A Computational Analysis of the Dynamics of R Style Based on 94 Million Lines of Code from All CRAN Packages in the Past 20 Years

Chia-Yi Yen, University of Mannheim
Mia Huai-Wen Chang, Akelius Residential Property AB
Chung-hong Chan, University of Mannheim

07-07-2019

Please cite this draft as: Yen, C.Y., Chang, M.H.W., Chan, C.H. (2019) *A Computational Analysis of the Dynamics of R Style Based on 94 Million Lines of Code from All CRAN Packages in the Past 20 Years.* Paper presented at the useR! 2019 conference, Toulouse, France.

Full paper is forthcoming.

Abstract

There are so many programming style variations in R. We have analyzed 94 million lines of R code and quantified the evolution in popularity of 12 style-elements from 1998 to 2018. We attribute 3 main factors that drive changes in programming style: effect of style-guides, effect of introducing new features, and effect of editors. We have identified community-specific programming style variations. For example, there are programming communities which do not use snake_case at all. A consensus in programming style is forming. We have summarised it into a *Consensus-based Style*.

Introduction

R is flexible. For example, one can use <- or = as assignment operators. The following two functions can both be correctly evaluated.

```
sum_of_square <- function(x) {
    return(sum(x^2))
}
```

```
sum_of_square = function(x) {
    return(sum(x^2))
}
```

One area that can highlight this flexible is naming conventions. According to the previous research by Bååth (2012), there are at least 6 styles and none of the 6 has dominated the scene. There are still some other style-elements that R programmers have the freedom to adopt, e.g. whether or not to add spaces around infix operators, use double quotation marks or single quotation marks to denote strings, etc. On one hand, these variations provide programmers with freedom. On the other hand, these variations can confuse new programmers and can have dire effects on program comprehension. Also, incompatibility between programming styles might also affect reuse, maintainability (Elish & Offutt, 2002) and open source collabration (Wang & Hahn, 2017).

Various efforts to standardize the programming style, e.g. Google's R Style Guide, The Tidyverse Style Guide (Wickham, 2017), Bioconductor Coding Style (Bioconductor, 2015), etc. are available. These style guides are

based on the normative assessment of code quality, e.g. style-elements that improve program comprehension (Oman & Cook, 1990). However, we argue that one should first study the current situation, and preferably, the historical development, of programming style variations (PSV) to supplement these standardization efforts. We have undertaken such a task, so that the larger R community can have a baseline to evaluate the effectiveness of those standardization efforts. Also, we can have a better understanding of the factors driving increase and decrease in PSV historically, such that more effective efforts can be formulated.

Analysis

Data Source

In January 2019, we cloned a local mirror of CRAN using the rsync method suggested by the CRAN Mirror HOWTO (Pausakerl, 2016). In our local mirror, it contains all packages as tarball files (.tar.gz). By all packages, we mean packages actively listed online on the CRAN websites and packages delisted for whatever reasons e.g. no maintainer, etc. In this analysis, we include all packages, including those delisted.

In order to facilitate the analysis, we have developed the baaugwo package to extract all R sourcecode and metadata from these tarballs. In this study, only the source code from the */R* directory of each tarball file is included. We have also archived the metadata from the *DESCRIPTION* and *NAMESPACE* files from the tarballs.

In order to cancel out the overrepresentation effect of multiple submissions in a year by a particular package, we have applied the *"one-submission-per-year"* rule to randomly selected only one submission from a year for each package. Unless explicitly notice, we present below the analysis of this "one-submission-per-year" sample. Similarly, unless explicitly notice, the unit of the analysis is **exported function**. The study period for this study is from 1998 to 2018.

Quantification of PSV

Every function in our sample are parsed into a parse tree (or expression) using the parser from the lintr package (Hester & Angly, 2019).

These parse trees were then filtered for lines with function definition and then linted them using the linters from the lintr package to detect for various style-elements. Style-elements considered in this study are:

- **fx_assign**: Use = as assignment operators

```
softplusFunc = function(value, leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L, value, value * 0.01))
    }
    return(log(1L + exp(value)))
}
```

- **fx_opencurly**: An open curly is on its own line

```
softplusFunc <- function(value, leaky = FALSE)
{
    if (leaky)
    {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L, value, value * 0.01))
```

```
    }
    return(log(1L + exp(value)))
}
```

- **fx__infix**: No spaces are added around infix operators.

```
softplusFunc<-function(value, leaky=FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value>0L, value, value*0.01))
    }
    return(log(1L+exp(value)))
}
```

- **fx__integer**: Not explicitly type integers

```
softplusFunc <- function(value, leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0, value, value * 0.01))
    }
    return(log(1 + exp(value)))
}
```

