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

Caroline X. Gao [1], Matthew Hamilton [2]

[1] University of Melbourne, Parkview, VIC, Australia

[2] Orygen, Parkview, VIC, Australia

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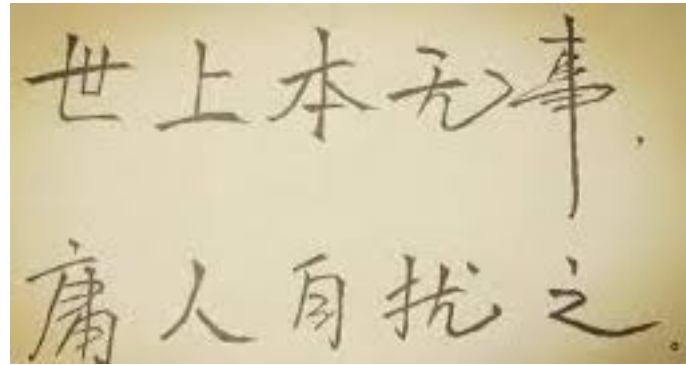
The famous quote is not by Confucius (it is from the New Book of Tang published about 1500 years after his death), and it is not well translated, but you get the idea. We often find it easier to face a complex problem than a simple one, and often feel that a complex solution must be better than an easier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

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 :)

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

```
##      version                date              nickname
## 115    3.6.2 2019-12-12 08:05:03   Dark and Stormy Night
## 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
```

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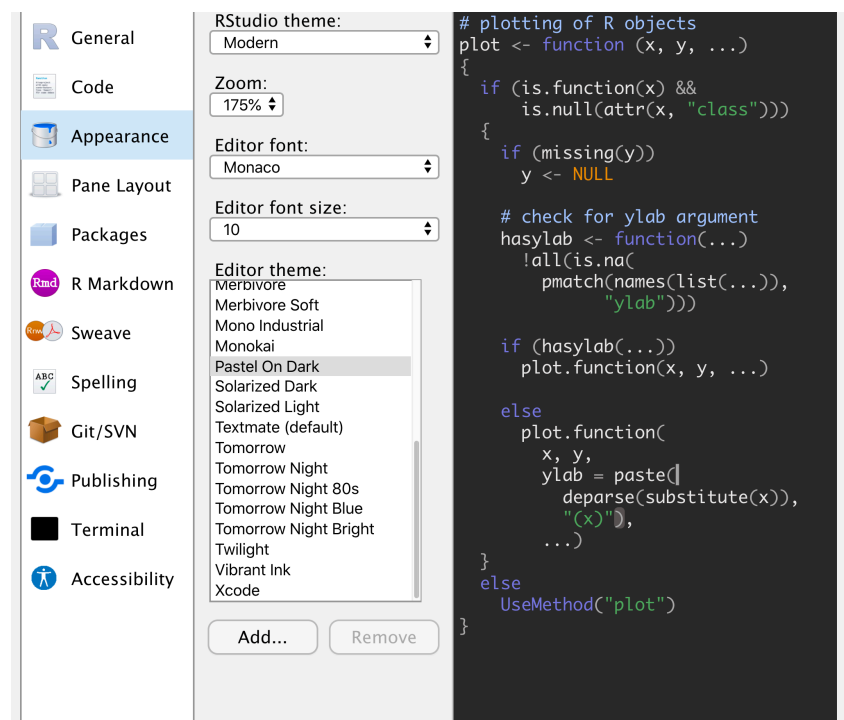
I also like to turn on a few diagnostics including "Check usage of <- in function call" and "Provide R style diagnostics" (make sure you have a good R coding style, see style guide here

