Data Science with R
Doing it with Style

Graham.Williams@togaware.com

9th June 2014

Visit http://onepager.togaware.com/ for more OnePageR's.

Data Scientists write lots of programs to manage, manipulate, and model data in a variety of ways. Our code will need to read and understood by others. Here we present guidelines for programming in R (R Core Team, 2014).

A primary aim of any coding and programming style is to ensure consistency and to ease the task for the reader in understanding the code. When we write programs we should **write our code for others to read and to learn from and to share our knowledge**. We should write a story to keep the reader informed and engaged. If they can't follow the codes, then we have not succeeded. Keep it simple.

Of course, we generally write code to have it run on a computer which cares little about coding style— a compiler or interpreter will translate your program into machine code that the computer understands. So very ugly code will work and achieve that goal but why not make it beautiful?

As we work through this chapter, new R commands will be introduced. Be sure to review the command's documentation and understand what the command does. You can ask for help using the `?` command as in:

```
?read.csv
```

We can obtain documentation on a particular package using the *help=* option of `library()`:

```
library(help=rattle)
```

This chapter is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

1 Naming Files

1. Filenames use the `R` extension. This aligns with the fact that the language is unambiguously called "R" and not "r."
   Preferred
   ```
   generatePlots.R
   ```
   Discouraged
   ```
   generatePlots.r
   ```

2. The file name should match the name of the main function defined within the file. For example, if the function defined in the file is `fancyPlot()` then name the file as:
   Preferred
   ```
   fancyPlot.R
   ```
   Discouraged
   ```
   fancy_plot.R
   fancy.plot.R
   fancy_plot.r
   ```

3. R binary data filenames end in ".RData". I have no strong motivation for this except that it conforms to the capitalised naming scheme.
   Preferred
   ```
   weather.RData
   ```
   Discouraged
   ```
   weather.rdata
   weather.Rdata
   ```

4. Other standard file names use lowercase where there is a choice.
   Preferred
   ```
   weather.csv
   ```
   Discouraged
   ```
   weather.CSV
   ```

2 Naming Objects

5. Function names use capitalised verbs, beginning with lowercase.
   Preferred
   ```
   displayPlotAgain
   ```
   Discouraged
   ```
   DisplayPlotAgain
   displayplotagain
   ```
6. Variable names and function argument names use dot separated words.
   Preferred
   ```
   list.of.frames
   lib.cmd
   ```
   Discouraged
   ```
   list_of_frames
   ```
7. Constants are all capitals. This makes them stand out and makes it clear that they should not be changed.
   Preferred
   ```
   MAX.LINES
   ```
   Discouraged
   ```
   const.max.lines
   ```
8. Variables within a dataset (i.e., data frame) are lowercase and use underscore to separate the words. This has the advantage that underscore is acceptable in SQL databases for columns, whereas a period is often used to identify the server/database/table/schema names. We often load/save data in data frames from/to databases. We can use `normVarNames()` from rattle (Williams, 2014) to normalise variables names in this way.
   Preferred
   ```
   min_temp
   wind_gust_speed
   ```
   Discouraged
   ```
   max.pressure
   wind.dir
   WindSpeed
   ```

3 Layout

9. Named arguments in parameter lists do not have a space around the =. I prefer this as visually it ties the named arguments strongly together. This is the only situation where I tightly couple a binary operator. In all other situations there should always be a space around the operator. Another motivation is that it avoids splitting the line between the argument name and the argument value.
   Preferred
   ```
   read.csv(file="data.csv", sep=";", na.strings=".")
   ```
   Discouraged
   ```
   read.csv(file = "data.csv", sep =
            ";", na.strings
            = ".")
   ```

10. Use an indentation of 2. Some argue this is not enough to show the structure when using smaller fonts. If it is an issue for you then 4 is okay. But I would choose a different font instead.

11. Align curly braces. Thus an opening curly brace is on a line by itself. This is a particular difference with many other coding styles. My motivation is that the open and close curly braces are then aligned visually and this provides an added visual check of syntax correctness and visually gives a very strong code block view. The placement of the open curly bracket at the end of the previous line is endemic and in my opinion is bad practise, hiding the opening of a block of code simply to save on having some additional lines (which was only important in my much younger days where we used punched cards or terminals limited to 24 lines). This style also makes it easier to comment out, for example, just the line containing the "while" and still have valid syntax. Don't be afraid of the extra white space—for the human reader, white space is good, and the computer does not care.
    Preferred
    ```
    while (blueSky())
    {
      openTheWindows()
      doSomeResearch()
    }
    retireForTheDay()
    ```
    Discouraged
    ```
    while (blueSky()) {
      openTheWindows()
      doSomeResearch()
    }
    retireForTheDay()
    ```

