An Introduction to Pubprint

Rudolf Siegel

Version 0.2.1
May 24, 2016

Abstract

Pubprint is an extension for the R programming language. This package takes the output of several statistical tests, collects the characteristic values and transforms it in a publish-friendly pattern. Currently only the APA (American Psychological Association) style is supported with output to HTML, LaTeX, Markdown and plain text. The pubprint package is easily customizable, extendable and can be used well with knitr. Additionally pubprint offers a memory system that allows to save and retrieve results of computations.

Contents

1 Introduction

The main function of pubprint is to convert the results of statistical computations into R in a publishable manner. This is possible for several publication styles (right now only for APA style) and output formats (like LaTeX, HTML, Markdown and plain text). Adapting the publication style or the output format to your own needs is quite easy.

Furthermore pubprint offers a smart memory system, meaning that you can simply store and retrieve results of computations. This can be used when working together with knitr. You might like to do all computations in one place or R file and call later only the results (works even with knitr inline R code).

In this document it is explained first, how the results of computations can be formatted. Then the memory system is expounded and how to change the general options and adapting the functions to your own needs.

2 Formatting statistical results

2.1 First steps

A specific example might give a closer look how the formatting of statistical computations works:

```
set.seed(4711) # better reproducibility

library("pubprint") # load library
pp_opts_out$set(pp_init_out("plain")) # better readability in this document

a <- rnorm(40)
b <- a + .3

pprint(t.test(a, b))

## [1] "(M_x=-0.01,M_y=0.29,t[78]=-1.26,p=.210)"

pprint(cor.test(a, b))

## [1] "(r=1.00,p<.001)"
```

In this example, the random seed for this document is set to a fixed number to get a better reproducibility. In a next step we load the pubprint package and change the output format from LaTeX (default) to plain text. This ensures better readability in this document. Then, simply the result of a t-test is given to the pprint function. As a result we get the formatted output according to APA style in plain text output format (surrounded by brackets). And that's it. Additionaly, as you can see, pubprint is also capable of handling correlations.

2.2 Arguments of pprint function

To further adapt the output to your needs, you can change the arguments of the pprint function:

```
args(pubprint:::pprint)

## function (x, format, ..., concat = TRUE, mmode = pp_opts$get("mmode"),
##     separator = pp_opts$get("separator"), toClip = FALSE)
## NULL
```

Here is an example what happens if the arguments are modified:

```
pprint(t.test(a, b))

## [1] "(M_x=-0.01,M_y=0.29,t[78]=-1.26,p=.210)"

pprint(t.test(a, b),
       concat = FALSE)

## [1] "(M_x=-0.01)"   "(M_y=0.29)"    "(t[78]=-1.26)" "(p=.210)"

pprint(t.test(a, b),
       separator = NULL)

## [1] "M_x=-0.01,M_y=0.29,t(78)=-1.26,p=.210"

pprint(t.test(a, b),
       concat = FALSE,
       separator = NULL)

## [1] "M_x=-0.01"   "M_y=0.29"    "t(78)=-1.26" "p=.210"

pprint(t.test(a, b),
       mmode = FALSE)

## [1] "(M_x=-0.01,M_y=0.29,t[78]=-1.26,p=.210)"

pprint(pprint(t.test(a, b),
              concat = FALSE,
              separator = NULL)[c(1, 2)])

## [1] "(M_x=-0.01,M_y=0.29)"
```

The first command is the old and well-known one. When passing `concat = FALSE` the single parts of a statistical output are not concatenated and `separator` specifies how the output is separated from the surrounding text (e.g. in your LaTeX document). These two options can be used to extract specific

items from a result. In the example above this is done with the two estimates (see last command). Maybe there will be a better solution in a later release. Altering `mmode` has no implication here. `mmode`, a logical, defines whether the output is set in math mode or not (e.g. in a LaTeX document). By changing the format argument you can define how to format your results.

```
pprint(t.test(a, b),
       format = "object")

## [[1]]
##
##  Welch Two Sample t-test
##
## data:  a and b
## t = -1.2649, df = 78, p-value = 0.2097
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.7721816  0.1721816
## sample estimates:
##  mean of x  mean of y
## -0.0113524  0.2886476

pprint(t.test(a, b),
       format = "t.test")

## [1] "(M_x=-0.01,M_y=0.29,t[78]=-1.26,p=.210)"

pprint(t.test(a, b),
       format = "chisq")

## [1] "(t[78,N=0]=-1.26,p=.210)"
```