- **fx__singleq**: Use single quotation marks for strings

```
softplusFunc <- function(value, leaky = FALSE) {
    if (leaky) {
        warnings('using leaky RELU!')
        return(ifelse(value > 0L, value, value * 0.01))
    }
    return(log(1L + exp(value)))
}
```

- **fx__commas**: No spaces are added after commas

```
softplusFunc <- function(value,leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L,value,value * 0.01))
    }
    return(log(1L + exp(value)))
}
```

- **fx__semi**: Use semicolons to terminate lines

```
softplusFunc <- function(value, leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!");
        return(ifelse(value > 0L, value, value * 0.01));
    }
    return(log(1L + exp(value)));
}
```

- **fx__t__f**: Use T/F instead of TRUE / FALSE

```
softplusFunc <- function(value, leaky = F) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L, value, value * 0.01))
    }
    return(log(1L + exp(value)))
}
```

- **fx__closecurly**: An close curly is not on its own line.

```
softplusFunc <- function(value, leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L, value, value * 0.01)) }
    return(log(1L + exp(value))) }
```

- **fx__tab**: Use tab to indent

```
softplusFunc <- function(value, leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L, value, value * 0.01))
    }
    return(log(1L + exp(value)))
}
```

We have studied also the naming conventions of all included functions. Using the similar technique of Bååth (2012), we classified function names into the following 7 categories:

- **alllower**: softplusfunc
- **ALLUPPER**: SOFTPLUSFUNC
- **UpperCamel**: SoftPlusFunc
- **lowerCamel**: softPlusFunc
- **lower__snake**: soft__plus__func
- **dotted.func**: soft.plus.func
- **other**: sOfTPluSfunc

The last style-element is line-length. For each R file, we counted the distribution of line-length. In this analysis, the unit of analysis is line.

By not considering line-length, we have studied 10 binary style-elements and one multinomial style-element with 7 categories. Therefore, the possible number of combinations based on these 11 style-elements is: $7 \times 2^{10} = 7168$.

Community-specific variations

On top of the overall patterns based on the analysis of all functions, the community-specific variations are also studied. In this part of the study, we ask the question: do local patterns of PSV exist in various programming communities? To this end, we constructed a dependency graph of CRAN packages by defining a package

