R-CodingStandard

Coding standard for R, combining: * The book 'Advanced R' by Hadley Wickham, CRC Press, 2014 * Google's R Style Guide * The document 'Rchaeology: Idioms of R Programming' by Paul E. Johnson, January 28, 2015, PDF * The document 'Rtips. Revival 2014!' by Paul E. Johnson, March 24, 2014, PDF * The document 'R Style. An Rchaeological Commentary.' by Paul E. Johnson, February 13, 2015, PDF

G: General

G.1: Be consistent

Whatever you do, do it consistently.

References

- Google's R Style Guide: `Use common sense and BE CONSISTENT.`

G.2: Use common sense

Not all things are caught in a coding standard.

References

- Google's R Style Guide: `Use common sense and BE CONSISTENT.`

G.3: Avoid lines longer than 80 characters

`[Example here]`

References

- Google's R Style Guide: `The maximum line length is 80 characters`

G.4: When modifying other people's code, follow their coding standard

`[Example here]`

References

- Google's R Style Guide: Use common sense and BE CONSISTENT. If you are editing code, take a few minutes to look at the code around you and determine its style. If others use spaces around their if clauses, you should, too. If their comments have little boxes of stars around them, make your comments have little boxes of stars around them, too. The point of having style guidelines is to have a common vocabulary of coding so people can concentrate on what you are saying, rather than on how you are saying it. [...]. But local style is also important. If code you add to a file looks drastically different from the existing code around it, the discontinuity will throw readers out of their rhythm when they go to read it. Try to avoid this.

G.5: Do not put more than one command on the same line

```
x <- 42; y <- 314 # Bad

# Good:
x <- 42
y <- 314
```

References

- Google's R Style Guide: Do not terminate your lines with semicolons or use semicolons to put more than one command on the same line

G.6: Prefer using S3 over S4

```
[Example here]
```

Exceptions

Justifications for an S4 object would be:

- to use objects directly in C++ code
- to dispatch on two arguments

References

- Google's R Style Guide: `Use S3 objects and methods unless there is a strong reason to use S4 objects or methods. A primary justification for an S4 object would be to use objects directly in C++ code. A primary justification for an S4 generic/method would be to dispatch on two arguments.`

G.7: Avoid mixing S3 and S4

`[Example here]`

References

- Google's R Style Guide: `Avoid mixing S3 and S4: S4 methods ignore S3 inheritance and vice-versa`

WS: Whitespace

WS.1: Prefer placing spaces around all binary operators [1,2]

```
x <- 42
y <- x + 314
z <- y - 42

if (a == b) print("Equal")
if (a != b) print("Unequal")
if (a < b) print("Less")
if (a <= b) print("Less or equal")
if (a > b) print("Greater")
if (a >= b) print("Greater or equal")

Show <- function(s = "Hello World") { # Note the spaces around the =
   # ...
}
```

Exception

Spaces around =s are preferred [2] or optional [1] when passing parameters in a function call.

```
rep(x = 314, times = 42) # [1] Good [2] Good
rep(x=314, times=42)     # [1] Okay [2] Bad
```

References

- [1][Google's R Style Guide](https://google.github.io/styleguide/Rguide.xml): Place spaces around all binary operators (=, +, -, <-, etc.). Exception: Spaces around ='s are optional when passing parameters in a function call.
- [2] Wickham, Hadley. Advanced R. CRC Press, 2014. Chapter 5.2.1: Place spaces around all infix operators (=, +, -, <-, etc.). The same rule applies when using = in function calls. Always put a space after a comma, and never before (just like in regular English).

WS.2: Avoid placing spaces around code in parentheses or square brackets

```
if (debug)   # Good
if ( debug ) # Bad
if (debug )  # Bad
if ( debug)  # Bad

x[1]   #Good
x[ 1 ] #Bad
x[1 ]  #Bad
x[ 1]  #Bad
```

Exception

Always place a space after a comma:

```
x[1, ] #Good
x[1,]  #Bad
```

References

- Google's R Style Guide: Do not place spaces around code in parentheses or square brackets. Exception: Always place a space after a comma.

WS.3: Prefer indenting with two spaces

```
if (x == 42) {
  print("Yes!")
}
```

Exception

When a line break occurs inside parentheses, align the wrapped line with the first character inside the parenthesis.

