来自 Google 的 R 语言编码风格指南

R 语言是一门主要用于统计计算和绘图的高级编程语言. 这份 R 语言编码风格指南旨在让我们的 R 代码更容易阅读、分享和检查. 以下规则系与 Google 的 R 用户群体协同设计而成.

概要: R编码风格约定

1. 文件命名: 以 `.R` (大写) 结尾
2. 标识符命名: `variable.name`, `FunctionName`, `kConstantName`
3. 单行长度: 不超过 80 个字符
4. 缩进: 两个空格, 不使用制表符
5. 空白
6. 花括号: 前括号不折行写, 后括号独占一行
7. 赋值符号: 使用 `<-`, 而非 `=`
8. 分号: 不要用
9. 总体布局和顺序
10. 注释准则: 所有注释以 `#` 开始, 后接一个空格; 行内注释需要在 `#` 前加两个空格
11. 函数的定义和调用
12. 函数文档
13. 示例函数
14. TODO 书写风格: `TODO(您的用户名)`

概要: R语言使用规则

1. `attach`: 避免使用
2. 函数: 错误 (error) 应当使用 `stop()` 抛出
3. 对象和方法: 尽可能避免使用 S4 对象和方法; 永远不要混用 S3 和 S4

1. 表示和命名

文件命名

文件名应以 `.R` (大写) 结尾, 文件名本身要有意义.
正例: `predict_ad_revenue.R`
反例: `foo.R`

标识符命名

在标识符中不要使用下划线 (`_`) 或连字符 (`-`). 标识符应根据如下惯例命名. 变量名应使用点 (`.`) 分隔所有的小写字母或单词; 函数名首字

母大写, 不用点分隔 (所含单词首字母大写); 常数命名规则同函数, 但需使用一个 `k` 开头.

- `variable.name`
  正例: `avg.clicks`
  反例: `avg_Clicks` , `avgClicks`
- `FunctionName`
  正例: `CalculateAvgClicks`
  反例: `calculate_avg_clicks` , `calculateAvgClicks`
  函数命名应为动词或动词性短语.
  例外: 当创建一个含类 (class) 属性的对象时, 函数名 (也是 constructor) 和类名 (class) 应当匹配 (例如, lm).
- `kConstantName`

2. 语法

单行长度

最大单行长度为 80 个字符.

缩进

使用两个空格来缩进代码. 永远不要使用制表符或混合使用二者.
例外: 当括号内发生折行时, 所折行与括号内的第一个字符对齐.

空白

在所有二元操作符 (=, +, -, <-, 等等) 的两侧加上空格.
例外: 在函数调用中传递参数时 = 两边的空格可加可不加.

不可在逗号前加空格, 逗号后总须加空格.

正例:

```
tabPrior <- table(df[df$daysFromOpt < 0, "campaignid"])
        total <- sum(x[, 1])
        total <- sum(x[1, ])
```

反例:

```
tabPrior <- table(df[df$daysFromOpt<0, "campaignid"])  # 在 '<' 两侧需要增加空格
tabPrior <- table(df[df$daysFromOpt < 0,"campaignid"])  # 逗号后需要一个空格
tabPrior<- table(df[df$daysFromOpt < 0, "campaignid"])  # 在 <- 前需要一个空格
tabPrior<-table(df[df$daysFromOpt < 0, "campaignid"])  # 在 <- 两侧需要增加空格
total <- sum(x[,1])  # 逗号后需要一个空格
total <- sum(x[ ,1])  # 逗号后需要一个空格, 而非逗号之前
```

在前括号前加一个空格, 函数调用时除外.

正例:
`if (debug)`

反例:
`if(debug)`

多加空格 (即, 在行内使用多于一个空格) 也是可以的, 如果这样做能够改善等号或箭头 (<-) 的对齐效果.

```
plot(x    = xCoord,
     y    = dataMat[, makeColName(metric, ptiles[1], "roiOpt")],
     ylim = ylim,
     xlab = "dates",
     ylab = metric,
     main = (paste(metric, " for 3 samples ", sep="")))
```

不要向圆括号或方括号中的代码两侧加入空格.
例外: 逗号后总须加空格.

正例:

```
if (debug)
        x[1, ]
```

反例:

```
if ( debug )  # debug 的两边不要加空格
x[1,]  # 需要在逗号后加一个空格
```

花括号

前括号永远不应该独占一行; 后括号应当总是独占一行. 您可以在代码块只含单个语句时省略花括号; 但在处理这类单个语句时, 您必须 *前后一致地* 要么全部使用花括号, 或者全部不用花括号.

```
if (is.null(ylim)) {
  ylim <- c(0, 0.06)
}
```

或 (不可混用)

```
if (is.null(ylim))
  ylim <- c(0, 0.06)
```

总在新起的一行开始书写代码块的主体.

反例:

```
if (is.null(ylim)) ylim <- c(0, 0.06)
if (is.null(ylim)) {ylim <- c(0, 0.06)}
```

赋值

使用 <- 进行赋值, 不用 = 赋值.

