Code

Style guide and naming conventions for CSDC

last updated: July 9, 2018

- Introduction
- Naming variables
    - General conventions
    - Naming conventions specific to CDSC
        - Column names
        - Factor names
        - Scale-specific naming conventions
- Style guide
    - Spacing
        - General spacing conventions
        - Spacing in if else statements
        - Spacing long lines

Introduction

This guide is a starting place for thinking about how to write clean, understandable, reproducible code. In it includes standards specific to CDSC (e.g., standardized naming conventions for frequently used variables), but it also includes comments about general good practices. This style guide is derived from the tidyverse style guide (available at http://style.tidyverse.org (http://style.tidyverse.org)) and only includes what is most immediately relevant for people using R in our lab. However, I *highly* encourage you to reference the tidyverse style guide as you progress through your journey into the world of R as it is more thorough than this guide.

Naming variables

General conventions

1. Always, *always*, **always** use only undercase. Consistency makes it easier for you (and others!) to work with your code.
2. Use `_` to break up variable names. Do not use `-` or `.`
3. Do not use other special characters (e.g., `%`, `$`, `#`, etc.) or spaces in variable names. Variables should only be lowercase letters, numbers, and `_`.

Hide

```
# good
test
test_one
test_1

# bad
Test
testOne
TestOne
test1
test-1
test.1
```

4. Variable or object names should use nouns, whereas function names should generally use verbs.
5. Names should be short as possible while also maintaining informativeness.

Hide

```
# good
race
day_one
day_1

# bad
race_of_participant
rop
first_day_of_the_month
```

6. If you create new variables in your dataset that may not be intuitive to others (e.g., acronyms of measures), it's always helpful to have a section of the comments (preferable at the top of your code but also possibly where you create the variable) that define your variables.
    - This goes both for names that you created in your R code but also names that may have been inherited directly from your .csv/data file.

Hide

```
# In the following analyses "cdi" refers to data collected using the MacArthur-Bates C
ommunicative Development Inventory (CDI; e.g., Fenson et al., 2007)
# "amss" refers to responses to the Ainsworth Maternal Sensitivity Scale (e.g., Ainswo
rth et al., 1974)

colnames(df) <- c("gender", "race", "age_exact", "cdi", "amss")
```

7. If you use a scale or measure that has multiple items, subscores, etc., label your items consistently with the scale name first followed by additional disambiguations.

    - E.g., if you collected CDI data (a measure of language development), name your variables accordingly
    - “cdi_item” if you have individual items (e.g., “cdi_1”, “cdi_2” OR “cdi_hello”, “cdi_ball”)
    - “cdi_subscale” if there’s subscale scores (e.g., “cdi_comprehension”, “cdi_production”)
    - “cdi_total” if you would like to be unambiguous that columns refers to a total score from measure
8. Using acronyms as names is convenient (especially when you describe what they stand for in a comments) but using too many acronyms can be less transparent. Try to limit variable names to only one set of acronyms.
    - E.g., “cdi_p” is quicker to type but less transparent than “cdi_production”
    - Using more than one acronym or shortened form per variable name isn’t a strict no, but think of the information that is being lossed by doing so and whether it’s worth the gain in typing efficiency.

Naming conventions specific to CDSC

Column names

1. `gender` **, not** `sex`
    - The measures we use typically capture the gender of the participant as opposed to their biological sex. Feel free to use `sex` if you feel the variable name more accurately captures the measure used in your study. Otherwise, use `gender` .
2. `race` **to refer to participant’s race or ethnicity**
3. `dob` **for date of birth and** `dot` **for date of test.**
    - If you have multiple dates of test, use `dot_1` , `dot_2` , etc. to reflect multiple timepoints.
4. `age_factor` **to refer to age when it is treated categorically and** `age_exact` **when age is calculated from date of birth and date of test**
    - Avoid using just `age` as it is ambiguous as to whether you are treating age as a categorical variable or as a linear variable.
5. `id` **when refering to a participant’s id**
    - Do not use `ID` (no caps!), nor `sid` (some datasets have adult participants and the s is redundant), nor `participant` (unnecessarily long variable name).
6. `condition` **to refer to the condition.**
7. `item` **to refer to the content of an individual trial**
    - For example, the `item` column may include the kinds of animals participants saw (e.g., “cheetah”, “skunk”, etc.) or whether the kid saw a black or white kid in the trial.
    - Here, `item` was chosen over `stimuli` or `trial` because in mixed-effects models, it’s often discussed as including a random effect of item when one wants to account for the stimuli used.
    - If you have multiple elements about the item you wish to encode (or wish to be more specific for clarity), add an `_` with additional content to `item` . E.g., `item_gender` and `item_race` if participants saw both black and white boys and girls as stimuli.
8. `response` **to refer to the column containing the participant’s answer or response when the data is in long format**

9. **`task` : if your data is in long format, consider having a single column `task` for all measures instead of separate `item` and `response` columns per task**
    - E.g., if participants complete two measures (a preference scale and a resource allocation task), set up your data so that you have one column `task` (with two levels: “preference” and “allocation”), a column (or more) for `item` (with however many levels to describe the stimuli), and a single column for `response` .
    - By doing this with your initial dataset that you read in, it’s easy to break up that dataset into smaller data frames for subsequent analyses (e.g., subset data frame by task) and keeps your data very tidy.

