Reproducible statistical analysis with R

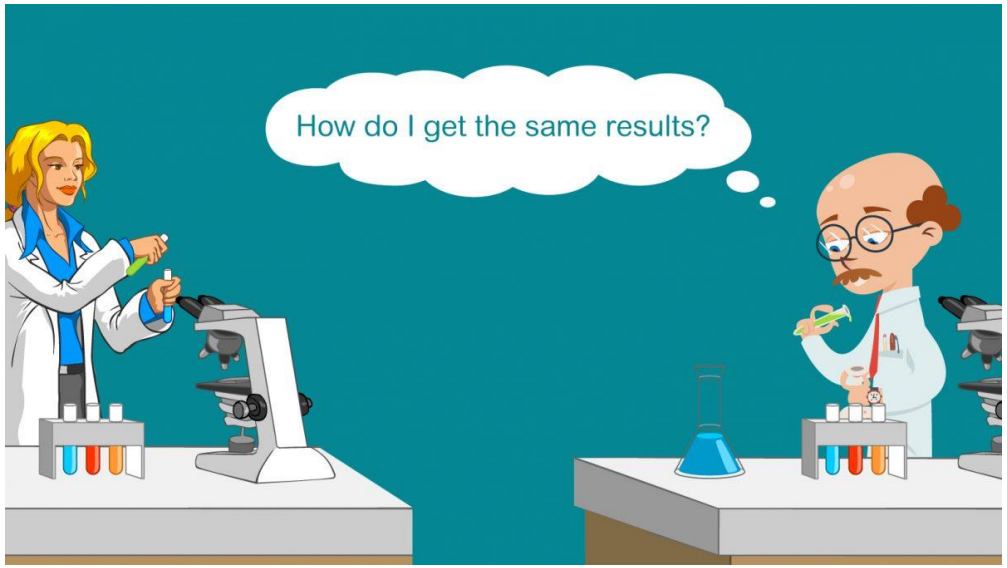
David Jorgensen

Adapted from article by Amy Gimma and Thibaut Jombart

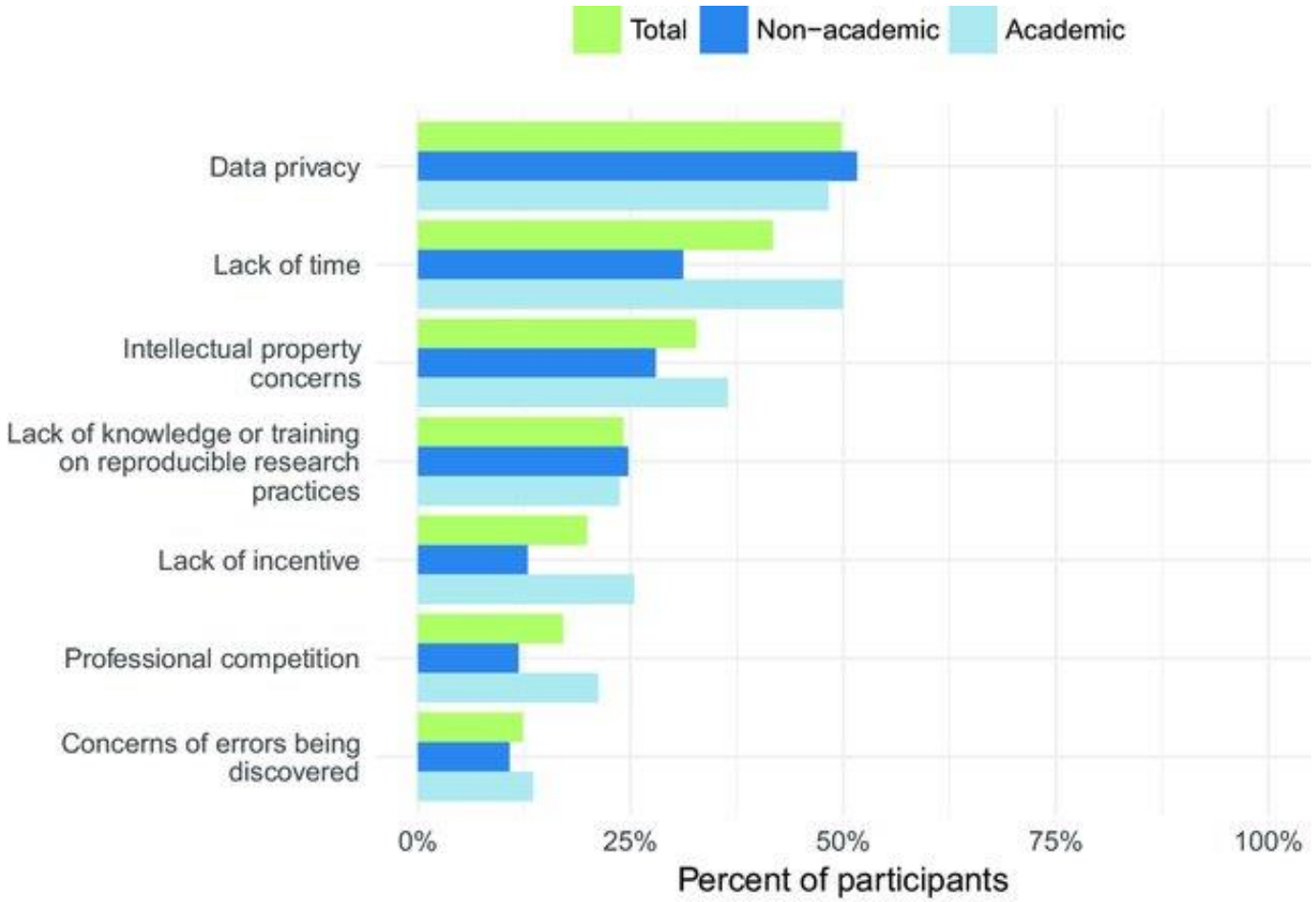
Reproducibility

- Ability to reproduce results by a peer (or yourself in the future)

- Requires <u>data</u>, <u>methods</u> and <u>procedures</u>



Barriers to reproducibility



Harris, J. K., Johnson, K. J., Carothers, B. J., Combs, T. B., Luke, D. A., & Wang, X. (2018, September 1). Use of reproducible research practices in public health: A survey of public health analysts. *PLoS ONE*, Vol. 13. https://doi.org/10.1371/journal.pone.0202447

Barriers to reproducibility

Although data privacy is a valid concern in public health, many of the other factors are readily overcome:

- Lack of time – ultimately faster for repeated analyses
- Fear of plagiarism – unlikely in practice
- Internal work with no need to share – reproducibility is still useful for yourself and colleagues

R is able to help overcome one of the main concerns – lack of tools to support reproducibility

Two aspects of reproducibility using R

- Implementing methods as R packages
- Making transparent and reproducible analyses