正例:

```
x <- 5
```

反例:

```
x = 5
```

分号

不要以分号结束一行, 也不要利用分号在同一行放多于一个命令. (分号是毫无必要的, 并且为了与其他Google编码风格指南保持一致, 此处同样略去.)

3. 代码组织

总体布局和顺序

如果所有人都以相同顺序安排代码内容, 我们就可以更加轻松快速地阅读并理解他人的脚本了.

1. 版权声明注释
2. 作者信息注释
3. 文件描述注释, 包括程序的用途, 输入和输出
4. `source()` 和 `library()` 语句
5. 函数定义
6. 要执行的语句, 如果有的话 (例如, `print`, `plot`)

单元测试应在另一个名为 `原始的文件名_unittest.R` 的独立文件中进行.

注释准则

注释您的代码. 整行注释应以 # 后接一个空格开始.

行内短注释应在代码后接两个空格, #, 再接一个空格.

```
# Create histogram of frequency of campaigns by pct budget spent.
    hist(df$pctSpent,
    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)")
```

函数的定义和调用

函数定义应首先列出无默认值的参数, 然后再列出有默认值的参数.

函数定义和函数调用中, 允许每行写多个参数; 折行只允许在赋值语句外进行.
正例:

```
PredictCTR <- function(query, property, numDays,
    showPlot = TRUE)
```

反例:

```
PredictCTR <- function(query, property, numDays, showPlot =
                       TRUE)
```

理想情况下, 单元测试应该充当函数调用的样例 (对于包中的程序来说).

函数文档

函数在定义行下方都应当紧接一个注释区. 这些注释应当由如下内容组成: 此函数的一句话描述; 此函数的参数列表, 用 `Args:` 表示, 对每个参数的描述 (包括数据类型); 以及对于返回值的描述, 以 `Returns:` 表示. 这些注释应当描述得足够充分, 这样调用者无须阅读函数中的任何代码即可使用此函数.

示例函数

```
CalculateSampleCovariance <- function(x, y, verbose = TRUE) {
# Computes the sample covariance between two vectors.
#
# Args:
#   x: One of two vectors whose sample covariance is to be calculated.
#   y: The other vector. x and y must have the same length, greater than one,
#      with no missing values.
#   verbose: If TRUE, prints sample covariance; if not, not. Default is TRUE.
#
# Returns:
#   The sample covariance between x and y.
n <- length(x)
# Error handling
if (n <= 1 || n != length(y)) {
stop("Arguments x and y have invalid lengths: ",
length(x), " and ", length(y), ".")
}
if (TRUE %in% is.na(x) || TRUE %in% is.na(y)) {
stop(" Arguments x and y must not have missing values.")
}
covariance <- var(x, y)
if (verbose)
cat("Covariance = ", round(covariance, 4), ".\n", sep = "")
return(covariance)
}
```

TODO 书写风格

编码时通篇使用一种一致的风格来书写 TODO.
`TODO(您的用户名): 所要采取行动的明确描述`

4. 语言

Attach

使用 `attach` 造成错误的可能数不胜数. 避免使用它.

函数

错误 (error) 应当使用 `stop()` 抛出.

对象和方法

S 语言中有两套面向对象系统, S3 和 S4, 在 R 中这两套均可使用. S3 方法的可交互性更强, 更加灵活, 反之, S4 方法更加正式和严格. (对这两套系统的说明, 参见 Thomas Lumley 的文章 "Programmer's

Niche: A Simple Class, in S3 and S4", 发表于 R News 4/1, 2004, 33 - 36 页: https://cran.r-project.org/doc/Rnews/Rnews_2004-1.pdf.)

这里推荐使用 S3 对象和方法, 除非您有很强烈的理由去使用 S4 对象和方法. 使用 S4 对象的一个主要理由是在 C++ 代码中直接使用对象. 使用一个 S4 泛型/方法的主要理由是对双参数的分发.

避免混用 S3 和 S4: S4 方法会忽略 S3 中的继承, 反之亦然.

5. 例外

除非有不去这样做的好理由, 否则应当遵循以上描述的编码惯例. 例外包括遗留代码的维护和对第三方代码的修改.

6. 结语

遵守常识, **前后一致**.

如果您在编辑现有代码, 花几分钟看看代码的上下文并弄清它的风格. 如果其他人在 `if` 语句周围使用了空格, 那您也应该这样做. 如果他们的注释是用星号组成的小盒子围起来的, 那您也要这样写。

遵循编码风格准则的意义在于, 人们相当于有了一个编程的通用词汇表, 于是人们可以专注于您在 *说什么*, 而不是您是 *怎么说* 的. 我们在这里提供全局的编码风格规则以便人们了解这些词汇, 但局部风格也很重要. 如果您加入文件中的代码看起来和周围的已有代码截然不同, 那么代码阅读者的阅读节奏就会被破坏. 尽量避免这样做. OK, 关于如何写代码已经写得够多了; 代码本身要有趣的多. 编码愉快!

7. 参考文献

http://www.maths.lth.se/help/R/RCC/ - R语言编码惯例
https://ess.r-project.org/ - 为 emacs 用户而生. 在您的 emacs 中运行 R 并且提供了一个 emacs mode.