R2HTML Library: Formatting HTML Output on the Fly or by Using a Template Scheme

Eric Lecoutre

Statistics are not only theory and methodology, but also computing and communication. Applied statistician knows they have to pay attention to the last step of an analysis: the report. A very elegant way to handle the final report with R is to use the wonderfull Sweave system [1] in tools package: not only it allows professional quality reports by using LaTeX but also it stores used code within the document, which is very usefull when coming back to the analysis some times after. Nevertheless, this solution is not applicable to every case, as the user may not know LaTeX or may need an other format to communicate with his client. In effect, in many cases, the client is waiting for a report he may edit to add some details. RTF format is ideal for this communication, asi it could be open on many systems and allows some formatting enhancements (bold, tables, ...). Nevertheless, it's not easy to produce and dont allow to embed graphs. An other universal format which can desserve our communication goal is HTML: it is light, readable on any plattform, editable, and allows graphs. Moreover, it could easily be exported to other formats.

This documents describes the `R2HTML` package which provide some support for writing formatted HTML output. Alhtough having some knowledge about HTML is preferable to personalize outputs, the user may use this package to obtain results without this knowledge. We will build different web pages, the reader could found them at the following address: http://www.stat.ucl.ac.be/ISpersonnel/lecoutre/R2HTML/P

Introduction to HTML and `R2HTML` package

In accordance with the W3 Consortium, HTML is the lingua franca for publishing hypertext on the World Wide Web. It is a non-proprietary format based upon SGML, and can be created and processed by a wide range of tools, from simple plain text editors - you type it in from scratch - to sophisticated WYSIWYG authoring tools. HTML uses tags such as `<b>` and `</b>` to structure text into headings, paragraphs, lists, hypertext links etc. Opening tags localize the beginning of the application of format (`<b>`: begin bold), format which ends with the corresponding closing tag (`</b>` for bold text).

Thus, in order to write basic HTML documents, the only required knowledge is the list of existing tags and their functionality. Then, here is the structure of a (rather basic) HTML document:

```
<html>
<h1>My first HTML page </h1>
<p>This is some basic text with a
   <b>bold</b> word.</p>
<p>It uses h1, and p tags which allows to create
   a title and to define a paragraph</p>
</html>
```

Now, we have a very easy way to obtain our first webpage from R: simply use `cat` function to write text to a external file. In the following example, see how we call severall times `cat` function with `append` argument set to `TRUE` to add information to the page.

```
> htmlfile = file.path(tempdir(),
+     "page1.html")
> cat("<html><h1>My first HTML page from R</h1>",
+     file = htmlfile)
> cat("\n<br>Hello Web World!",
+     append = TRUE, file = htmlfile)
> cat("\n</html>", append = TRUE,
+     file = htmlfile)
```

There now, we have all that we want to advance to the next step. The library `R2HTML` is simply a list of wrapper fonctions that calls `cat` in such a way to write HTML codes. The main functions are:

- `HTML()` Main generic function for which subfunctions are defined for all classic classes (matrix, lm, summary, ...).
- `HTMLbr()` Insert a `<br>` HTML tag, a break return code to start a new line.
- `HTMLhr()` Insert a `<hr>` HTML tag, a horizontal rule to separate pieces of text.
- `HTMLInsertGraph()` Insert a `<img>` HTML tag to add an existing graph to the report. The graph should have been created before in a suitable web format such as GIF, JPEG or PNG.

Basically, the `R2HTML` library contains a generic `HTML()` function which behaves like internal `cat()`. Common arguments are `append` and `file`, which default value is set by the hidden variable `.HTML.file`. Thus, it is convenient to begin to set the value of this variable, such as we can omit the file argument thereafter:

```
> .HTML.file = file.path(tempdir(),
+     "page2.html")
> HTML(as.title("Title of my report"),
+     append = FALSE)
> HTMLhr()
> HTML("3 dimensions identity matrix")
> HTML(diag(3))
```

Generating a HTML output on the fly

The first way to use `R2HTML` library is to generate an automatic HTML output during an interactive session. This is specially convenient for courses, as students can keep a log of the commands they asked for and their output, with graphs incorporated. To manage a dynamic session, two commands are present:

- `HTMLStart()`
- `HTMLStop()`

Here is a sketch of the way those commands work behind. When calling `HTMLStart()`, severall actions are performed:

- Three HTML files are written in the temporary directory of the session. The main (index.html) refers to the two others, by incorporating them within HTML frames. It allows to have at the left part of the screen the commands, and at the right one the corresponding outputs.
- A new environment, called `HTMLenv` is created, where some internal variables are stored. Those variables allows to store path to the output files, and to know which action has been done with the latest command.
- A new `fix` function is assigned to global environment, masking the internal one. When calling "new" `fix`, a boolean is set to `TRUE` in the `HTMLenv` environment, to know that the last action was to edit a function.
- `addTaskCallback` is called, adding a task to each submitted command. This task, handled by the function `ToHTML` (not user visible) is the main part of the thing, as it exports the last manipulated object. In this function, one tests if the boolean indicates that a function has been edited, and exports this one if this is the case. When doing so, a new file is created, so that at the end one can keep all the versions of the function at the different steps of work.
- Finally, as a side effect, the prompt is changed to `HTML>` in order to know that output are currently redirected.

From now on, every command is treated two times: once it is evaluated, the results goes on through `ToHTML` function which writes it to the HTML output.

As there is no convenient way to know when a graph has been performed (or modified) and as it's not wanted to export every graph, the user has to explicitely ask for the insertion of the current active graph to the output, by calling `HTMLplot()` function.

When desired, a call to the function `HTMLStop()` stops the process and remove all temporary created variables.

This examples only work in a interactive session with the RGUI. Simply copy the portion of code and paste it.

```
> HTMLStart(filename = "dynamic",
+     echo = TRUE)

 *** Output redirected to directory:  C:\DOCUME~1\lecout
 *** Use HTMLStop() to end redirection.[1] TRUE

HTML> sqrt(pi)

[1] 1.772454

HTML> x = rnorm(10)
HTML> x^2

 [1] 1.484002e+00 1.992549e+00
 [3] 1.319326e+00 9.624633e-01
 [5] 1.995133e+00 5.706354e-05
 [7] 2.737246e-01 2.105211e-03
 [9] 3.308138e-01 1.293306e-01

HTML> myfunction = function(x) return(summary(x))
HTML> cat("\n### try to fix the function: fix(myfunctio

### try to fix the function: fix(myfunction)

HTML> myfunction(x)

   Min. 1st Qu.  Median    Mean 
-0.3596  0.1365  0.7781  0.6872 
3rd Qu.    Max. 
 1.2010  1.4120 
```

```
HTML> plot(x)
HTML> HTMLplot()

[1] TRUE


HTML> HTMLStop()


[1] "C:\\DOCUME~1\\lecoutre\\LOCALS~1\\Temp\\Rtmp28524/dynamic_main.html"
```

Creating personalised reports

Let's begin with a simple analysis

For the user who knows some basics on HTML, the package `R2HTML` offers all necessary stuff to developp fast routines to create one own reports. Nevertheless, without knowing HTML codes, we can still easily create reports. What we propose here is a so-called template approach. Let's imagine we have to perform a daily analysis which output consists in some summary tables and graphs.

First, we gather all the stuff necessary to write the report in a list object. An easy way is to create a user function `MyAnalysis` which output this list. Moreover, we assign a user-defined class for this object.

```
> MyAnalysis = function(data) {
+     table1 = summary(data[,
+         1])
+     table2 = mean(data[, 2])
+     dataforgraph1 = data[,
+         1]
+     output = list(tables = list(t1 = table1,
+         t2 = table2), graphs = list(d1 = dataforgraph1))
+     class(output) = "MyAnalysisClass"
+     return(output)
+ }
```

Then, we provide a new `HTML` function, based on the structure of our output object and corresponding to it's class:

```
> HTML.MyAnalysisClass = function(x,
+     file = "report.html", append = TRUE,
+     directory = getwd(), ...) {
+     file = file.path(directory,
+         file)
+     cat("\n", file = file,
+         append = append)
+     HTML.title("Table 1: summary for first variable",
+         file = file)
+     HTML(x$tables$t1, file = file)
+     HTML.title("Second variable",
+         file = file)
+     HTML(paste("Mean for second variable is: ",
+         round(x$tables$t2,
+             3), sep = ""),
+         file = file)
+     HTMLhr(file = file)
+     png(file.path(directory,
+         "graph1.png"))
+     hist(x$graphs$d1, main = "Histogram for 1st varia
+     dev.off()
+     HTMLInsertGraph("graph1.png",
+         Caption = "Graph 1 - Histogram",
+         file = file)
+     cat(paste("Report written: ",
+         file, sep = ""))
+ }
```

If we want to write create the report, we simply have to do the following:

```
> data = matrix(rnorm(100), ncol = 2)
> out = MyAnalysis(data)
> setwd(tempdir())
> HTML(out, file = "page3.html")

Report written: C:/DOCUME~1/lecoutre/LOCALS~1/Temp/Rtmp2
```

The interest if that we store all the analysis raw material with on object, and that we dissociate the process that creates the report. If we keep all our objects, it's easy to modify the `HTML.MyAnalysisClass` function and to generate again all reports.