Literate programming

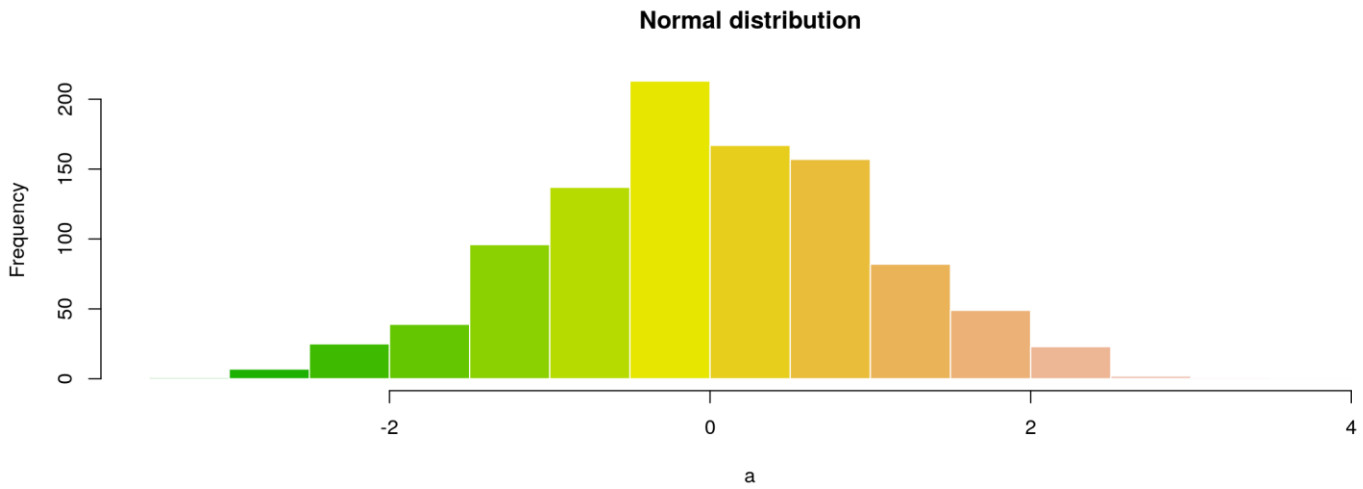
Let us change our traditional attitude to the construction of programs:
instead of imagining that our main task is to instruct a computer what to do,
let us concentrate rather on explaining to humans what we want the computer to do.
(Donald E. Knuth, Literate Programming, 1984)

Rmarkdown: R chunks in markdown

```{r chunk-title, ...}
a <- rnorm(1000) hist(a, col = terrain.colors(15), border = "white", main = "Normal distribution")
```

Results in:

```r
a <- rnorm(1000)
hist(a, col = terrain.colors(15), border = "white", main = "Normal distribution")
```



Formatting outputs

```
```{r another-chunk-title, ...}
[some R code here]
```
```

where `...` are options for processing and formatting, e.g:

- `eval` (`TRUE`/`FALSE`): evaluate code?
- `echo` (`TRUE`/`FALSE`): show code input?
- `results` (`"markup"`/`"hide"`/`"asis"`): show/format code output
- `message/warning/error`: show messages, warnings, errors?
- `cache` (`TRUE`/`FALSE`): cache analyses?

See http://yihui.name/knitr/options for details on all options.

One format, several outputs

`rmarkdown` can generate different types of documents:

- standardised reports (`html`, `pdf`)
- journal articles. using the `rticles` package (`.pdf`)
- handouts (`.pdf`)
- word documents (`.doc`)
- slides for presentations (`html`, `pdf`)
- …

See: http://rmarkdown.rstudio.com/gallery.html.

Rules for reproducible statistical analysis in R

1 Organise R script logically

1. Data preparation

We recommend the following structure:

