Practical: Tidying

BaselRBootcamp 2017

Slides

Here a link to the lecture slides for this session: **LINK** (https://therbootcamp.github.io/_sessions/D3S1_Tidying/Tidying.html)

Overview

In this practical you'll learn how to produce **tidy** code (and data). By the end of this practical you will know how to:

1. Write clean, documented code.
2. Understand errors and warnings.
3. Deal with missing values.

The Do's and Don'ts of clean code

Filenames should be meaningful. To order them, prefix them with numbers.

```
# Good
analyze_my_data.R
0_read_my_data.R
1_analyze_my_data.R

# Bad
stuff.r
code.r
```

Object names should be lowercase. Use `_` rather than `.` or 'CamelCase' (using capitalization) for multi-word names. If possible use nouns for variables and verbs for functions. Use meaningful names. **Avoid using names of existing objects**.

```
# Good
trial_id
trial_1

# Bad
name_of_trial
trialID
tid
t1
```

Place **spaces** around all operators, such as, `=`, `+`, `-`, `<-`, etc. Also applies for defining arguments in functions. Always put a space after, never before a comma.

```
# Good
var_rt <- var(rt, na.rm = TRUE)

# Bad
var_rt<-var(rt,na.rm=TRUE)
```

Extra **spacing** may be used to align assignments.

```
# Good
list(
  var_rt  = var(rt)
  mean_rt = mean(rt)
)
```

An opening **curly bracket** should never be on its own line. Always indent within curly brackets. To **indent** code use two spaces. Don't use tabs.

```
# Good
if (my_dbl < 2){
  message('my_dbl is smaller 2')
} else {
  message('my_dbl is larger or equal 2')
}

# Bad
if (my_dbl < 2)
{
message('my_dbl is smaller 2')
} else {
message('my_dbl is smaller 2')
}
```

For **assignments** use `<-` , not `=` .

```
# Good
x <- 24324

# Bad
x = 24324
```

Comment each line of your code. To break up your code in chunks use `-` or `=` .

```
# Plot data ----------------------------

# Plot data ============================
```

The most frequent errors

R's error messages are not always very intuitive, but over time you will learn to understand them. In the beginning it helps to focus on the part after the colon. E.g.,

```
sapply(1:10,'fefkl')
```

```
Error in get(as.character(FUN), mode = "function", envir = envir): object 'fefkl' of
 mode 'function' was not found
```

Here the key message is `'function' was not found` . I.e., R is interpreting `'fekl'` as a function but cannot find an instance of this function anywhere (because it doesn't exist).

According to an analysis of stackoverflow.com (https://stackoverflow.com/), a popular help forum, the **7 most frequent error messages** and their meaning are:

Error	Example	Description
`'could not find function'`	lenth(my_vec)	There is a typo in the function name or that a package has not been loaded.
`'error in if'`	if(NA == 2) 2 + 2	The object in the `if` clause is non-logical or NA.
`'error in eval'`	lm(fefq~wzfe)	An object is used that does not exist.
`'cannot open()'`	read_csv('hjht.txt')	The file does not exist. Could be a typo or a missing filepath.
`'no applicable method'`	predict('efwe')	A 'generic function' has not been defined for this type/class
`'subsscript out of bounds'`	a <- matrix(c(1,2)); a[2,2]	R tried to access an element (or variable) that does not exist
package errors		Occur when R is unable to install, compile, or load a package. Often this means that some software background is missing.

For more information visit here (https://github.com/noamross/zero-dependency-problems/blob/master/misc/stack-overflow-common-r-errors.md).

A `mice` example

To impute missing values, the `mice` package is very helpful. The code below loads the titanic dataset containing records on 1313 Titanic survivors and then attempts to predict missing values in Age using central tendencies and `mice`.

```
# Load packages
library(readr)
library(mice)

# read and duplicate in titanic data
data <- read_csv('https://therbootcamp.github.io/_slides/data/titanic.csv')
data_mean <- data
data_median <- data
data_mice <- data

# use central tendencies
data_mean$Age[is.na(data_mean$Age)] <- mean(data_mean$Age, na.rm = TRUE)
data_median$Age[is.na(data_median$Age)] <- median(data_median$Age, na.rm = TRUE)

# use mice
mice_model <- mice(data_mice, method = 'rf') # uses random forests
data_mice  <- complete(mice_model)
```

Tasks

Begin new project

Begin a new project in a new folder. Within the folder, create two new folders called `1_data` and `2_code`.

Clean code

1. Go through the code below and clean it, according to the above principles (incl. commenting). Then download all of the data files referenced in the code, store them in `1_data`, and change the filepaths in the code to this location. When ready save the code in the `2_code` folder as `cleaned_code_2017Sep16.R`.

```
library(readr   )
library(      magrittr)
library(  dplyr)
  syc =   read_csv('https://therbootcamp.github.io/_slides/data/galton.txt')
  syc = syc%>%mutate(fcm=father/2.54,mcm=mother/2.54)
    a   = syc$father
      b = syc[['mother']]
        t.test(a,b)
```

2. Start again with the 'dirty' code above in a new script. Save it in your `2_code` folder. Now install and load the `formatR` package and try to clean up the 'dirty code' using the `tidy_source()` function. What did it fix and what didn't it?

Correct code

3. Go through the code below, clean it, and remove the errors. When ready save the code in the `2_code` folder as `corrected_code_2017Sep16.R`.

```
library(readr   )
library(magritt)
syc =   read_csv('1data/galton')
syc %>% mutate(higher_ratio = height / fther)
m=lm(height~father,data=syk)
yhat=predict(M)
plot(syc[[father]],yhat)
vs = syc %>% group_by(father) %>% summarize(a = meaen(height))
points(vs,pch=16)
```

Replace NAs

4. Run 'A `mice` example' from above and evaluate the mean and variance (`var()`) of the variable `Age` as a function of mean and mice imputation, respectively. Are they different? Which imputation method produces the larger variance and why?

5. Evaluate the performance of mean versus mice imputation. To do this, first load the galton data set, copy it, and remove 10% of the values in the variable `height` in the copy. Now create one new data frame for each method and apply to the mean and mice imputation. Finally, compare the mean squared error `mean((original - imputed)^2)` between the imputed and original values. Which method works better?

6. Repeat the analysis of the last exercise but this time introduce NAs systematically rather than randomly. That is use other variable or combination of other variables to determine the locations of NAs in the `height` variable. For instance, introduce NAs for `father > 70` and the `sex == 'M'`. Have the results changed? If yes, why?

7. The mice package offers a variety of different methods. See `?mice` . Try out different methods and compare their performance.

Additional reading

- For more details check out check out Hadley Wickham's Advanced R (http://adv-r.had.co.nz/).
- Google style guide (https://google.github.io/styleguide/Rguide.xml)