구글 R 스타일 가이드(Google's R Style Guide)

R은 주로 통계 계산 및 그래픽에 사용되는 고급 프로그래밍 언어입니다. 이 R 프로그래밍 스타일 가이드의 목표는 R 코드를 더 쉽게 읽고, 공유하고 그리고 확인하도록 만드는 것입니다. 아래 규칙은 Google의 전체 R 사용자 커뮤니티와 공동으로 설계되었습니다.

Summary: R Style Rules

1. 파일 이름: 마지막에 `.R`를 넣자.
2. 식별자(identifiers): `variable.name` (or `variableName`), `FunctionName`, `kConstantName`
3. 줄 길이(Line Length): 최대 80 글자
4. 들여 쓰기(Indentation): 탭 쓰지 말고, 2 스페이스
5. 공백 넣기(Spacing)
6. 중괄호(Curly Braces):
7. else: Surround else with braces
8. 할당(Assignment): `<-`을 사용하고, `=`는 사용하지 마세요.
9. 세미콜론(Semicolons): 이것을 사용하지 마세요.
10. 일반적인 레이아웃과 순서(General Layout and Ordering)
11. 주석 달기 지침(Commenting Guidelines): 모든 주석은 공백 한 칸(a space)이 딸려 있는 #으로 시작합니다; 인라인(inline) 주석은 # 앞에 공백 2 칸이 필요합니다.
12. 함수 정의와 호출(Function Definitions and Calls)
13. 함수 문서화(Function Documentation)
14. Example Function
15. TODO Style: `TODO(username)`

Summary: R Language Rules

1. `attach`: 이것을 사용하지 마세요.
2. Functions: `stop()`을 사용할 때에는 에러가 발생하게 해야 합니다.
3. Objects and Methods: 가능한 한 S4 객체와 메소드는 피하세요; 절대로 S3와 S4를 섞지 마세요

Notation and Naming

파일 이름(File Names)

파일 이름은 `.R`으로 끝나야 하고, 의미가 있어야 한다.
GOOD: `predict_ad_revenue.R`
BAD: `foo.R`

식별자(Identifiers)

식별자(identifiers)안에는 밑줄 (_)이나 하이픈 (-)을 사용하지 마십시오. 식별자는 다음 규칙에 따라 이름을 붙여야 합니다. 변수(variable) 이름으로 선호되는 형식은 모두 소문자이고 점으로 구분된 단어 (`variable.name`) 이지만, `variableName` 도 허용합니다; 함수 이름 맨 앞글자는 대문자로 하고 점이 없이 (`FunctionName`) 과 같이 정합니다; 상수(constants)에는 함수처럼 이름을 붙이지만 맨 앞에 `k`라는 이니셜을 붙입니다.

- `variable.name`을 선호하지만, `variableName` 도 받아드립니다.
  GOOD: `avg.clicks`
  OK: `avgClicks`
  BAD: `avg_Clicks`
- 함수 이름
  GOOD: `CalculateAvgClicks`
  BAD: `calculate_avg_clicks` , `calculateAvgClicks`
  함수의 이름은 동사로 만듭니다.
  예외: 클래스된 대상(classed object)을 생성할 때에는, 함수 이름(constructor)과 클래스를 맞춰야 합니다.(보기, lm).
- `kConstantName`

Syntax

줄길이(Line Length)

줄 길이는 최대 80 글자입니다.

들여 쓰기(Indentation)

코드를 들여 쓰기 할 때, 공백 두 개(two spaces)을 사용하십시오. 절대로 탭이나 탭과 공백을 섞어서 사용하지 마세요.
예외: 괄호 안에서 줄을 바꿀 때, 줄 바뀜된 행(line)을 괄호 안의 첫 번째 문자와 맞춥니다.

공백 넣기(Spacing)

모든 이진 연산자 (=, +, -, <-, 등) 앞뒤에 공백을 넣으십시오.
예외 : 함수 호출에서 매개 변수(parameters)를 전달할 때 = 앞뒤에 공백을 넣는 것은 선택 사항입니다.

쉼표 앞에 공백을 두지 말고, 항상 쉼표 뒤에 공백 한 칸을 넣으십시오.