Extra spacing is okay if it improves alignment of equals signs `=` or arrows `<-`.

```
plot(
  x    = xs,
  y    = ys,
  xlab = "X",
  ylab = "Y",
  main = "Title"
)
```

References

- Google's R Style Guide: `When indenting your code, use two spaces. Never use tabs or mix tabs and spaces. Exception: When a line break occurs inside parentheses, align the wrapped line with the first character inside the parenthesis.`
- Google's R Style Guide: `Extra spacing (i.e., more than one space in a row) is okay if it improves alignment of equals signs or arrows (<-).`

WS.4: Avoid using tabs

`[Example here]`

References

- Google's R Style Guide: `When indenting your code, use two spaces. Never use tabs or mix tabs and spaces.`

WS.5: Avoid placing a space before a comma. Prefer to place a space after a comma

```
t <- sum(x[, 1]) # Good
t <- sum(x[1, ]) # Good
t <- sum(x[,1]) # Bad
t <- sum(x[1,]) # Bad
```

References

- Google's R Style Guide: Do not place a space before a comma, but always place one after a comma.

WS.6: Prefer placing a space before a left parenthesis, except when calling a function

```
if (debug) # Good
if(debug)  # Bad
print("Hello")  #Good
print ("Hello") #Bad
```

References

- Google's R Style Guide: Place a space before left parenthesis, except in a function call.

BR: Braces

BR.1: Prefer putting an opening curly brace at the end of a line. Prefer to put a closing curly brace on its own line

```
# GOOD
if (x == 42) {
  # ...
}

# BAD
if (x == 42)
{
  # ...
}
```

```
# BAD
if (x == 42)
  {
  # ...
  }
```

References

- Google's R Style Guide: `An opening curly brace should never go on its own line; a closing curly brace should always go on its own line.`

BR.2: Be consistent in ommitting curly braces when they can be ommitted

You may omit curly braces when a block consists of a single statement:

```
# Good: keeping the curly braces
if (x == 42) {
  print("OK")
}

if (y == 314) {
  print("OK")
}

# Good: ommitting the curly braces
if (x == 42)
  print("OK")

if (y == 314)
  print("OK")

# Bad: inconsistency in ommitting the curly braces
if (x == 42)
  print("OK")

if (y == 314) {
  print("OK")
}
```

References

- Google's R Style Guide: You may omit curly braces when a block consists of a single statement; however, you must consistently either use or not use curly braces for single statement blocks.

BR.3: Prefer to begin the body of a block on a new line

```
# OK
if (x == 42) {
  print("OK")
}

# Bad?
if (x == 42) { print("Bad") }
```

References

- Google's R Style Guide: Always begin the body of a block on a new line.

BR.4: Prefer to surround `else` with braces

```
# OK
if (x == 42) {
  # ...
} else {
  # ...
}


# Bad
if (x == 42) {
  # ...
}
else {
  # ...
}

# Bad
if (x == 42)
  # ...
```

```
else
  # ...
```

References

- Google's R Style Guide: `Surround else with braces. An else statement should always be surrounded on the same line by curly braces.`

I: Identifiers

I.1: Prefer naming non-constant variables in all lower case separated with dots [1] or underscores [2]

Here, the advice disagrees:

```
my.value # [1] Good [3] Bad
myVlicks # [1] Okay [3] Bad
my_value # [2] Bad  [3] Good
my-value # [2] Bad  [3] Bad
```

References

- [1] [Google's R Style Guide](https://google.github.io/styleguide/Rguide.xml): `Identifiers should be named according to the following conventions. The preferred form for variable names is all lower case letters and words separated with dots (variable.name), but variableName is also accepted.`
- [2] [Google's R Style Guide](https://google.github.io/styleguide/Rguide.xml): `Don't use underscores ( _ ) or hyphens ( - ) in identifiers`
- [3] Wickham, Hadley. Advanced R. CRC Press, 2014. Chapter 5.1.2: `Variables and function names should be lowercase. Use an underscore to seperate words within a name. Generally, variable names should be nouns and function names should be verbs. Strive for names that are concise and meaningful`

I.2 Prefer a variable being a noun

```
variance # Good
working # Bad
```

References

- [1] Wickham, Hadley. Advanced R. CRC Press, 2014. Chapter 5.1.2: `Variables and function names should be lowercase. Use an underscore to seperate words within a name. Generally, variable names should be nouns and function names should be verbs. Strive for names that are concise and meaningful`

I.2: consider naming constant variables differently [1]

This idea is suggested by [1], it is unknown what [2] thinks about this.