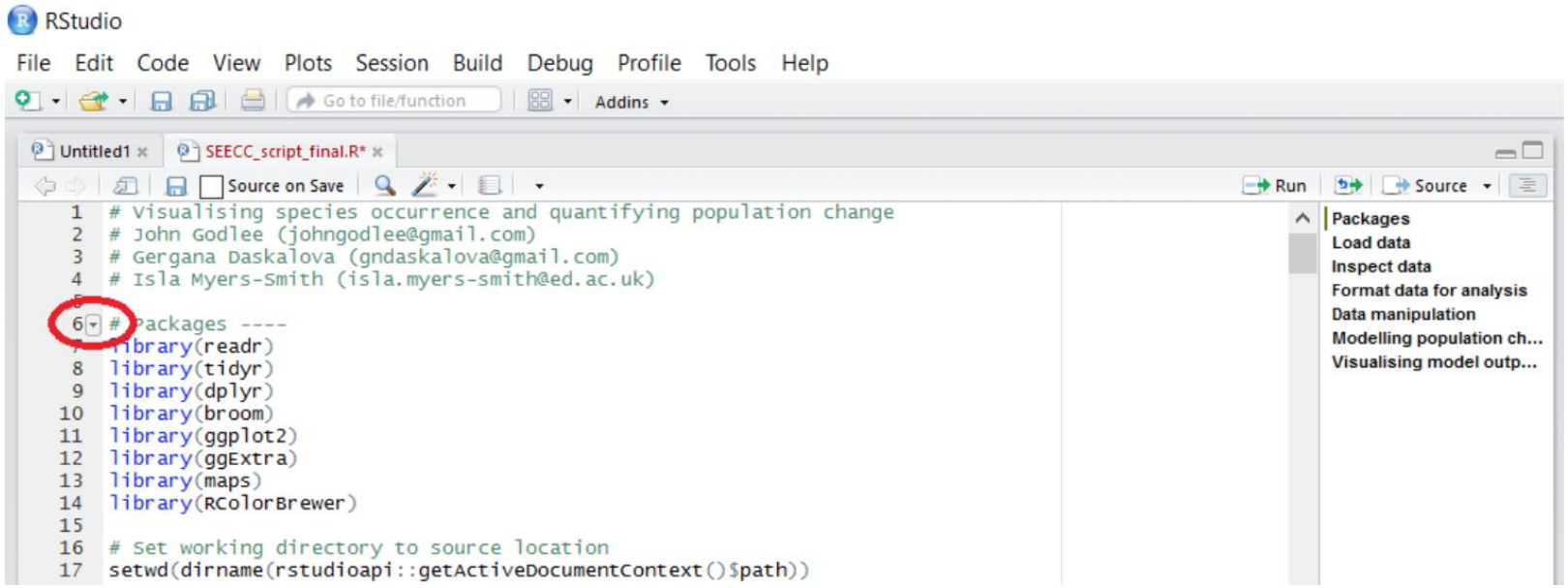
- **Load packages needed**
- **Load the raw data**
- **Clean the raw data**
  - *standardise data* (e.g. `linelist::clean_data`)
  - *convert dates* that need converting (e.g. `linelist::guess_dates`)
  - *fix typos* (e.g. `linelist::clean_variable_spelling`)
- **Add new variables** (e.g. using `mutate`)
- **Subset entries** (rows) of the data (e.g. using `filter`)
- **Define custom colors** (e.g. using `scale_fill_manual` or `scale_color_manual`)

1 Organise R script logically

2. Data analysis

Organise work systematically

- **Sub-sections** for types of analysis
- General analyses **before** subsets or stratified analyses



1 Organise R script logically

3. Export outputs

If you want to come back to work in the future or use other software on the cleaned data it is important to export your outputs.

- `.rds` files for future R analysis
- `.xlsx` or `.csv` files for analysis with other software

1 Organise R script logically

4. System info

This is optional, but can be useful for audit purposes, or for diagnosing issues in the results generated. We recommend including the following:

- `Sys.info()`: basic system information
- `R.version`: version of **R**
- `sessionInfo()`: which packages are loaded, and which versions are they?
- `params`: this list will contain optional parameters passed at compilation time through the `params` argument of `compile_report` or `update_reports`

2 Naming convention recommendations

- **Use only** lower case letters, numbers and _ as separator (use – for dates)
- **Never use** special characters in file or variable names (e.g. éÈçôï\/# %?!&:;,@*^) or blank spaces
- If you need special characters for plots **only use when defining labels**
- Standard date format in R is **yyyy-mm-dd**

`linelist::clean_data` will do this for you on a `data.frame` or `tibble`

```
messy <- linelist::messy_data(n = 10) %>% select(1:5)
messy
##       'ID Date of Onset. DisCharge.. GENDER_  Épi.Case_définition
## 1  zhpmio     2018-01-10  20/01/2018   Female         suspected
## 2  sckroc     2018-01-04  14/01/2018     Male          not a case
## 3  snwdom     2018-01-11  21/01/2018   female         suspected
## 4  kheaqv     2018-01-08  18/01/2018     MALE         suspected
## 5  papdba     2018-01-11  21/01/2018   Female         suspected
## 6  pypdsv     2018-01-02  12/01/2018     MALE         suspected
## 7  cyuiex     2018-01-05  15/01/2018   Female         suspected
## 8  koexkj     2018-01-08  18/01/2018     Male           Confirmed
## 9  u
## 10 n
```

```
clean <- clean_data(messy)
clean
##        id date_of_onset  discharge gender epi_case_definition
## 1  zhpmio    2018-01-10 2018-01-20 female           suspected
## 2  sckroc    2018-01-04 2018-01-14   male          not_a_case
## 3  snwdom    2018-01-11 2018-01-21 female           suspected
## 4  kheaqv    2018-01-08 2018-01-18   male           suspected
## 5  papdba    2018-01-11 2018-01-21 female           suspected
## 6  pypdsv    2018-01-02 2018-01-12   male           suspected
## 7  cyuiex    2018-01-05 2018-01-15 female           suspected
## 8  koexkj    2018-01-08 2018-01-18   male           confirmed
## 9  uuljge    2018-01-02 2018-01-12 female          not_a_case
## 10 nuqiit    2018-01-02 2018-01-12   male            probable
```