Template scheme to complete the report

What we write before is not a real HTML file, as it even doesn't contain standard headers `<html><head>...</head><body>...` and so on. We see two differents ways to handle this, each one having it's advantages/disadvantages. For this personalisation, it's mandatory to have some knowledge of HTML.

At first, we could have a pure R approach, by adding to our report two functions, such as:

```
> MyReportBegin = function(file = "report.html",
+     title = "My Report Title") {
+     cat(paste("<html><head><title>",
+         title, "</title></head>",
+         "<body bgcolor=#D0D0D0>",
+         "<img=logo.gif>", sep = ""),
+         file = file, append = FALSE)
+ }
> MyReportEnd = function(file = "report.html") {
+     cat("<hr size=1></body></html>",
+         file = file, append = TRUE)
+ }
```

```
> MyReport = function(x, file = "report.html") {
+     MyReportBegin(file)
+     HTML(x, file = file)
+     MyReportEnd(file)
+ }
```

Then, instead of calling `HTML` function directly, we consider it at an internal function and we rather call `MyReport`.

```
> out = MyAnalysis(data)
> MyReport(out, file = "page4.html")
```

```
Report written: C:/DOCUME~1/lecoutre/LOCALS~1/Temp/Rtmp28524/page4.html
```

The advantage is that we can even personalize the head and the footer of our report depending on R variables such as the name of the data or whatever we want.

If we dont need to go that further and only need hard coded contents, we can build the report based on two existing files `header.html` and `footer.html`, which could be modified to suit our needs. To work properly, the following piece of code supposes that those two files exist in the working directory:

```
> MyReport = function(x, file = "report.html",
+     headerfile = "header.html",
+     bottomfile = "footer.html") {
+     header = readLines(headerfile)
+     cat(paste(header, collapse = "\n"),
+         file = file, append = FALSE)
+     HTML(x, file = file, append = TRUE)
+     bottom = readLines(bottomfile)
+     cat(paste(bottom, collapse = "\n"),
+         file = file, append = TRUE)
+ }
```

Going a step further with CSS

Cascading Style Sheets (CSS) compensates for some laks of HTML language. CSS add to each standard HTML element it's own style, which is defined in an external file. Thus, when the house-style book of the report has to change, it is enough to modify the definition of classes in an only spot to change the look of all reports - past or to come - that rely on the defined classes.

The use of cascading style sheets allow:

- a homogeneous look for all generated reports
- to change the look of a bunch of reports at one time
- to have lighter reports, as formatting instructions are separated
- a faster download and viewing of reports

All details about CSS specification could be found on the World Wide Web consortium: http://www.w3.org/Style/CSS/.

All the functions of the package `R2HTML` rely on CSS and a sample CSS file, `R2HTML.CSS`, which is used by `HTMLStart` is provided. In order to work properly, the CSS file has to be present in the same directory than the report and we simply have to add the following line to it `<link rel=stylesheet type=text/css href=R2HTML.css>`. The function `HTMLCSS()` performs this job. This is a good idea to systematically begin a report with this, as CSS files are very powerfull. Thus, and finally, our reporting function becomes:

```
> MyReport = function(x, file = "report.html",
+     CSSfile = "R2HTML") {
+     MyReportBegin(file)
+     HTMLCSS(file = file, CSSfile = CSSfile)
+     HTML(x, file = file)
+     MyReportEnd(file)
+ }
```

Summary

The `R2HTML` package provides functions to export all base R objects to HTML. We describe here a simple mechanism to use this functions to write HTML reports for statistical analysis performed with R. The mechanism is flexible and allow customizations in many ways, mainly by using a template approach (separating the body of the report from the wrapper - header and footer) and by using an external CSS file.

Availability

The `R2HTML` package is available from `CRAN` (e.g., `http://cran.us.r-project.org`).

References

[1] Friedrish Leisch. Sweave: Dynamic generation of statistical reports using litterate data analysis. *Compstat 2002 Proceddings in Computational Statistics*, pages 575–580, 2002.