4 Function Definition Layout

12. Align function arguments by comma. This is a controversial style, but it works to emphasize the arguments and makes it easier to comment out some arguments with little fuss.
    Interesting Option

```
dial.plot <- function(label="UseR!"
                      , value=78
                      , dial.radius=1
                      , value.cex=3
                      , value.color="black"
                      , label.cex=3
                      , label.color="black")
{
  ...
}
```

Traditional

```
dial.plot <- function(label="UseR!", value=78, dial.radius=1, value.cex=3,
                      value.color="black", label.cex=3, label.color="black")
{
  ...
}

dial.plot <- function(label="UseR!",
                      value=78,
                      dial.radius=1,
                      value.cex=3,
                      value.color="black",
                      label.cex=3,
                      label.color="black")
{
  ...
}
```

5 Function Call Layout

13. Similarly when we call the function.
Interesting Option

```
dial.plot(label="UseR!"
          , value=78
          , dial.radius=1
          , value.cex=3
          , value.color="black"
          , label.cex=3
          , label.color="black")
```

Traditional

```
dial.plot(label="UseR!", value=78, dial.radius=1, value.cex=3,
          value.color="black", label.cex=3, label.color="black")

dial.plot(label="UseR!",
          value=78,
          dial.radius=1,
          value.cex=3,
          value.color="black",
          label.cex=3,
          label.color="black")
```

6 Kuhn Checklist

14. Max Kuhn, author of caret (Kuhn *et al.*, 2014) wrote up this checklist and posted it to the R developers mailing list in January 2012. I have paraphrased some of the points here and embellished it a little with my views, but they are quite in sync with Kuhn's views.

    (a) Extend the work of others and avoid redundancy. Reuse others functions, with due credit, to add any missing features.

    (b) For a categorical model builder ensure the target is a factor (like Yes/No) rather than integers (like 1/0). The factor levels should be identified in the resulting model object and the `predict()` function should return predicted classes as factors with the same levels and ordering of levels. Support a `type=` to switch between predicted classes and class probabilities. Use `type="prob"` for probabilities.

    (c) Implement a separate `predict()`, using `predict.class()` where *class* is the class of the object returned by the model builder. Do not use special functions like `modelPredict()`.

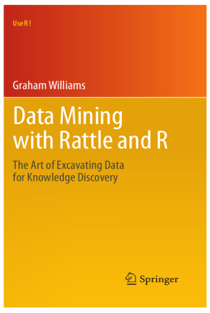
    (d) Provide both a formula interface as in `foo(y~x, data=ds)` and non-formula interface as in `foo(x, y)` to the function. "Formula methods are really inefficient at this time for large dimensional data but are fantastically convenient. There are some good reasons to not use formulas, such as functions that do not use a design matrix (e.g., `cforest()`) or need factors to be handled in a non-standard way (e.g., `cubist()`)."

    (e) "Don't require a test set when model building."

    (f) If not all variables are used in the resulting model, allow the required subset of variables to be provided for `predict()` and not all the original variables, and avoid referencing variables by position rather than name.

7 Further Reading

The Rattle Book, published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from Amazon. Other documentation on a broader selection of R topics of relevance to the data scientist is freely available from `http://datamining.togaware.com`, including the Datamining Desktop Survival Guide.


Use R!
Graham Williams
Data Mining with Rattle and R
The Art of Excavating Data for Knowledge Discovery
Springer

This module is one of many OnePageR modules available from `http://onepager.togaware.com`. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.

I like the guidelines at Google but I have my own idiosyncrasies. The style decisions I have made I motivate above, based on over 30 years of programming in very many different languages. Also see Wikipedia for an excellent summary of many styles.

Rasmus Bååth, in The State of Naming Conventions in R, reviews naming conventions used in R, finding that the initial lower case capitalised word scheme for functions was the most popular, and dot separated names for arguments similarly. This is the style I prefer.

8 References

Kuhn M, Wing J, Weston S, Williams A, Keefer C, Engelhardt A, Cooper T, Mayer Z, the R Core Team (2014). *caret: Classification and Regression Training.* R package version 6.0-29, URL `http://CRAN.R-project.org/package=caret`.

R Core Team (2014). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL `http://www.R-project.org/`.

Williams GJ (2009). "Rattle: A Data Mining GUI for R." *The R Journal*, **1**(2), 45–55. URL `http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf`.

Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery.* Use R! Springer, New York. URL `http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896`.

Williams GJ (2014). *rattle: Graphical user interface for data mining in R.* R package version 3.0.4, URL `http://rattle.togaware.com/`.

This document, sourced from StyleO.Rnw revision 419, was processed by KnitR version 1.6 of 2014-05-24 and took 1.5 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-06-09 10:37:34.