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*.

```r
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_0.19         haven_2.3.1       colorspace_2.0-0
##  [5] vctrs_0.3.5       generics_0.1.0    htmltools_0.5.0   viridisLite_0.3.0
##  [9] yaml_2.2.1        rlang_0.4.9       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      evaluate_0.14
## [25] curl_4.3          fansi_0.4.1      broom_0.7.2      Rcpp_1.0.5
## [29] formatR_1.7       scales_1.1.1     backports_1.2.1  webshot_0.5.2
## [33] jsonlite_1.7.2    fs_1.5.0         hms_0.5.3        digest_0.6.27
## [37] stringi_1.5.3     bookdown_0.21    rprojroot_2.0.2  grid_4.0.2
## [41] here_0.1          cli_2.2.0        tools_4.0.2      magrittr_2.0.1
## [45] crayon_1.3.4      pkgconfig_2.0.3  ellipsis_0.3.1   xml2_1.3.2
## [49] reprex_0.3.0      lubridate_1.7.9.2 assertthat_0.2.1 rmarkdown_2.5.3
## [53] httr_1.4.2        rstudioapi_0.13  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.

```r
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.

THE ART OF PROGRAMMING – PART 2: KISS
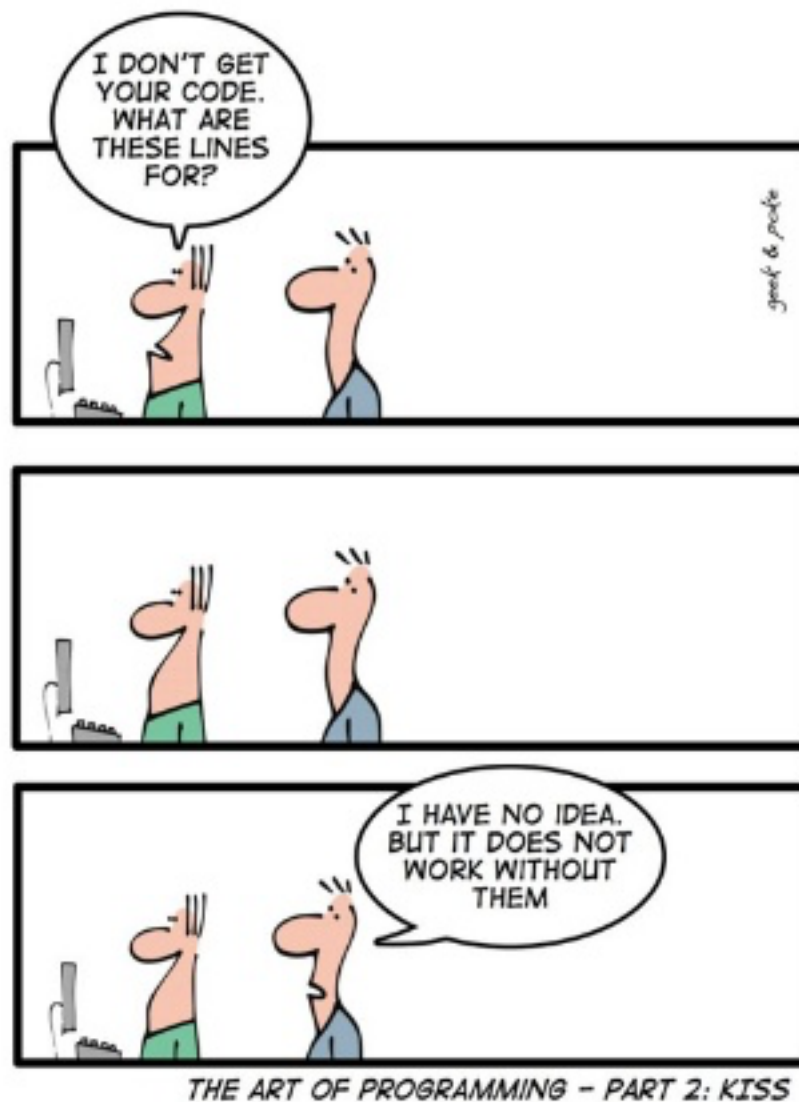
However, the sudden boom of data science brought in new challenges. As software engineers move towards data science, the community gradually realised that the traditional coding style (i.e. modular programming, black-box approach) became barriers instead of advantages. Each step of the programming code, 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.
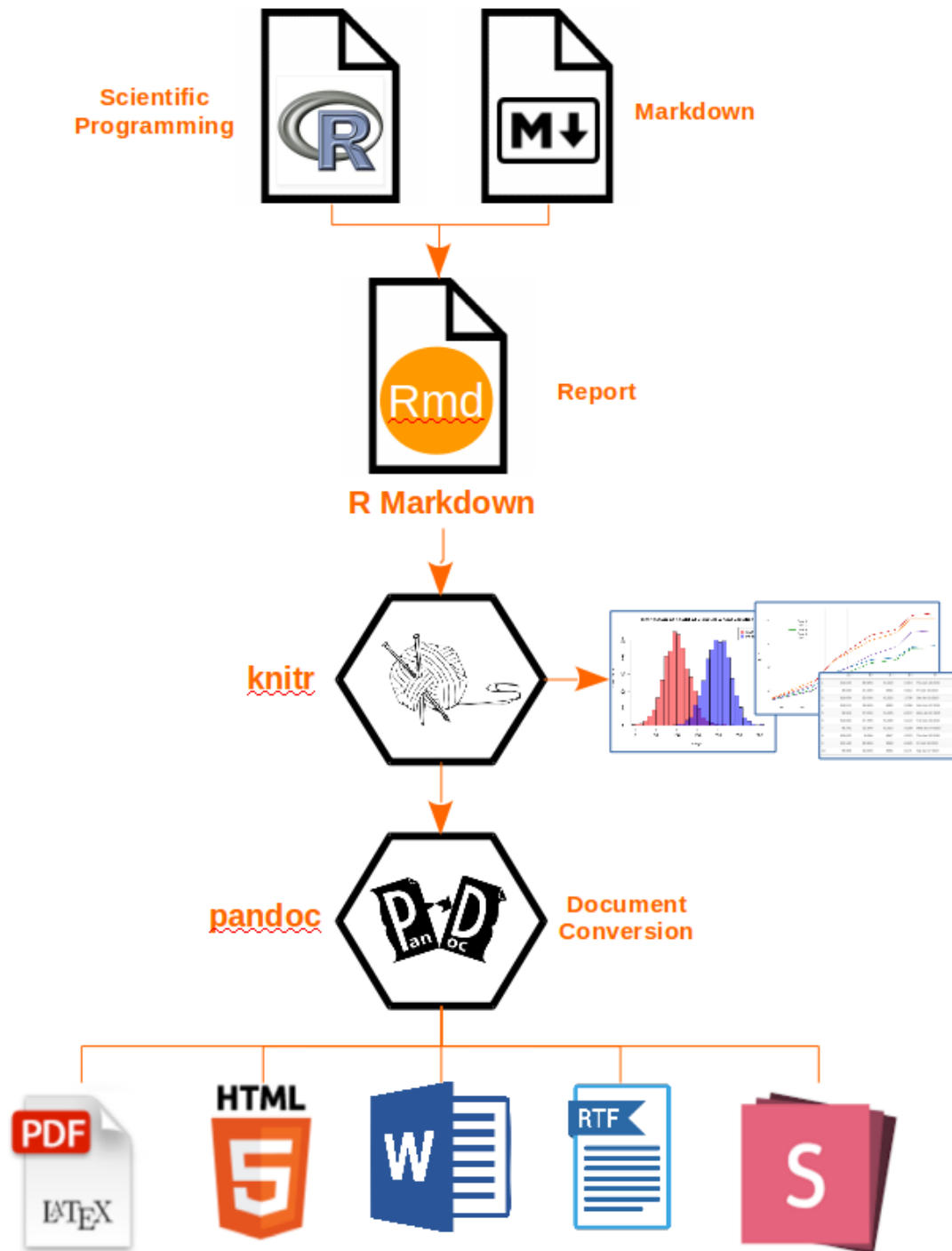
R markdown is a system that integrate narrative, analysis, code and output to create a production quality document. The code and text were implemented via R in a *.Rmd file, which will be knit to Markdown file and then convert to file type of choice via Pandoc. There are a few important features that 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:
- familyname: Gao
  othernames: Caroline X.
  address: Centre for Youth Mental Health, University of Melbourne;
  School of Public Health and Preventive Medicine, Monash University
  email: caroline.gao@orygen.org.au
```