When embracing the idea, I add the naming schemes for both [1] and [2]:

```
kPi  <- 3.14  # [1] Good [2] Bad
k_pi <- 3.14  # [1] Bad  [2] Unknown
```

References

- [1] [Google's R Style Guide](https://google.github.io/styleguide/Rguide.xml): `constants are named like functions but with an initial k`
- [2] Wickham, Hadley. Advanced R. CRC Press, 2014. Chapter 5.1.2: `Variables and function names should be lowercase. Use an underscore to seperate words within a name. Generally, variable names should be nouns and function names should be verbs. Strive for names that are concise and meaningful`

I.4: Prefer naming functions in CamelCase [1] or lower_case_with_underscores [2]

Here, the advice disagrees:

```
CalculateVariance  # [1] Good [2] Bad
calculateVariance  # [1] Bad  [2] Bad
calculate_variance # [1] Bad  [2] Good
```

References

- [1] [Google's R Style Guide](https://google.github.io/styleguide/Rguide.xml): `function names have initial capital letters and no dots`

- [2] Wickham, Hadley. Advanced R. CRC Press, 2014. Chapter 5.1.2: `Variables and function names should be lowercase. Use an underscore to seperate words within a name. Generally, variable names should be nouns and function names should be verbs. Strive for names that are concise and meaningful`

I.5: Prefer the first word of a function being a verb

```
MeasureSpeed  # [1] Good
measure_speed # [3] Good
Speed         # [1,3] Bad
speed         # [1,3] Bad
```

Exception

When creating a class object, the function name (constructor) and class name should match [2]

References

- [1][Google's R Style Guide](https://google.github.io/styleguide/Rguide.xml): `Make function names verbs`
- [2][Google's R Style Guide](https://google.github.io/styleguide/Rguide.xml): `When creating a class object, the function name (constructor) and class name should match`
- [3] Wickham, Hadley. Advanced R. CRC Press, 2014. Chapter 5.1.2: `Variables and function names should be lowercase. Use an underscore to seperate words within a name. Generally, variable names should be nouns and function names should be verbs. Strive for names that are concise and meaningful`

FU: Functions

FU.1: Prefer to start function definition argument lists with the non-defaultable values

```
# Good
Transmogrify <- function(
  x,
  y = 42
)
```

```
# Bad
Transmogrify <- function(
  x = 314,
  y
)
```

References

- Google's R Style Guide: `Function definitions should first list arguments without default values, followed by those with default values.`

Fu.2: Prefer to optionally use line breaks in function definitions between assignments only

```
# Good
Transmogrify <- function(
  x = 314,
  y = 42
)

# Good
Transmogrify <- function(x = 314, y = 42)

# Bad
Transmogrify <- function(x
  = 314, y = 42)
```

References

- Google's R Style Guide: `In both function definitions and function calls, multiple arguments per line are allowed; line breaks are only allowed between assignments.`

FU.3: Prefer to optionally use line breaks in function calls between assignments only

```
# Good
Transmogrify(
  x = 314,
  y = 42
```

```
)

# Good
Transmogrify(x = 314, y = 42)

# Bad
Transmogrify(x
  = 314, y = 42)
```

References

- Google's R Style Guide: `In both function definitions and function calls, multiple arguments per line are allowed; line breaks are only allowed between assignments.`

FU.4: Prefer to describe a function in comments in the line(s) directly below the function definition

These comments may consist of:

- a one-sentence description of the function
- a list of the function's arguments, denoted by Args:, with a description of each (including the data type)
- and a description of the return value, denoted by Returns:

The comments should be descriptive enough that a caller can use the function without reading any of the function's code.

```
SumFloats <- function(x) {
  # Calculates the sum of a vector of floats
  # Args: x: a vector containing zero or more floating point values
  # Returns: the sum of the vector as a floating point
}
```

References

- Google's R Style Guide: `Functions should contain a comments section immediately below the function definition line. These comments should consist of a one-sentence description of the function; a list of the function's arguments, denoted by Args:, with a description of each (including the data type); and a description of the return value, denoted by`

Returns:. The comments should be descriptive enough that a caller can use the function without reading any of the function's code.