Include special characters in plot *without* renaming the variables

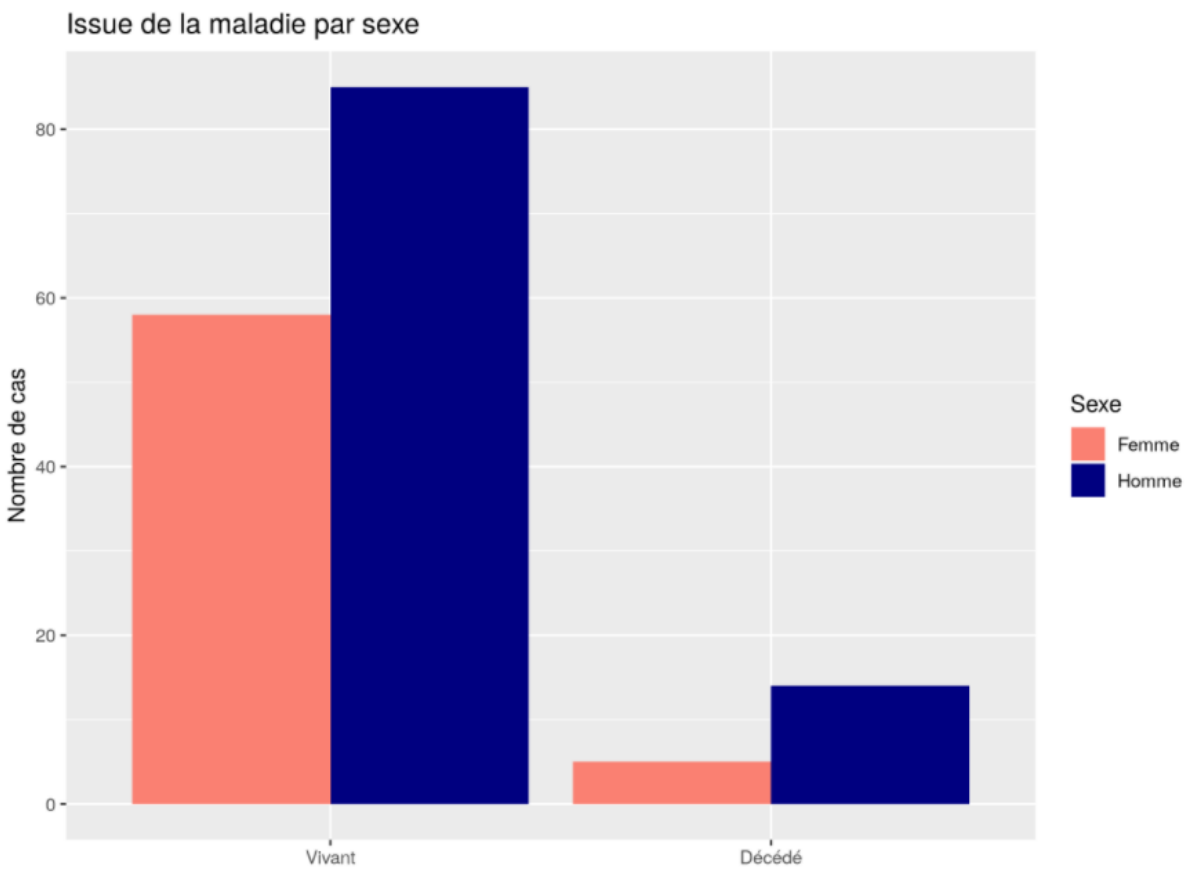
```r
## load and clean data
mers <- outbreaks::mers_korea_2015$linelist %>%
  as_tibble() %>%
  clean_data()

mers
## # A tibble: 162 x 15
##    id      age age_class sex   place_infect reporting_ctry loc_hosp
##    <chr> <int> <chr>     <fct> <fct>        <fct>          <fct>
##  1 sk_1     68 60_69     m     middle_east  south_korea    pyeongt…
##  2 sk_2     63 60_69     f     outside_mid… south_korea    pyeongt…
##  3 sk_3     76 70_79     m     outside_mid… south_korea    pyeongt…
##  4 sk_4     46 40_49     f     outside_mid… south_korea    pyeongt…
##  5 sk_5     50 50_59     m     outside_mid… south_korea    365_yeo…
##  6 sk_6     71 70_79     m     outside_mid… south_korea    pyeongt…
##  7 sk_7     28 20_29     f     outside_mid… south_korea    pyeongt…
##  8 sk_8     46 40_49     f     outside_mid… south_korea    seoul_c…
##  9 sk_9     56 50_59     m     outside_mid… south_korea    pyeongt…
## 10 sk_10    44 40_49     m     outside_mid… china          pyeongt…
## # … with 152 more rows, and 8 more variables: dt_onset <date>,
## #   dt_report <date>, week_report <fct>, dt_start_exp <date>,
## #   dt_end_exp <date>, dt_diag <date>, outcome <fct>, dt_death <date>

## define scales for sex and outcome, tweak labels as appropriate
scale_sex <- scale_fill_manual(
    "Sexe",
    values = c(m = "navy", f = "salmon"),
    labels = c(m = "Homme", f = "Femme"))

scale_x_outcome <- scale_x_discrete(
    labels = c(alive = "Vivant",
               dead = "Décédé"))

## make the plot
ggplot(mers, aes(x = outcome, fill = sex)) +
  geom_bar(position = "dodge") +
  scale_sex +
  scale_x_outcome +
  labs(title = "Issue de la maladie par sexe",
       x = "",
       y = "Nombre de cas")
```



