Intercept)") %>%
        mutate(Variables = as.character(Variables)) %>%
        full_join(levels) %>% mutate(L1 = ifelse(is.na(L1),
        as.character(Variables), as.character(L1))) %>%
        left_join(varlabels) %>% arrange(listvar,
        level) %>% dplyr::select(-listvar, -level,
        -L1) %>% group_by(label) %>% mutate(total = n(),
        order = seq(n())) %>% mutate(expend = ifelse(order ==
        1 & total > 1, 2, 1)) %>% uncount(expend) %>%
        mutate(order = seq(n())) %>% mutate(value = ifelse(is.na(value) |
        (total > 1 & order == 1), as.character(label),
        as.character(value))) %>% mutate(Variables = ifelse(total >
        1 & order == 1, NA, Variables)) %>% ungroup() %>%
        dplyr::select(Variables, value, total)

    # extract final table
    finaltable <- tablelabel %>% full_join(Coef) %>%
        mutate(Coef = ifelse(!is.na(Variables) &
            is.na(Coef), "Ref", Coef)) %>% replace(is.na(.),
        "")
    # extract which row for indentation and top
    # row name
    indent <- which(finaltable$Variables != "" &
        finaltable$total > 1)

    # print
```

```
    print(finaltable %>% select(-Variables, -total) %>%
        rename(Predictors = value) %>% kable(align = c("l",
        rep("c", 3)), booktabs = T, linesep = "") %>%
        kable_styling(bootstrap_options = "striped",
            full_width = F) %>% add_indent(indent))
}
```

```
Hmisc::label(iris$Petal.Length) <- "Petal length"
Hmisc::label(iris$Sepal.Width) <- "Sepal width"
Hmisc::label(iris$Sepal.Length) <- "Sepal length"
iris$Sepal.Length.Width <- iris$Sepal.Width *
    iris$Sepal.Length
Hmisc::label(iris$Sepal.Length.Width) <- "Sepal length * width"
mod <- lm(Petal.Width ~ Species + Petal.Length +
    Sepal.Width + Sepal.Length + Sepal.Length.Width,
    data = iris)
reg_table(mod, iris)
```

Predictors	Coef	95% CI	p-value
Species			
setosa	Ref		
versicolor	0.64	0.39, 0.88	<0.001
virginica	1.03	0.70, 1.36	<0.001
Petal length	0.24	0.15, 0.34	<0.001
Sepal width	0.10	-0.46, 0.65	0.73
Sepal length	-0.17	-0.45, 0.12	0.26
Sepal length * width	0.02	-0.07, 0.12	0.61

(3) Use table + Figure representation if you need/can

	No of counties	Relative risk (95% CI)	*p* -value	*p* -value for difference*
Literacy rate quartile				
Q1(lowest)	660	1.060 (1.042, 1.077)	<0.001	Ref
Q2	663	1.029 (1.014, 1.043)	<0.001	0.007
Q3	659	1.034 (1.019, 1.048)	<0.001	0.024
Q4(highest)	658	0.990 (0.966, 1.014)	0.405	<0.001
College rate quartile				
Q1(lowest)	660	1.044 (1.028, 1.060)	<0.001	Ref
Q2	660	1.054 (1.039, 1.068)	<0.001	0.368
Q3	660	1.032 (1.019, 1.046)	<0.001	0.301
Q4(highest)	660	1.009 (0.986, 1.033)	0.456	0.017
Urbanization rate quartile				
Q1(lowest)	660	1.035 (1.020, 1.051)	<0.001	Ref
Q2	660	1.046 (1.032, 1.061)	<0.001	0.315
Q3	660	1.034 (1.019, 1.049)	<0.001	0.870
Q4(highest)	660	1.008 (0.985, 1.031)	0.520	0.050
PGDP quartile				
Q1(lowest)	660	1.049 (1.027, 1.071)	<0.001	Ref
Q2	660	1.039 (1.019, 1.059)	<0.001	0.503
Q3	660	1.047 (1.028, 1.067)	<0.001	0.903
Q4(highest)	660	1.004 (0.985, 1.024)	0.676	0.002
Overall	2640	1.038 (1.030, 1.050)	<0.001	

Source:https://www.sciencedirect.com/science/article/pii/S0160412020321966#f0015

Advanced topics

Loops

Functions

Render anlaysis results from different dataset with the same rmarkdown file

Data visulisation

Reporting

One-stop shop

A good graph takes forever

Write up of the analysis results

- Report the nature and source of the data, validity of instrument and data collection process (i.e. response rate and any possible bias).
- Report any data editing procedures, including any imputation and missing data mechanisms

- When reporting analyses of volunteer data or other data that may not be representative of a defined population, includes appropriate disclaimers and, if used, appropriate weighting.
- Include the complete picture of the analysis results, which may require presenting tables and figures in appendix tables. For example, when reporting a series of multivariate regression models between an exposure and different outcomes, you can choose to include a summary table of adjust coef between exposure and different outcomes in the main text and include all the individual regression model results in the Appendix. The reader can use the appendix tables to understand the impact of confounding variables in the model.
- Report prevalence of outcomes or weighted prevalence of outcomes for representative samples.
- Report point estimate, 95% confidence interval and p-value in results
- Use graphical representations for reporting interaction effects (marginal plot)
- Acknowledges statistical and substantive assumptions made in the execution and interpretation of any analysis.
- Reports the limitations of statistical inference and possible sources of error.
- Where appropriate, addresses potential confounding variables not included in the study.
- Conveys the findings in ways that are meaningful and visually apparent and to the user/reader. This includes properly formatted tables and meaningful graphics (use guidelines by Gordon and Finch (2015)).
- To aid peer review and replication, shares the data (or synthetically generated data) used in the analyses whenever possible/allowable
- Provide all analysis code either as an Appendix or in open repositories such as Github

Advanced topics

Write a paper using R

Advanced Latex

Version control

Version control framework

Github

https://swcarpentry.github.io/git-novice/

Publication

Reference

Gordon, Ian, and Sue Finch. 2015. "Statistician Heal Thyself: Have We Lost the Plot?" *Journal of Computational and Graphical Statistics* 24 (4): 1210–29. https://minerva-access.unimelb.edu.au/bitstream/handle/11343/55491/Gordon%20and%20Finch%202014%20DOI%20version.pdf;jsessionid=557C01A51F9EC550925E94F564ED970A?sequence=1.

Perperoglou, Aris, Willi Sauerbrei, Michal Abrahamowicz, and Matthias Schmid. 2019. "A Review of Spline Function Procedures in R." *BMC Medical Research Methodology* 19 (1): 1–16.

Williamson, Tyler, Misha Eliasziw, and Gordon Hilton Fick. 2013. "Log-Binomial Models: Exploring Failed Convergence." *Emerging Themes in Epidemiology* 10 (1): 1–10. https://doi.org/https://doi.org/10.1186/1742-7622-10-14.