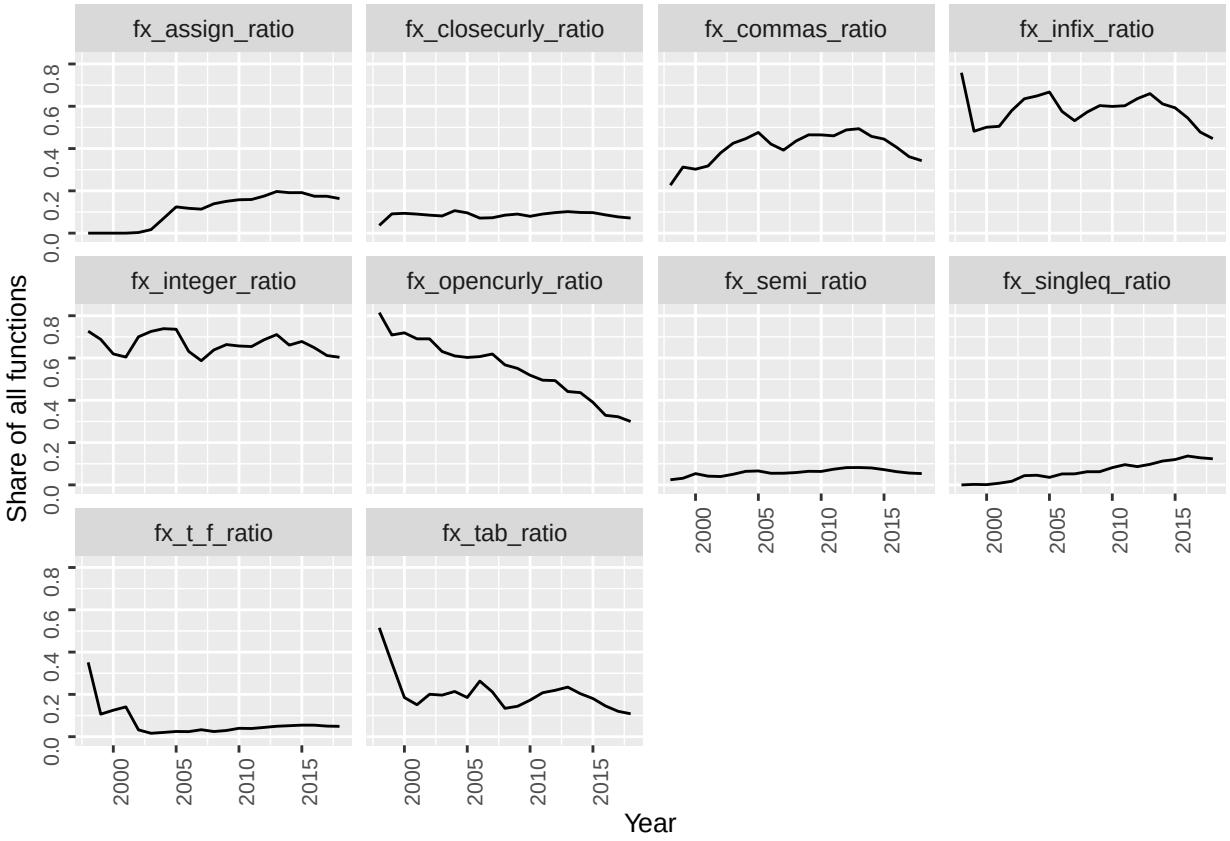

Figure 1: Evolution in popularity of 10 binary style-elements from 1998 to 2018.

as a node and an import/suggest relationship as a directed edge. Communities in this dependency graph were extracted using the Walktrap Community Detection Algorithm (Pons & Latapy, 2005) provided by the igraph package. The step parameter was set at 4 for this analysis. Notably, we analyzed the dependency graph as a snapshot, which is built based on the latest submission of each package in or before 2018.

The 18 largest communities were extracted to study local patterns in PSV.

Results

We studied more than 94 million lines of code from 15530 unique packages. In total, 1898142 exported functions were studied. Figure 1 displays the popularity of the 10 binary style-elements from 1998 to 2008. Some style-elements have a very clear trends towards a majority-vs-minority pattern, e.g. fx_closecurly, fx_semi, fx_t_f and fx_tab. Some styles-elements are instead trending towards a divergence from a previous majority-vs-minority pattern, e.g. fx_assign, fx_commas, fx_infix, fx_integer, fx_opencurly and fx_singleq. There are two style-elements that deserve special scrutiny. Firstly, the variation in fx_assign is a clear example illustrating the effect of introducing a new language element by the R Development Core Team. The introduction of the language feature (= as assignment operator) in R 1.4 (Chambers, 2001) has coincided with the taking off in popularity of such style-element since 2001. Up to now, around 20% of exported functions use such style.

Secondly, the popularity of fx_opencurly shows how a previously established majority style (~80% in late 90s) slowly reduced into a minority, but still very prominent, style (~30% in late 10s).

Similarly, the evolution of different naming conventions is shown in figure 2. This analysis can best be

