CCD's R Style Guide

The code standards from Clube de Ciência de Dados

"R is a language and environment for statistical computing and graphics.". The goal of the R Programming Style Guide is to make our R code easier to read, share, and verify. The rules below were designed by Clube de Ciência de Dados based on existing standards.

Summary

Notation and Naming

File Names

File names must be meaningful and end in .R

```
#GOOD

fit-models.R

utility-functions.R

#BAD

foo.R

stuff.R
```

Identifiers

Variables

Variable names must be lowercase. Use an underscore (_) to separate words within a name. Generally, variable names should be nouns. Try to find names that are concise and meaningful (that's quite not easy!).

```
#GOOD

day_one

day_1

#BAD

first.day.of.the.month

DayOne

dayone

djm-1
```

Functions

Function names must be on lowerCamelCase style. Generally, function names should be verbs.

```
#GOOD

calculateSum

#BAD

calculate_sum

CalculateSum
```

Constants

Constants names must have all capital letters. Use an underscore (_) to separate words within a name.

```
#GOOD

CONSTANT_NAME

#BAD

kConstantName

ConstantName

constant_name
```

- Syntax
  - Line Length
    The maximum line length is 80 characters.
  - Indentation
    When indenting your code, use two spaces. Never use space of a tab or mix spaces of tabs and other spaces.

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

- Spacing

  Place spaces around all binary operators ('=', '+', '-', '<-', etc.). Do not place a space before a comma, but always place one after a comma.

  Exception: Spaces around ='s are optional when passing parameters in a function call.

```
#GOOD

tab_prior <- table(df[df$days_from_opt < 0, "campaign_id"])

total <- sum(x[, 1])

total <- sum(x[1, ])

#BAD

tab_prior <- table(df[df$days_from_opt<0, "campaign_id"])  # Needs spaces around '<'

tab_prior <- table(df[df$days_from_opt < 0,"campaign_id"])  # Needs a space after the comma

tab_prior<- table(df[df$days_from_opt < 0, "campaign_id"])  # Needs a space before <-

tab_prior<-table(df[df$days_from_opt < 0, "campaign_id"])  # Needs spaces around <-

total <- sum(x[,1])  # Needs a space after the comma

total <- sum(x[ ,1])  # Needs a space after the comma, not before
```

  Place a space before left parenthesis, except in a function call.

```
#GOOD

if (debug)

Calculate()

#BAD

if(debug)

Calculate ()
```

  Extra spacing (i.e., more than one space in a row) is okay if it improves alignment of equals signs or arrows (<-).

```
#GOOD

plot(x    = x.coord,
     y    = data.mat[, MakeColName(metric, tiles[1], "roiOpt")],
     ylim = ylim,
     xlab = "dates",
     ylab = metric,
     main = (paste(metric, " for 3 samples ", sep = "")))
#BAD

plot(x = x.coord,
     y = data.mat[, MakeColName(metric, tiles[1], "roiOpt")],
     ylim = ylim,
     xlab = "dates",
     ylab = metric,
     main = (paste(metric, " for 3 samples ", sep = "")))
```

Do not place spaces around code in parentheses or square brackets.
Exception: Always place a space after a comma.

```
#GOOD

if (debug)

x[1, ]

#BAD

if ( debug )  # No spaces around debug

x[1,]  # Needs a space after the comma
```

- Braces

  Curly Braces

  An opening curly brace must never go on its own line; a closing curly brace must always go on its own line.

```
#GOOD

if (a < 1) {
  ylim <- c(0, 0.06)
}
#BAD
if (a < 1){ylim <- c(0, 0.06)}

if (a < 1)ylim <- c(0, 0.06)

if (a < 1)
  ylim <- c(0, 0.06)
```

  Surround else with braces

  An `else` statement must always be surrounded on the same line by curly braces and it have to be at the same line of the `if`'s end brace.

```
#GOOD

if (condition) {
  one or more lines
} else {
  one or more lines
}
#BAD

if (condition) {
  one or more lines
}
else {
  one or more lines
}

if (condition)
  one line
else
  one line
```

- Assignment
  Use '<-', not '=', for assignment.

```
#GOOD

x <- 5

#BAD

x = 5
```

- Semicolons
  Do not terminate your lines with semicolons or use semicolons to put more than one command on the same line.

```
#GOOD

x <- 0
x <- x + 5

x <- 0; x + 5

#BAD

x <- 0;
x <- x + 5;

x <- 0 x <- x + 5
```

- Organization
  - Written Language

    All simple variable name, function name, comment or any other thing must be written in english.

    ```
    #GOOD

    multiplyByTwo

    #BAD

    multiplicaPorDois

    multiplizieremitZwei

    由兩個乘
    ```

  - General Layout and Ordering

    If everyone uses the same general ordering, we'll be able to read and understand each other's scripts faster and more easily.

    1. Copyright statement comment
    2. Author comment
    3. File description comment, including purpose of program, inputs, and outputs
    4. `source()` and `library()` statements
    5. Function definitions
    6. Executed statements, if applicable (e.g., print, plot)

  - Commenting Guidelines

    Comment your code (it will be funny if you have sense o humor)! Entire commented lines should begin with # and one space. Short comments can be placed after code preceded by two spaces, #, and then one space.

    ```
    # Create histogram of frequency of campaigns by pct budget spent.
    hist(df$pct_spent,
         breaks = "scott",  # method for choosing number of buckets
         main   = "Histogram: fraction budget spent by campaignid",
         xlab   = "Fraction of budget spent",
         ylab   = "Frequency (count of campaignids)")
    ```

  - Function Documentation

    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. After all, it must a enter space between the function documentation and the code.

```
CalculateSum <- function(value_1 = 0, value_2 = 0) {
  # Calculate the sum of two numbers.
  #
  # Args:
  #   x: One of the two number.
  #   y: The other number.
  #
  # Returns:
  #   The sum of the two numbers

  return(value_1 + value_2)
}
```

Conclusion

The great point of having style guidelines is creat a common sense in coding, so people can really fast read your codes, understands your way-of-thinking and work on your code. 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 (and that sucks!). Try to avoid this (more like do not do this!).

OK, enough writing about writing code; the code itself is much more interesting. Have fun!

Reference

[1] https://google-styleguide.googlecode.com/svn/trunk/Rguide.xml

[2] http://adv-r.had.co.nz/Style.html

[3] http://www1.maths.lth.se/help/R/RCC/