If argument `format = "object"` is given, the object list is returned, but you can specify an internal style function as well. Choosing `t.test` does not change anything, because this internal style function is selected for a t-test by default. Changing this argument makes sense when there may be different desired outputs for the same statistical value (for example you could print a single number as a rounded number or treat it like a p-value, etc.). Obviously choosing `chisq` does not make sense in this case.

Even the internal style functions are not exported to user environment, you can display the documentations. You can find them in the manual or with `?style.<publication style>.<function>`. As there is only the APA publication style right now, you can replace `<publication style>` always with `apa`. Using tabulator key you will find a list of all internal style functions. The names should be self-explanatory. If you haven chosen the appropriate function, you can take the name (only the name of the function without `style.apa.`) as an argument for `format`.

2.3 Pass further information to pprint

If you want to pass additional informations to pprint, you can do that by creating a list. For example you could add Cohen's d to a t-test:

```
pprint(list(t.test(a, b), 0.2828363))

## [1] "(M_x=-0.01,M_y=0.29,t[78]=-1.26,p=.210,d=0.28)"
```

Check the documentation of the internal style functions to determine which further objects are processed by the these functions. Unused list items are ignored.

2.4 Passing arguments to internal style functions

The argument list of pprint has an ellipsis (. . .), that offers the possibility to pass arguments to the internal style functions. For example you can suppress the estimates of a t-test or alter their names (check the documentations for appropriate arguments):

```
pprint(t.test(a, b), print.estimate = FALSE)

## [1] "(t[78]=-1.26,p=.210)"

pprint(t.test(a, b), estimate.names = c("control", "treatment"))

## [1] "(control=-0.01,treatment=0.29,t[78]=-1.26,p=.210)"
```

3 Using the memory functions

3.1 First steps

To use the memory functions of pubprint, you have to create a pubprint object before:

```
ppo <- pubprint()
```

Then, you can store and retrieve some computations in/from the pubprint object

```
push(ppo) <- t.test(a, b)
pull(ppo)

## [1] "(M_x=-0.01,M_y=0.29,t[78]=-1.26,p=.210)"
```

Obviously `pull()` has not simply returned the R object, instead it called `pprint()`, too. You can pass all arguments of `pprint()` to the pull function. For example `format = "object"` will simply return the object, what can be used for plot objects, etc. Objects without a known internal style function will be handled in the same manner.

3.2 Arguments of the memory functions

The push and pull functions have the following arguments:

```
args(pubprint:::`push<-.pubprint`)

## function (x, item, add = FALSE, n = 1, ..., value)
## NULL

args(pubprint:::pull.pubprint)

## function (x, item = 1, remove = pp_opts$get("removeItems"), ...)
## NULL
```

Argument `x` corresponds to a pubprint object and `value` to the assigned value (like the result of the t-test). Pubprint offers two different possibilities to save your results. First, there is a enumerated list, that is used as a pipe. To be more precise, the first saved value will be first returned as well. Second, there is a named list, that is used as a memory. Storing and retrieving to this memory works only by naming the desired value. In the next subsection this is explained in more detail.

3.3 Using the pipe and the named memory

The advantage of the pipe is a very simple interface. In the order the calculations are saved, they will be retrieved again. In contrast the named memory offers more flexibility and security of retrieving the correct result with slightly more effort. Even if you add at a later point a statistical computation, you do not have to care about the order. If `item` is a numeric, your results will be saved in the pipe. If it is a character, it will be saved in the named memory. You can use both systems concurrently.