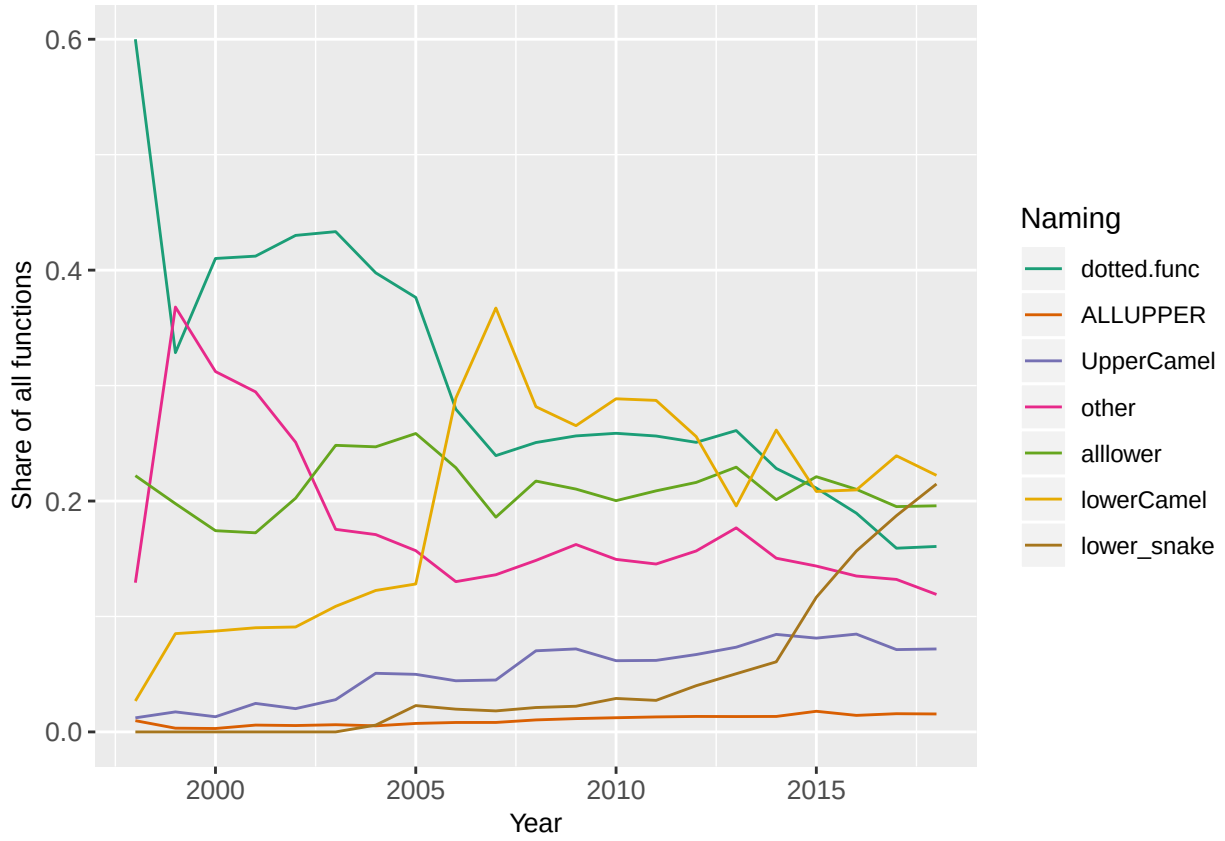

Figure 2: Evolution in popularity of 7 naming conventions from 1998 to 2018.

used to illustrate the effect of style-guides. According to Bååth(2012), dotted.func style is very specific to R programming. This style is the most dominant style in the early days of CRAN. However, multiple style guides advise against the use of dotted.func style and thus a significant declining trend is observed. lower_snake and UpperCamel are the styles endorsed by the Tidyverse Style Guide and the Google's R Style Guide respectively. These two styles see an increasing trend since the 10s, although the growth of lower_snake is relatively more impressive. To our surprise, lowerCamel case, a style endorsed by no style-guide, is currently the most popular naming convention (22.5% in 2018). However, its reign might soon be dethroned by lower_snake (21.5% in 2018) in the near future.

The evolution of line lengths is tricky to be visualized on a 2-D surface. We have prepared an animation to visualize the change in line distribution over the span of 20 years. In this paper, figure 3 shows the snapshot of the change in line length distribution in the range of 40 characters to 100 characters. In general, developers of newer packages write with lesser number of characters per line. Similar to previous analyses with Python programs (e.g. VanderPlas, 2017), artificial peaks corresponding to recommendations from either style-guides, linters, and editor settings are also observed in our analysis. In 2018, the artificial peak of 80 characters (recommended by most of the style-guides and linters such as lintr) is more pronounced for lines with comments but not those with actual code.

Communitiy-based variations

Using the aforementioned community detection algorithm of the dependency graph, 18 large communities were extracted. These communities are named by their applications. Table 1 lists the details of these communities.