3 Use descriptive naming

Clarity is improved by giving descriptive names to files and variables

```
Jennifers-MacBook-Pro-3:analysis jenny$ ls -l

01_marshal-data.md                          01.md
01_marshal-data.r                           01.r
02_pre-dea-filtering.md                     02.md
02_pre-dea-filtering.r                      02.r
03_dea-with-limma-voom.md                   03.md
03_dea-with-limma-voom.r                    03.r
04_explore-dea-results.md                   04.md
04_explore-dea-results.r                    04.r
90_limma-model-term-name-fiasco.md          90.md
90_limma-model-term-name-fiasco.r           90.r
Makefile                                    Makefile
figure                                      figure
helper01_load-counts.r                      helper01.r
helper02_load-exp-des.r                     helper02.r
helper03_load-focus-statinf.r               helper03.r
helper04_extract-and-tidy.r                 helper04.r
tmp.txt                                     tmp.txt
```

4 Simple, readable code

```r
lapply(iris_clean[iris_clean$species%in%c("setosa","versicolor"),grep("sepal",names(iris_clean))],summary)
```

- Write short lines
- Break up complex code
- Describe code – readme and inline comments
- ## for comments – better if you want to use code editors other than Rstudio

```r
## find rows to keep: species setosa and versicolor
rows_to_keep <- iris_clean$species %in% c("setosa", "versicolor")

## identify columns with 'sepal' in their name
sepal_columns <- grep("sepal", names(iris_clean))

## subset data to analyse
sepals_setosa_versicolor <- iris_clean[rows_to_keep, sepal_columns]

## get summaries
lapply(sepals_setosa_versicolor, summary)
```