Pay attention, that `pull()` has a `remove` argument. It specifies whether items are removed from pipe only (`"pipe"`), memory only (`"memory"`, the default), never (`FALSE`) or always (`TRUE`) on retrieving (there is also a general option). Here is a first example:

```
# save items in pipe and named memory
push(ppo) <- t.test(a)
push(ppo) <- t.test(a, b)
push(ppo, item = "i1") <- t.test(a, b + .2)
```

```
# retrieve items from pipe
pull(ppo) # item is removed from pipe

## [1] "(M_x=-0.01,t[39]=-0.07,p=.946)"

pull(ppo) # here as well

## [1] "(M_x=-0.01,M_y=0.29,t[78]=-1.26,p=.210)"

pull(ppo) # error because there are no more items in pipe

## Error in pull.pubprint(ppo):  subscript out of bounds

# retrieve items from named memory
pull(ppo, item = "i1") # item is not removed

## [1] "(M_x=-0.01,M_y=0.49,t[78]=-2.11,p=.038)"

pull(ppo, item = "i1", remove = TRUE) # item is removed

## [1] "(M_x=-0.01,M_y=0.49,t[78]=-2.11,p=.038)"

pull(ppo, item = "i1") # error, item does not more exist

## Error in pull.pubprint(ppo, item = "i1"):  item "i1" not available
```

3.4 Append additional information

As seen in the section about `pprint()` it may be useful to add further information to a statistical output. Therefore the push function owns the `add` argument. By specifing `add = TRUE`, you add an object to an existing item. The corresponding item is either specified by `item` (pipe or named memory) or `n`. While `item` specifies an absolute position in the pipe or an item of the named memory, `n` addresses a relative position in the pipe, counting backwards. So `n = 1` (default) corresponds to the last added item in the pipe, `n = 2` the second last item, and so on. The `n` argument is ignored, when `item` is specified.

```
push(ppo) <- t.test(a)
push(ppo) <- t.test(a, b)
# add to last pipe item (n = 1 is default)
push(ppo, add = TRUE) <- 0.2828363

pull(ppo) # retrieve one way t-test

## [1] "(M_x=-0.01,t[39]=-0.07,p=.946)"

pull(ppo) # retrieve two way t-test with Cohen's d
```

```
## [1] "(M_x=-0.01,M_y=0.29,t[78]=-1.26,p=.210,d=0.28)"
```

4 Changing general options

4.1 Change publication style or output format

You can change the output format by calling (supported are `"latex"`, `"markdown"`, `"html"` and `"plain"`):

```
pp_opts_out$set(pp_init_out("html"))
pprint(t.test(a, b))

## [1] "(<math xmlns=\"&mathml;\"><mfenced open=\"\" close=\"\" separators=\",\"><mrow><msub

pp_opts_out$set(pp_init_out("latex"))
pprint(t.test(a, b))

## [1] "(\\ensuremath{M\\ifmmode_{x}\\else\\textsubscript{x}\\fi=-0.01,M\\ifmmode_{y}\\else\
```

Output format is automatically determined if pubprint is used in a document that is processed by knitr. Currently there are no supported publication styles except APA.

4.2 Change package defaults

General options can be changed with `pp_opts` (see `?pp_opts` for more information):

```
pp_opts$set(mmode = FALSE)
```

5 Adapting style and internal output functions

Changing the provided style or internal output functions is quite easy. You can write your own function and replace a supplied function (or add a new one) with it through `pp_opts_out` or `pp_opts_style`.

```
myttest <- function(...) return("Hello World!")
pp_opts_style$set("t.test" = myttest)
pprint(t.test(a, b))

## [1] "(Hello World!)"
```

If you have written a new style function, you have to call it by specifying the `format` argument of the `pprint` function:

```
# set a new function but do not overwrite a new one
myttest <- function(...) return("Hello World!")
pp_opts_style$set("new-t.test" = myttest)
pprint(t.test(a, b))

## [1] "(M_x=-0.01,M_y=0.29,t[78]=-1.26,p=.210)"

pprint(t.test(a, b), format = "new-t.test")

## [1] "(Hello World!)"
```

Please, consider a contribution to this package if you have written functions that could be useful to other people. You will find more information on the website of this package.