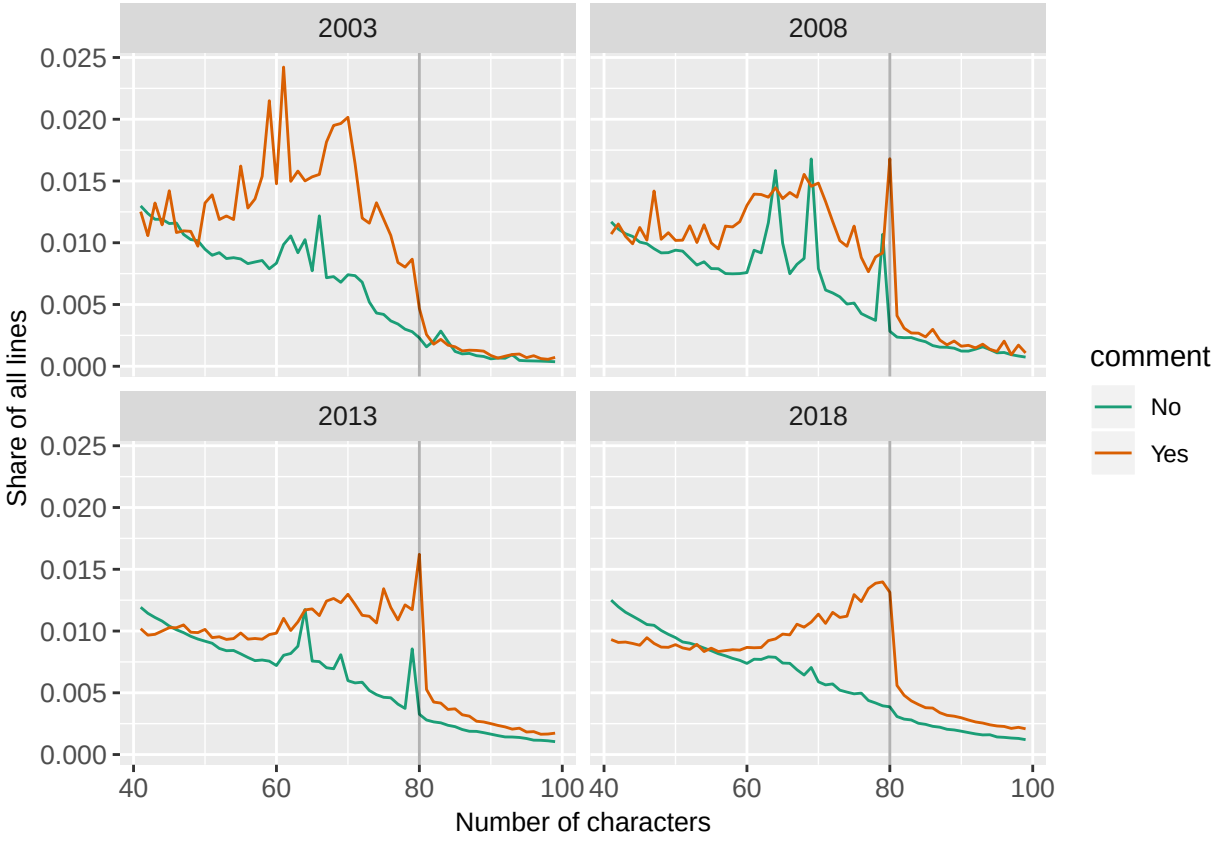

Figure 3: Change in line length distribution: 2003, 2008, 2013 and 2018.

Table 1: The largest 18 communities and their top 3 packages according to eigenvector centrality

Community	Number of Packages	Top 3 Packages
RStudio-related	3426	knitr, testthat, rmarkdown
base	2618	methods, graphics, lattice
Image Plotting	2228	png, rgl, highr
RCpp	677	Rcpp, inline, pkgKitten
GPS and Geography	530	deldir, sp, maptools
Machine learning	319	rpart, nnet, randomForest
Text Analysis	92	stopwords, NISTunits, ISOcodes
Social Network Analysis	53	texreg, network, ergm
Graphics	49	gWidgetstcltk, gWidgets, RGtk2
Graph data structure	48	graph, scagnostics, RBGL
Genetics	44	Biostrings, IRanges, GenomicRanges
Finance	36	RUnit, RcppCCTZ, fingerprint
Insurance and Actuary	34	rsp, Quandl, DiffusionRgqd
Numerical Optimization	30	pbivnorm, rgenoud, Matching
Sparse Matrix	30	registry, slam, Rglpk
Java	28	rJava, RWekajars, openNLP
Neuroscience	27	STAR, rstream, RMTstat
Time, Date, and Money	27	tis, setRNG, tfplot

Using naming convention as an example, there are local patterns in PSV (Figure 4). For example, snake case is the most popular naming convention in the *"RStudio-related"* communitiy as expected because it is the naming convention endorsed by the Tidyverse Style-guide. However, none of the functions exported by the packages from *"Time, Date, and Money"* community uses such convention.

For the binary style-elements, local patterns are also observed (Figure 5). The most salient pattern is the *"Java"* and *"Sparse Matrix"* communities exceptional high usage of tab indentation, probably due to influences from Java or Matlab. Also, the high level in usage of open curly on its own line for the *"Graphics"* is also exceptional.

Discussion

In this study, we study the PSV in 20 years of CRAN packages across two dimensions: 1) temporal dimension: the longitudinal changes in popularity of various style-elements over 20 years, and 2) cross-sectional dimension: the variations among communities of the latest snapshot of all packages. From our analysis, we identify three factors that possibly drive PSV: effect of style-guides (trending of naming conventions endorsed by RStudio and Google), the effect of introducing a new language feature (trending of = usage as assignments after 2001) and effect of editors (the dominance of 80-character line limit).