Lintr (optional)

Rstudio plugin which can tell you where your code differs from common style conventions

github.com/jimhester/lintr

bad.R

```
1  fun = function(one)
2  {
3    one.plus.one <- oen + 1
4    four <- newVar <- matrix(1:10,nrow = 2)
5    four[ 1, ]
6    txt <- 'hi'
7    three <- two+ 1
8    if(txt == 'hi') 4
9    5}  
10 {
11 
```

6:14 fun(one) R Script

Console Markers

lintr

~/Dropbox/projects/lintr/bad.R

```
Line 1  Use <-, not =, for assignment.
Line 2  Opening curly braces should never go on their own line and should always be followed by a new line.
Line 3  Words within variable and function names should be separated by '_' rather than '.'.
Line 3  local variable ‘one.plus.one’ assigned but may not be used
Line 3  no visible binding for global variable ‘oen’, Did you mean 'one'?
Line 4  Variable and function names should be all lowercase.
Line 4  local variable ‘newVar’ assigned but may not be used
Line 4  Commas should always have a space after.
Line 5  Do not place spaces around code in parentheses or square brackets.
Line 6  Only use double-quotes.
Line 7  local variable ‘three’ assigned but may not be used
Line 7  no visible binding for global variable ‘two’, Did you mean 'txt'?
Line 7  Put spaces around all infix operators.
Line 8  Place a space before left parenthesis, except in a function call.
Line 8  Only use double-quotes.
Line 9  Closing curly-braces should always be on their own line, unless it's followed by an else.
Line 9  Trailing whitespace is superfluous.
Line 10 unexpected end of input
```

5 Avoid messy projects

- 1 project = 1 folder
- Subfolders for data, analyses, figures, manuscripts etc.
- Use Rstudio projects (as in our case study)
- Document projects with README file and inline comments

6 Relative file paths

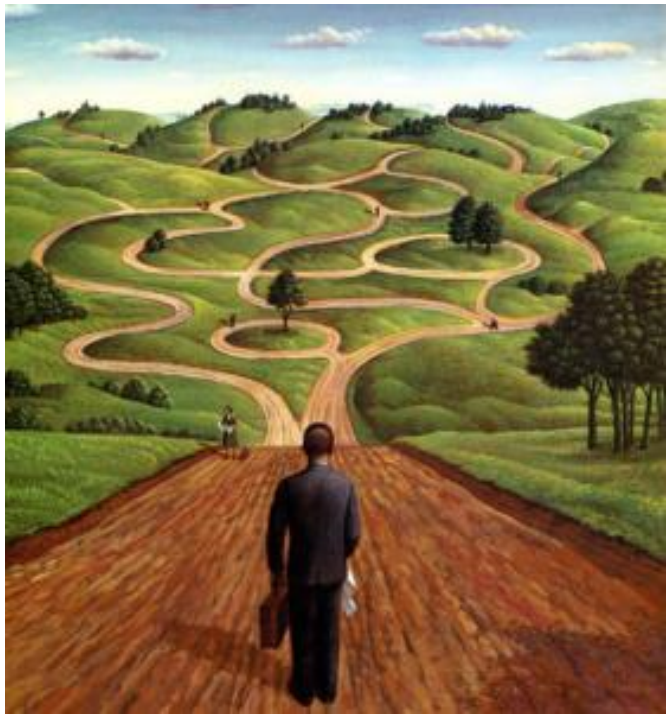
X `my_file <- 'C:\project1\data\data.csv'`

✓ `setwd('C:\project1') ## project does this`
`my_file <- './data/data.csv'`

✓✓ `my_file <- here('data/data.csv')`

Using an **R project** automatically sets your working directory to the project directory meaning you can share code more efficiently

here works from the project directory even if you open an R script directly rather than via the project. It also translates between windows and unix file structure (good for sharing)



7 Avoid losing work

- **Never rely on a single copy of work**
- Backups are good, syncing with a server is better (dropbox, sharepoint etc.)
- Use versions to keep track of changes
- For serious coding projects use version control – e.g. git and github