FU.5: Prefer using `typeof()` over `mode()`

`mode()` is an alias of `typeof()` that only exists for S compatibility.

```
typeof(x) # Good
mode(x)   # Bad
```

References

- Wickham, Hadley. Advanced R. CRC Press, 2014. Chapter 7.1, page 102: You may have heard of mode() and storage.mode(). I recommend ignoring these functions because they are just aliases of the names returned by typeof(), and exist solely for S compatibility

FU.6: Prefer using `typeof()` over `storage.mode()`

`storage.mode()` is an alias of `typeof()` that only exists for S compatibility.

```
typeof(x)         # Good
storage.mode(x)   # Bad
```

References

- Wickham, Hadley. Advanced R. CRC Press, 2014. Chapter 7.1, page 102: You may have heard of mode() and storage.mode(). I recommend ignoring these functions because they are just aliases of the names returned by typeof(), and exist solely for S compatibility

FU.7: Avoid using `attach`

[Example here]

References

- Google's R Style Guide: The possibilities for creating errors when using attach are numerous. Avoid it.

O: Operator use

O.1: Prefer using <- over = for assignment.

```
x <- 42 # Good
x = 42  # Bad
```

References

- Google's R Style Guide: `Use <-, not =, for assignment`

O.2: Avoid using semicolons ;

```
x <- 42  # Good
x <- 42; # Bad
```

References

- Google's R Style Guide: `Do not terminate your lines with semicolons or use semicolons to put more than one command on the same line`

FI: File

FI.1: Prefer meaningful files names ending in .R

- Good filenames: `predict_ad_revenue.R`, `fit-models.R`
- Bad filenames: `predict_ad_revenue.r`,`foo.R`, `utility-functions.R`

References

- Google's R Style Guide: `File names should end in .R and, of course, be meaningful`
- Wickham, Hadley. Advanced R. CRC Press, 2014. Chapter 5.1.1: `Filenames should be meaningful and end in .R`

FI.2: If files need to be run in sequence, consider prefixing them with numbers

```
0-download.R
1-parse.R
2-explore.R
```

References

- Wickham, Hadley. Advanced R. CRC Press, 2014. Chapter 5.1.1: `If files need to be run in sequence, prefix them with numbers`

FI.3: Prefer naming unit test files ending in `_test.R`

- Good filenames: `predict_ad_revenue_test.R`
- Bad filames: `test_predict_ad_revenue.R`

References

- Google's R Style Guide: `Unit tests should go in a separate file named originalfilename_test.R.`

FI.4: Be consistent in the ordering of a file

An example ordering: * Copyright statement comment * Author comment * File description comment, including purpose of program, inputs, and outputs * `source()` and `library()` statements * Function definitions * Executed statements, if applicable, for example `print` and `plot`

References

- Google's R Style Guide: `If everyone uses the same general ordering, we'll be able to read and understand each other's scripts faster and more easily.`

CO: Comments

CO.1: Prefer to add comments to your code

```
# Determine if expensive calculation can be avoided
```

References

- Google's R Style Guide: `Comment your code`

CO.2: Prefer to start an entire-line-comment with # and one space

```
# Determine if expensive calculation can be avoided
```

References

- Google's R Style Guide: `Entire commented lines should begin with # and one space.`

CO.3: For code directly after code, add two spaces, # and then one space

```
if (x == 42) {  # Are we lucky?
  # ...
}
```

References

- Google's R Style Guide: `Short comments can be placed after code preceded by two spaces, #, and then one space.`

ER: Error handling

ER.1: Errors should be raised using stop()

```
if (x < 0) stop()
```

References

- Google's R Style Guide: `Errors should be raised using stop()`

TE: Testing

TE.1: Prefer to put unit tests in seperate files

See also 'prefer naming unit test files ending in `_test.R`'

References

- Google's R Style Guide: `Unit tests should go in a separate file named originalfilename_test.R.`

TE.2: Prefer to let your unit tests serve as example function calls

`[Example]`

References

- Google's R Style Guide: `Ideally, unit tests should serve as sample function calls (for shared library routines).`

TO: TODO's

TO.1: Prefer to use a consistent style for TODOs

For example:

```
TODO([username]): [Explicit description of action to be taken]
```

Like:

```
TODO(richelbilderbeek): Check if this really works
```

References

- Google's R Style Guide: `Use a consistent style for TODOs throughout your code. TODO(username): Explicit description of action to be taken`