From a policy recommendation standpoint, our study provides important insight for the R Development Core Team and other stakeholders to improve the current situation of PSV in R. Firstly, the introduction of a new language can have a very longlasting effect on PSV. "Assignments with the = operator" is a feature that introduced by the R Development Core Team to *"increase compatibility with S-Plus (as well as with C, Java, and many other languages)."* (Chamber, 2001) This might be a good intention but it has an unintended consequence of introducing a very persistent PSV that two major style-guides (Tidyverse and Google) consider being a bad style.

Secondly, style-guides, linters, and editors are important standardizers of PSV. Nonetheless, we observe very strong path dependency in programming styles. As indicated by the local patterns of PSV we found in some communities, some package developers are very resistant to these standardizers and keep using their own styles. Having said so, we are not accusing those developers of not following the trendy programming styles. Instead, they follow one of the golden rules: *"if it ain't broke don't fix it"*. Again, from a policy

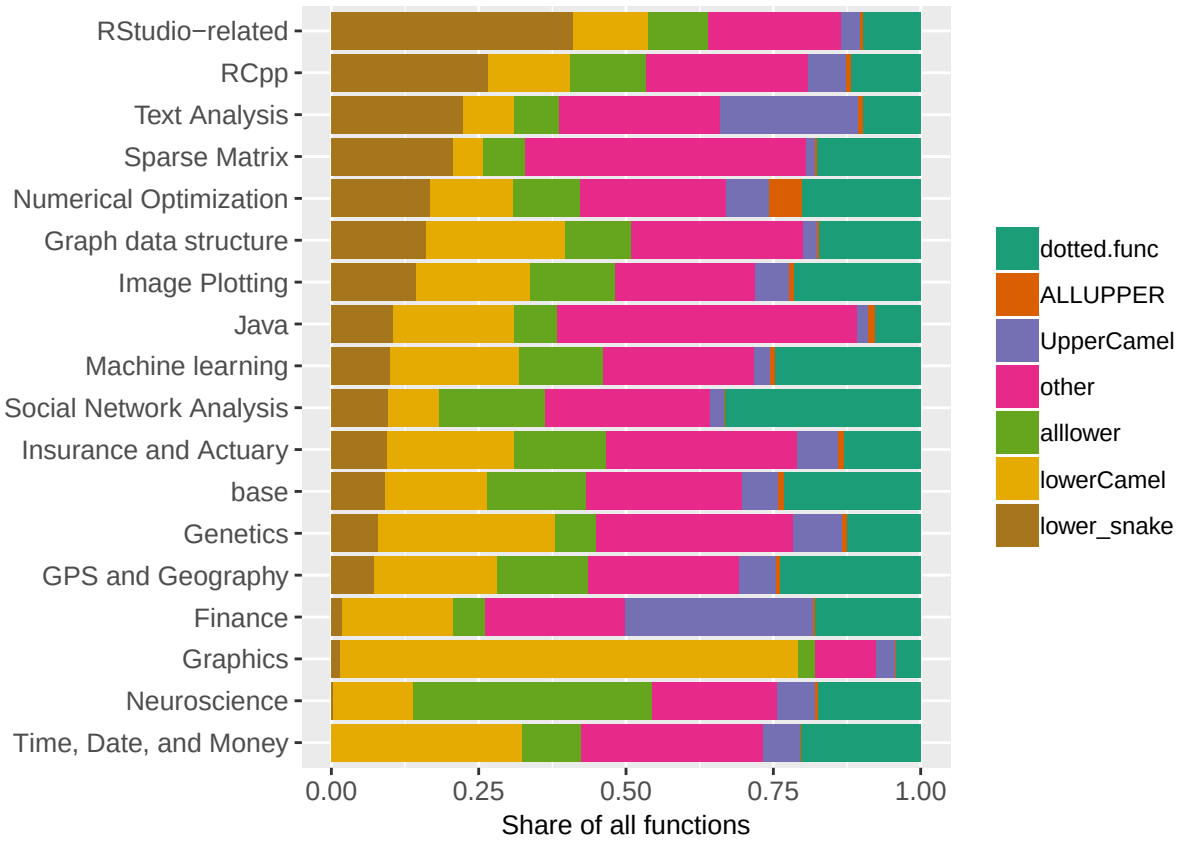

Figure 4: Community-specific distribution of naming conventions among 18 large communities