```
- familyname: Hamilton
  othernames: Matthew
  address: Orygen, Melbourne, Australia
output:
  bookdown::pdf_document2:
    toc: yes
    number_sections: false
    fig_caption: no
    pandoc_args:
      - --template=template.tex
  bookdown::word_document2:
    toc: yes
    number_sections: false
    fig_caption: no
    reference_docx: "template.docx"
date: "`r format(Sys.time(), '%B %d, %Y')`"
geometry: margin=1in
fontsize: 11pt
bibliography: GSP.bib
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.
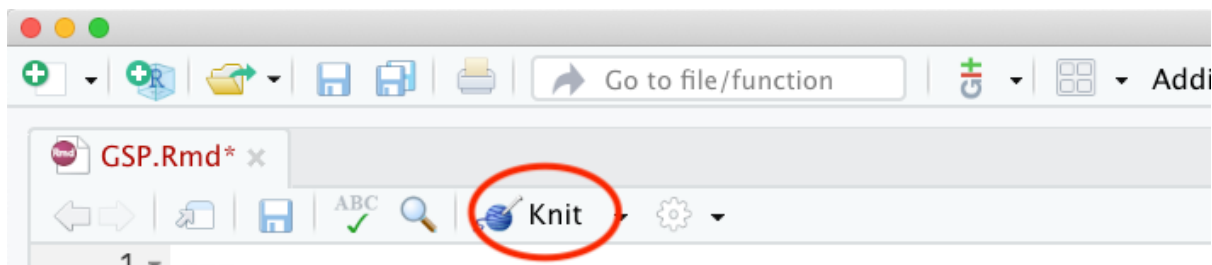
You also need to install LaTex (https://www.latex-project.org/get/). However, LaTex is really a large monster to work with. So Yihui Xie has developed TinyTeX for R users, which is much smaller. You can install additional LaTex packages easily if needed.

