Tidying

The R Bootcamp
Twitter: @therbootcamp

September 2017

Tidying

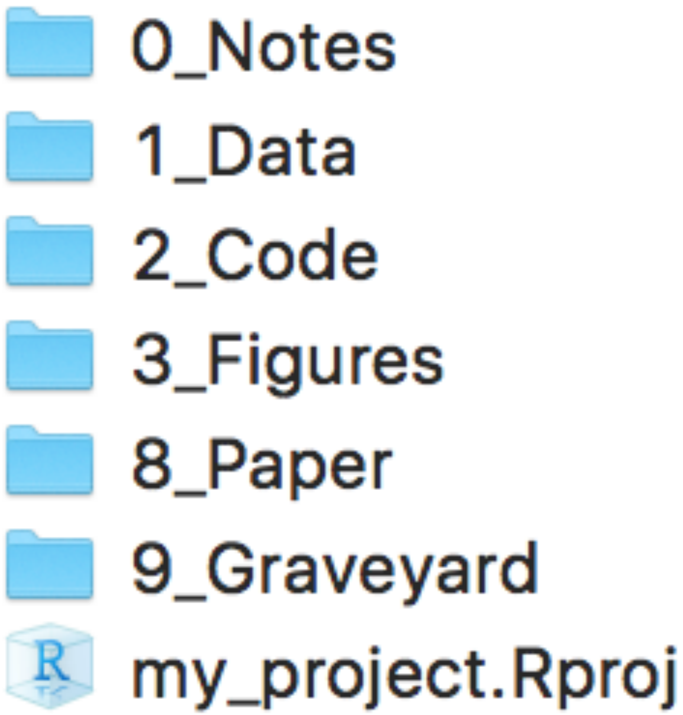
In this introduction you will learn…

> …how to write clean, documented code.
>
> …to understand errors (and warnings).
>
> …how to deal with missing values.

source https://build2be.com/

Project structure

Good, clean, documented code begins with a **project** and a **folder structure**.



0_Notes
1_Data
2_Code
3_Figures
8_Paper
9_Graveyard
my_project.Rproj

Naming files

Filenames should be **meaningful**. If you like, **order them by prefixing** them with numbers.

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

# Bad
stuff.r
code.r
```

Naming objects

- **Object names** should be lowercase.
- Use `_` rather than `.` or 'camelCase' (using capitalization) for multi-word names.
- 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
```

Spaces

- Place **spaces** around all operators, e.g., =, +, −, <−, etc. Also applies for defining arguments in functions.
- Always put a space **after a comma**, never before.
- Extra **spacing** may be used to align assignments.

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

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

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

Curly brackets

- 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')
}
```

Assignments & Comments

For **assignments** use <-, not =. However, to specify arguments in functions use =.

```
# Good
x <- 24324

# Bad
x = 24324
```

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

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

this_is <- "pseduo_code"
my_function(arg1 = x,
            arg2 = y)

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

this_is <- "pseduo_code"
```

Errors, Warnings, Messages

R has different categories for telling you something has happened depending on the severity of the event.

Errors indicate that **something bad** has happened. Errors always stop the code.

```r
# Error
stop('This is an error') ; men(c(1, 2, 3))
```

```
## Error in eval(expr, envir, enclos): This is an error
```

```
## Error in men(c(1, 2, 3)): could not find function "men"
```

Warnings indicate that **something potentially worrying** has happened. Warnings do not stop the code.

```r
# Warning
warning('This is an error') #; c(1, 2) + c(2, 3, 4)
```

```
## Warning: This is an error
```

Messages indicate that **something noteworthy** has happened, e.g., completition of an analysis step.

```r
# Message
message('This is a message')
```

```
## This is a message
```

7 most frequent errors

According to stackoverflow.com

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.

Missing data

A pervasive problem in working with data is missing values.

In R there are **two kinds of missing values**: the more general and frequent NA, and the more specific NaN.

```
# NA and NaN
my_vec <- c(1,2) ; my_vec[5]
```

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

```
0/0
```

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

```
# Tests
is.na(my_vec[5]) ; is.na(0/0)
```

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

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

```
is.nan(my_vec[5]) ; is.nan(0/0)
```

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

Handling missing data

Many functions have **inbuilt handlers** for missing data.

In most cases, however, missing values have to and should be dealt with **before the analysis**.

```
# Example
my_vec_1 <- c(1, 2, 3,  4, NA)
my_vec_2 <- c(4, 2, NA, 3, 5)

# Functions examples that include handlers
mean(my_vec_1) ; cor(my_vec_1,my_vec_2)
```

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

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

```
# Actually using the handlers
mean(my_vec_1, na.rm = TRUE)
```

```
## [1] 2.5
```

```
cor(my_vec_1, my_vec_2, use = 'complete.obs')
```

```
## [1] -0.3273
```

Impute missing data

Missing data can be **imputed**.

How missing data should be imputed depends on whether the **data is missing at random** or not.

Packages: `Hmisc`, `DMwR`, `mice`, etc.

```r
# Example
my_df <- data.frame('x' = c(1, 2,  3, 4, NA),
                    'y' = c(4, 2, NA, 3, 5)
                    )

# Impute using mean
my_df[[1]][is.na(my_df[[1]])] <-
  mean(my_df[[1]], na.rm = TRUE)

# Impute using regression (package: mice)
model <- mice(my_df, method = 'norm', printFlag = FALSE)
my_df <- complete(model)

# print
my_df
```

```
##     x     y
## 1 1.0 4.000
## 2 2.0 2.000
## 3 3.0 1.833
## 4 4.0 3.000
```

Practical

Link to practical