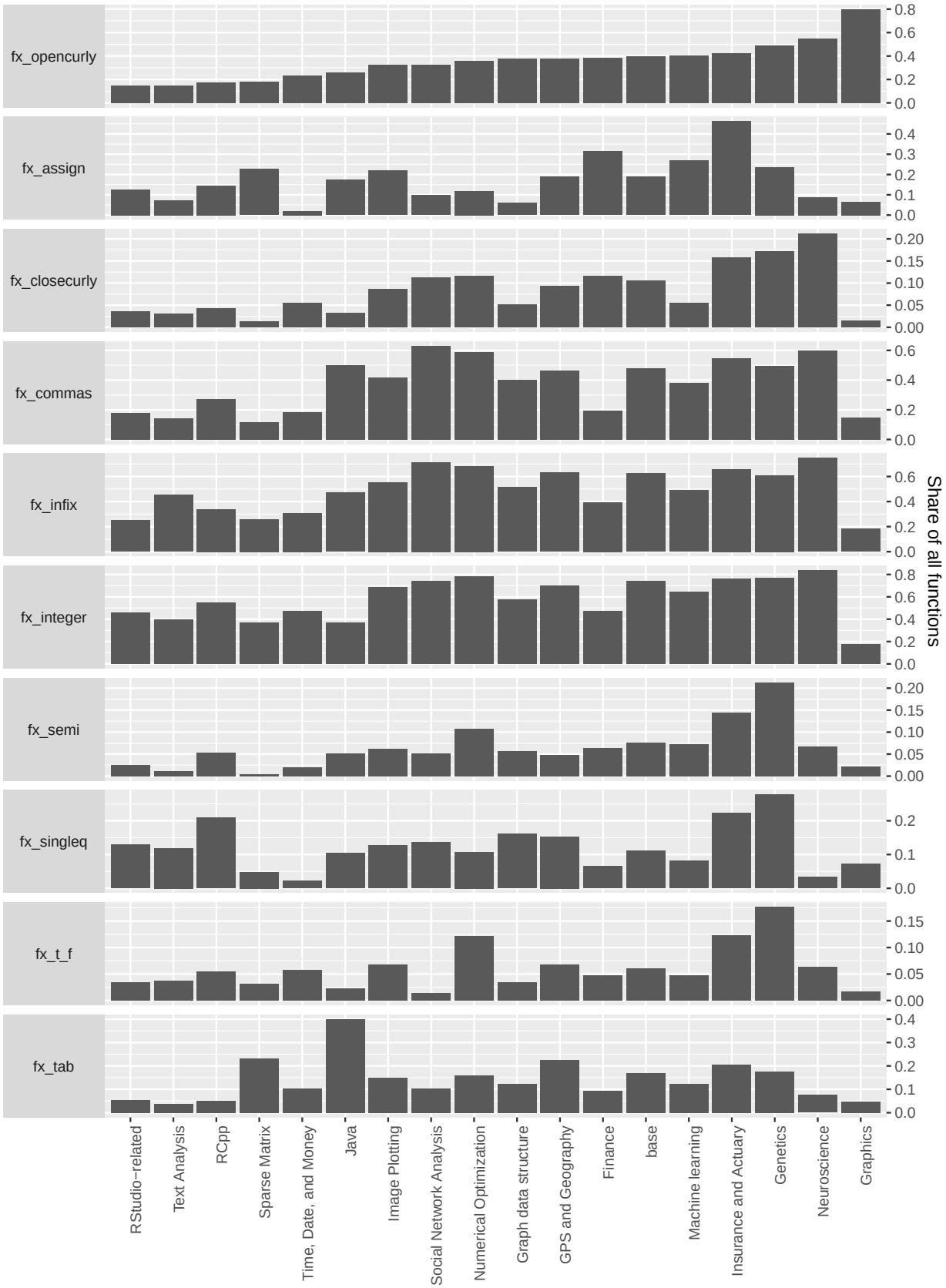

Figure 5: Community-specific distribution of naming conventions among 18 large communities

recommendation standpoint, the existence of local PSV patterns suggests there are many blind spots to the previous efforts in addressing PSV. Style-guide's authors may consider community outreach to promote their endorsed styles, if they want other communities adopt their styles.

Our analysis also opens up an open question: should R adopt an official style-guide akin the PEP-8 of the Python Software Foundation (van Rossum, et al. 2001)? There are of course pros and cons of adopting such official style-guide. As written by Christiansen (1998), *"style can easily become a religious issue."* It is not our intention to meddle in this "religious issue". If such an effort would be undertaken by someone else, we suggest the following *consensus-based style*. We must stress here that this *consensus-based style* is only the most popular style based on our analysis, i.e. the *Zeitgeist* (the spirit of the age). We have no guarantee that this style can improve clarity or comprehensibility. The following is an example of a function written in such *consensus-based style*.

```
softplusFunc <- function(value, leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0, value, value * 0.01))
    }
    return(log(1 + exp(value)))
}
```

In essense,

- Use lowerCamel or snake case
- Use <- to assign, don't use =
- Add a space after commas
- Use TRUE / FALSE, don't use T / F
- Put open curly bracket on same line then a newline
- Use double quotation mark for strings
- Add spaces around infix operators
- Don't terminate lines with semicolon
- Don't explicitly type integers (i.e. 1L)
- Put close curly bracket on its own line
- Don't use tab to indent

As a final remark: although enforcing a consistent style can improve open source collaboration (Wang & Hahn, 2017), one must also bear in mind that these rules might need to be adjusted sometimes to cater for programmers with special needs. For example, using spaces instead of tabs for indentation can make code not assessible to visually impaired programmers (Mosal, 2019).