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$days.from.opt<0, "campaign.id"])  # '<'주위에 공백이 필요합니다.
tab.prior <- table(df[df$days.from.opt < 0,"campaign.id"])  # 쉼표 뒤에 공백 한 칸를 넣으세요
tab.prior<- table(df[df$days.from.opt < 0, "campaign.id"])  #  <- 앞에 공백 한 칸를 넣으세요
tab.prior<-table(df[df$days.from.opt < 0, "campaign.id"])  #  <- 주위에 공백이 필요합니다
total <- sum(x[,1])  #  쉼표 뒤에 공백 한 칸를 넣으세요
total <- sum(x[ ,1])  #  쉼표 앞이 아니라 뒤에 공백 한 칸를 넣으세요
```

함수 호출을 제외하고, 왼쪽 괄호 앞에 공백을 둡니다.

GOOD:
```
if (debug)
```

BAD:
```
if(debug)
```

같다(번역자 추가: =)나 화살표 (<-) 기호의 정렬을 향상시키는 것이라면 여분의 공백(즉, 한 줄에 두 개 이상의 공백)을 넣어도 괜찮습니다.

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

괄호 나 대괄호 안에 코드 주위에는 공백을 넣지 마십시오.
예외 : 항상 쉼표 뒤에 공백을 넣으십시오.

GOOD:

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

BAD:

```
if ( debug )  # No spaces around debug
x[1,]  # Needs a space after the comma
```

중괄호(Curly Braces)

여는 중괄호는 절대로 그 자체의 줄을 따라 가면 안됩니다; 닫는 중괄호는 항상 자신의 줄에 있어야 합니다. 블록이 단일 명령문으로 구성되어 있으면 중괄호를 생략할 수 있습니다; 그러나 단일 명령문 블록에 항상 중괄호를 사용하거나 사용하지 않아야 합니다.

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

xor (but not both)

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

항상 블록의 몸통은 새로운 줄에서 시작하세요.

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

else를 괄호로 둘러싸기(Surround else with braces)

`else` 명령문은 중괄호로 항상 같은 줄에서 둘러싸여야 합니다.

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

BAD:

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

BAD:

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

할당(Assignment)

할당하기 위해서 `<-`을 사용하고, `=`을 사용하지 마세요.

GOOD:
```
x <- 5
```

BAD:
```
x = 5
```

세미콜론(Semicolons)

세미콜론을 가지고 여러분의 줄을 제거하지 말고 세미콜론을 사용하여 같은 줄에 한 개 이상의 명령를 넣지 마세요. (세미콜론이 꼭 필요하지도 않고, 다른 구글 스타일 가이드에서도 지속적으로 생략하고 있습니다.)

Organization

일반적인 레이아웃과 순서(General Layout and Ordering)

만약 모든 사람들이 똑같은 일반적인 순서를 사용한다면, 우리는 다른 사람들의 스크립터(scripts)를 더 빠르고 더 쉽게 이해하고 읽을 수 있을 것입니다.

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`)

Unit tests should go in a separate file named `originalfilename_test.R`.

주석 달기 지침(Commenting Guidelines)

여러분 코드에 주석을 다세요. 모든 주석이 있는 줄은 #과 공백 한 칸(one space)으로 시작해야 합니다.

코드 뒤에 오는 짧은 주석은 공백 2칸(two spaces)을 한 다음, #, 그런 다음 공백 한 칸(one space) 뒤에 위치합니다.

```
# 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)")
```

함수 정의와 호출(Function Definitions and Calls)

함수 정의는 우선 디폴트 값이 없는 인수(arguments)부터 우선 배열하고 디폴트 값이 있는 것들이 뒷따라 와야 합니다.

함수 정의와 함수 호출 모두, 한 줄에 여러 개의 인수(arguments)가 허용됩니다; 줄 바꿈은 오로지 할당 사이에서만 허용됩니다.
GOOD:

```
PredictCTR <- function(query, property, num.days,
                       show.plot = TRUE)
```

BAD:

```
PredictCTR <- function(query, property, num.days, show.plot =
                       TRUE)
```

Ideally, unit tests should serve as sample function calls (for shared library routines). 이상적으로, 단위 테스트는 샘플 함수 호출로 사용되어야 합니다(공유된 라이브러리 루틴을 위해서)

함수 문서화(Function Documentation)

함수는 함수를 정의한 줄 바로 아래의 주석 섹션을 가져와야 합니다. 이런 주석은 그 함수를 한 문장으로 기술하는 것으로 구성되어야 합니다. 그 함수의 인수들(arguments)의 목록은 `Args:`로 표시하고, 각각의 설명(description)을 가집니다; 그리고 반환(return) 값 설명은, `Args:`로 표시합니다. 주석은 호출자(caller)가 그 함수의 어떤 코드도 읽지 않고 사용할 수 있을만큼 충분한 설명이어야 합니다.

Example Function

```
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 different 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 Style

코드 전체에서 TODO를 위하여 일관된 스타일을 사용하십시오.
`TODO(username): Explicit description of action to be taken`

Language

Attach

`attach`를 사용할 때 에러(errors)를 생성하게 될 가능성이 많습니다. 조심하세요.

Functions

`stop()`을 사용할 때에는 에러가 발생하게 해야 합니다.

Objects and Methods

S 언어는 2개의 객체 시스템, S3와 S4를 가지고 있고 있으며, 이 둘 모두 R에서는 사용가능합니다. S3 메소드(methods)는 더 상호 작용적이고(interactive) 더 유연하며, 반면에 S4 메소드(methods)는 보다 형식적이고 엄격합니다. (두 시스템에 대한 설명을 위해서, Thomas Lumley's "Programmer's Niche: A Simple Class, in S3 and S4" in R News 4/1, 2004, pgs. 33 - 36 을 보세요:) https://cran.r-project.org/doc/Rnews/Rnews_2004-1.pdf.)

S4 객체 또는 메소드를 사용할 강력한 이유가 없는 한 S3 객체 및 메소드를 사용하십시오. S4 객체를 위한 근본적인 정당성(justification)는 C ++ 코드에서 객체를 직접 사용하는 것입니다. S4 제네틱/메소드(generic / method)을 위한 근본적인 정당성은 두 개의 인수를 전달하는 것입니다.

S3과 S4을 섞는 것을 피하십시오. S4 메소드(methods)는 S3 상속을 무시하고 그 반대의 경우도 마찬가지입니다.

Exceptions

다르게 할 좋은 이유가 없는 한, 위에서 설명한 코딩 관습(conventions)을 따라야 합니다. 예외로는 레거시(legacy) 코드 및 제 3 자 코드(third-party) 수정이 포함됩니다.

Parting Words

상식을 사용하고 일관성을 유지하십시오.

여러분들이 코드를 편집하는 경우, 몇 분 정도 시간을 내어 주변의 코드를 쳐다 보고 스타일을 결정하십시오. 만약 다른 사람들이 `if` 문 앞뒤에 공백을 사용한다면, 역시 여러분들도 그렇게 해야 합니다. 만약 그들의 주석 둘레를 별표로 만든 작은 상자로 둘러싸여 있다면, 여러분의 주석도 별표로 만든 작은 상자로 둘러지도록 만들어야 합니다.

스타일 가이드 라인을 갖추고 있다는 점은 코딩하는데 공통 어휘를 갖는다는 것이기에 사람들은 여러분들이 말하고자 하는 ***방식***보다는 말하고 있는 ***것***에 집중할 수 있게 됩니다. 우리는 사람들이 그 어휘를 알 수 있도록 여기에 전반적인 (global) 스타일 규칙을 제시합니다. 그러나 특정한(local) 스타일 역시 중요합니다. 만약 여러분이 파일에 추가하는 코드가 기존 코드와 크게 다르다면, 그런 불연속은 (코드를) 보는 사람들의 리듬을 어긋나게 합니다. 이러한 일이 없도록 신경써주세요.

OK, 코드를 쓰는 법에 대해선 이 정도로 충분합니다; 코드 자체가 더 흥미로운 법이지요. 수고하세요!

References

http://www.maths.lth.se/help/R/RCC/ - R Coding Conventions

http://ess.r-project.org/ - For emacs users. This runs R in your emacs and has an emacs mode.