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

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. As I normally use mutating joins from *tidyverse*. An important thing to check is whether the *by* variable(s) can uniquely define a records before merging the data. If not, joins will create all possible pairs, so you are at risk of 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/68440762925952614
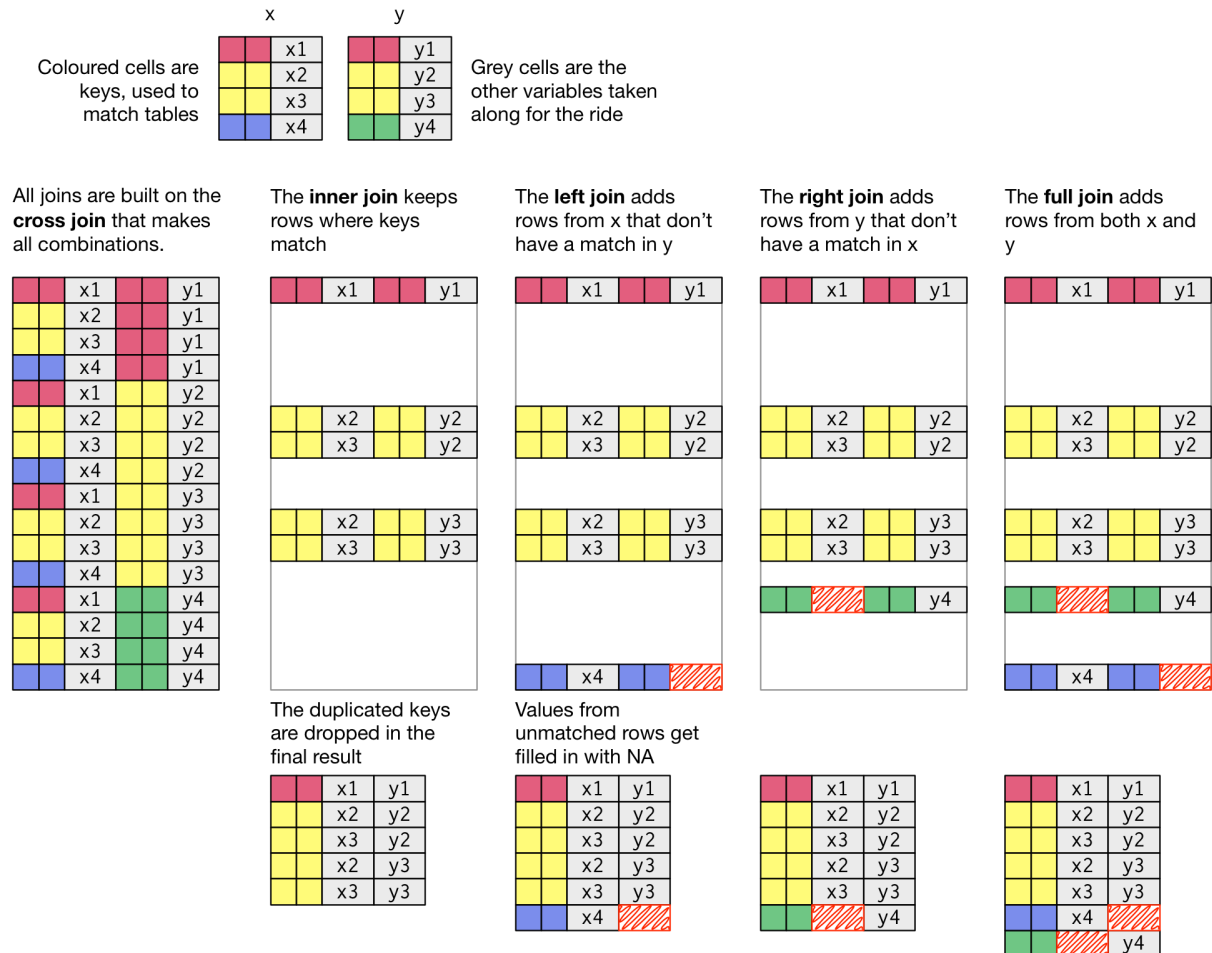
One important benefit of using R is that it allows you to open many datasets and process them simultaneously. This is the major difference compared with working with SPSS and Stata which allows one data file at a time.

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.

```r
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?
- Are factor variables stored as character of numeric ?
- Is the data file long or wide (longitudinal)?
- How is missing data coded?
- What's the proportion of missing in different variables?
- Are date variables correct?

There are a few functions that are quite useful:

```
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
```
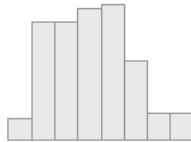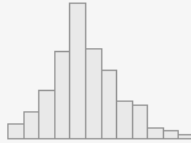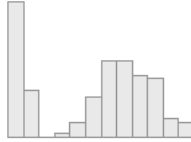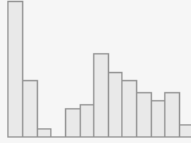
```
## --------------------------------------------------------------------------
```

```
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

Tidy-version data cleaning routine

After the data is properly organised, you will need to start

Notes for yourself and others

Validity checking

Common pitfalls

Documentation for the never ending data cleaning process

Analysis

Exploratory phase

Statistical modeling with R

Extract results

Advanced topics

Loops

Functions

Render anlaysis results from different dataset with the same rmarkdown file

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.