About the authors

Ms. **Chia-Yi Yen** is currently a finance PhD student at the Mannheim Business School. Prior to her doctoral studies, she worked as a financial engineer in hedge fund industry and a data science consultant for financial institutions. Besides, she is also listed as one of the inventors of three Taiwanese patents (under review) of innovative Fintech applications, a translator of two children's novels on computer science, the co-founder of R-Ladies Taipei — a local community helping women learn data science and R language — and one of the Microsoft Most Valuable Professionals from January 2017 to June 2019.

Ms. **Huai-Wen Chang** works as a data scientist in Berlin. She has a formal background in computer science and mathematics. She has worked in the AI area for years: starting from conducting AI research in computer games and customer services analysis in e-commerce, creating computer vision solutions for start-ups, to innovating new data science solutions for a real estate company. With her technical expertise in

theory of computer games, computer vision, and deep learning, she has been recognized as Microsoft Most Valuable Professional since 2017.

Dr. **Chung-hong Chan** is a research associate at the Institue of Media and Communication Studies and project staff at the Mannheim Center for European Social Research, the University of Mannheim. He earned his doctorate in communication studies at the University of Hong Kong and studied biostatistics and epidemiology at the Chinese University of Hong Kong. Previously, he has worked in a hospital as a biostatistician for 10 years. His research interests are cyberbalkanization, polarization, platform interventions (e.g. online censorship), text mining, social network analysis and meta-analysis/metaregression. He is the corresponding author of this article. E-mail him at: chung-hong.chan@mzes.uni-mannheim.de

References

Bioconductor. (2015) Coding Style. http://bioconductor.org/developers/how-to/coding-style/

Bååth, R. (2012). The state of naming conventions in R. The R journal, 4(2), 74-75.

Chambers, J. (2001). Assignments with the = Operator. http://developer.r-project.org/equalAssign.html

Christiansen, T. (1998). Perl Style: Everyone Has an Opinion. https://www.perl.com/doc/FMTEYEWTK/style/slide1.html/

Csardi, G., & Nepusz, T. (2006). The igraph software package for complex network research. InterJournal, Complex Systems, 1695(5), 1-9.

Elish, M. O., & Offutt, J. (2002). The adherence of open source java programmers to standard coding practices. https://pdfs.semanticscholar.org/bf08/9bc9897253f5d92202a5986669365ee5e9bc.pdf

Google's R Style Guide. https://google.github.io/styleguide/Rguide.xml

Hester, J., Angly, F. (2019). A 'Linter' for R Code. https://github.com/jimhester/lintr/

Mosal, C. (2019). Nobody talks about the real reason to use Tabs over Spaces. https://www.reddit.com/r/javascript/comments/c8drjo/nobody_talks_about_the_real_reason_to_use_tabs/

Oman, P. W., & Cook, C. R. (1990). A taxonomy for programming style. Proceedings of the 1990 ACM Annual Conference on Cooperation - CSC '90. doi:10.1145/100348.100385

Pausakerl, P. (2016). CRAN Mirror HOWTO/FAQ. https://cran.r-project.org/mirror-howto.html

Pons, P., & Latapy, M. (2005, October). Computing communities in large networks using random walks. In International symposium on computer and information sciences (pp. 284-293). Springer, Berlin, Heidelberg.

Wickham, H. (2017). The tidyverse style guide. https://style.tidyverse.org/

van Rossum, G., Warsaw, B., Coghlan, N. (2001). PEP 8 – Style Guide for Python Code. https://www.python.org/dev/peps/pep-0008/

VanderPlas, J. (2017). Exploring Line Lengths in Python Packages. https://jakevdp.github.io/blog/2017/11/09/exploring-line-lengths-in-python-packages/

Wang, Z., & Hahn, J. (2017). The Effects of Programming Style on Open Source Collaboration. http://repository.ittelkom-pwt.ac.id/2872/1/The%20Effects%20of%20Programming%20Style%20on%20Open%20Source%20Collaboration.pdf