Naming conventions in R

Abhijit Dasgupta

Naming rules

Why do we need names?

Everything in R is an object that can be named

Unless you name an object, it doesn't stay in the R environment

You name an object using the assignment operator `<-`
I read this as a left arrow

For example

```
spine_data <- readr::read_csv('data/Dataset_spine.csv')
```

Here, we read the CSV file into R, and then save it in the current R session by naming it `spine_data`

Syntactic names

Following these rules will always give you valid object names without the need for additional trouble.

1. Contains letters, numbers, `.` or `_` characters
2. Starts with a letter or `.` not followed by a number
3. Is not a reserved word ()

So `.2way` is not a syntactically valid name

Going wild

You can actually use any name for an object, as long as you encase it in backticks

The name can then include spaces, special characters, start with a number, etc.

Still, don't use reserved words.

You can do

```
`First variable`
```

```
`2 many bikes`
```

But should you?!!

Naming conventions

There are several prevalent naming conventions in R

- snake case, e.g., `my_variable`
- lower camel case, e.g., `myVariable`
- upper camel case, e.g., `MyVariable`
- period separated, e.g., `my.variable`
- lower case, e.g., `myvariable`

Choose one and stick with it. It will save you some sanity.

> The janitor package has a nice function called `clean_names` which will transform the column names in a data.frame to follow a particular naming convention, with the default being snake case.


janitor::clean_names
convert all column names to _*_ case!
(not so awesome column names)
FUN(units)!
Such.Mess!
DOO.OOF_
MESS
Mm.,OOP
clean_names(your data frame + case choice here)
CHOOSE:
✓ snake
lower_camel
big_camel
screaming_SNAKE
+ more!
WAY MORE DEALWITHABLE COLUMN NAMES!
so_much_nicer
_yay_ya_
hope_rest
_wo
_than
_cool
_good_feel_
teddy
snake
HORST 2019

Resources

The following commands, typed in the R console, will get you to the corresponding help files.

- `?make.names`
- `?assign`

Brackets

Brackets

There are three kinds of brackets in R

`[]` are used for extracting elements from arrays, matrices, data frames.

Arrays | Lists

- `x[3]` is the 3rd element of an array `x`
- `d[1,3]` is the element in the 1st row and 3rd column of a matrix/data frame `d`
- `d[2,]` is the entire first row of a matrix/data frame `d`
- `d[,4]` is the entire 4th column of a matrix/data frame `d`
- `d[,'gender']` is the column named *gender* in a *data frame* `d`

Brackets

There are three kinds of brackets in R

`[]` are used for extracting elements from arrays, matrices, data frames.

Arrays | Lists

To extract single elements, you use a double bracket:

- `lst[[3]]` is the third element of the list `lst`
- `lst[['apple']]` is the element named *apple* in a named list `lst`

To extract a sub-list, you use single brackets:

- `lst[1:3]` creates a sub-list with the first 3 elements of `lst`

Brackets

There are three kinds of brackets in R

`()` are used for specifying arguments to functions

- `mean(x)` gives the mean of an array of numbers `x`
- `summary(d)` gives a summary representation of a data frame `d`
- There are several functions used to make the following visualization

```
ggplot(beaches, aes(x = temperature, y = rainfall)) +
  geom_point() +
  geom_smooth() +
  theme_classic()
```

Brackets

There are three kinds of brackets in R

{} are used to contain groups of commands/statements

A conditional statement

```
if (age < 18){
   person <- 'Minor'
} else if (age > 65) {
  person <- 'Senior'
} else {
  person <- 'Adult'
}
```

A function definition

```
my_mean <- function(x, na.rm = T){
  if(na.rm){
    x <- x[!is.na(x)]
  }
  s <- sum(x)
  n <- length(x)
  mn <- s / n # There is a built-in function mean, s
  return(mn)
}
```