Factor names

Like column names, remember to keep these all undercase! It’s easier to work with while coding. When it comes time to make plots, you can specify these in your `ggplot()` to be capitalized.

1. **`gender` : `male` and `female`**
    - **not** boy/girl, **not** m/f, **not** man/woman
    - male/female is neutal to age
2. **`condition` : avoid untransparent acronyms**
    - e.g., “black_first” instead of “bf”
3. **`race` :**
4. **`age_factor` : use numerics (e.g., 4, 5, etc.), *not* words (e.g., four, five, etc.)**

Scale-specific naming conventions

We recycle a lot of measures in our studies, which means we should be recycling naming conventions, too. If you borrow a measure from someone, ask how they labeled things in R. As measure become widely used, they should be added to this document.

Style guide

It’s also important to format your code in a way that is easy to read. Take this example of poorly styled code:

Hide

```
#bad
plot <- ggplot(data=means_diversity) + facet_grid (. ~ age) +geom_point(aes(x = exp_ch
oice, y = response, color=age),shape = 16,size = 7)  +scale_y_continuous(expand = c
(0, 0)) + coord_cartesian(ylim=c(0.9,5.1)) + labs(x = "Experimenter Choice", y =" Aver
age Response") + theme(text=element_text(size = 14), axis.title.x = element_text(size
= 14, face = "bold",margin = margin(t =20, r = 0, b = 0, l = 0)),axis.title.y = elemen
t_text(size = 14, face = "bold",margin = margin(t = 0, r = 20, b =0,l = 0)),strip.tex
t = element_text(size = 12),strip.background = element_blank(),panel.border = element_
rect(color = "black", fill = NA),panel.background = element_rect(fill = NA))
```

The above code is nearly impossible to parse. It’s clear from the beginning that the chunk of code is creating some kind of plot, but it’s hard to tell what kind of plot it is and what exactly it’s plotting because the elements of the code aren’t well spaced.

Here’s the same code but styled properly. In this format, it’s much easier to parse each element of the graph, thus marking it easier to make changes to.

Hide

```
# good
plot <- ggplot(data = means) +
  facet_grid(. ~ age) +
  geom_point(
    aes(x = exp_choice, y = response, color = age),
    shape = 16,
    size = 7) +
  scale_y_continuous(expand = c(0, 0)) +
  coord_cartesian(ylim = c(0.9, 5.1)) +
  labs(
    x = "Experimenter Choice",
    y = " Average Response") +
  theme(
    text = element_text(size = 14),
    axis.title.x = element_text(
      size = 14,
      face = "bold",
      margin = margin(t = 20, r = 0, b = 0, l = 0)),
    axis.title.y = element_text(
      size = 14,
      face = "bold",
      margin = margin(t = 0, r = 20, b = 0, l = 0)),
    strip.text = element_text(size = 12),
    strip.background = element_blank(),
    panel.border = element_rect(color = "black", fill = NA),
    panel.background = element_rect(fill = NA))
```

Spacing

General spacing conventions

There should almost always be spaces around operators (e.g., `<-`, `+`, `-`, `=`, etc.). Commas, like in English, should have spaces after but never before.

The consistent exceptions where spaces around the operator is not required are: `^`, `:`, and `::`.

Spaces are not needed after an open parenthesis `(`, bracket `[`, or brace `{`, or before a closed parenthesis `)`, bracket `]`, or brace `}`.

Hide

```
# good
var_1 <- mean(x, na.rm = TRUE)
var_2 <- a + b - c
var_3 <- c(1:10)
var_4 <- x^2

#bad
var_1<-mean(x ,na.rm=TRUE)
var_2 = a +b- c
var_3 <- c( 1 : 10 )
var_4 <- x ^ 2
```

Spacing in if else statements

There should always be spaces before the conditional (i.e., before the `(`) and after the conditional (i.e., after `)`). The opening brace `{` should always be on the same line as the conditional, and the argument inside the `{}` should always be on its own line and indented.

Additional arguments within the `{}` should be on their own lines.

Hide

```
# good
if (x == 1) {
  print("x is equal to 1") # argument on its own line and indented
}

if (x == 1) {
  print("x is equal to 1")
} else {
  print("x is not equal to 1")
}

if (x == 1) {
  y <- 0 # each argument is on their own line
  print("x is equal to 1")
}

if (x == 1) {
  print("x is equal to 1")
} else if (x == 2) {
  print("x is equal to 2")
}


# bad
if(x == 1)
print("x is equal to 1")  # this will evaluate but styling is poor

if(x == 1){
  print("x is equal to 1")
}else{print("x is not equal to 1")}
```

Spacing long lines

It's generally best to keep code under approximately 80 characters per line. If line starts to get too long, its best to put each argument of a function on its own line.

Hide

```
# good
geeglm(
  response ~ group + age + group * age,
  data = df,
  family = binomial(logit),
  id = participant,
  corstr = "exchangeable"
)

# bad
geeglm(response ~ group + age + group * age, data = data_keep_test, family = binomial
(logit), id = participant, corstr = "exchangeable")

geeglm(response ~ group + age + group * age, data =
       data_keep_test, family = binomial(logit),
       id = participant, corstr = "exchangeable"
       )
```