R2HTML package

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

Eric Lecoutre

Statistics is not only theory and methodology, but also computing and communication. Applied statisticians are aware that they have to pay particular 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 wonderful Sweave system [1] in package **tools**: not only does it allow professional quality reports by using LaTeX, but it also stores used code within the document, which is very useful when returning to the analysis later on. This solution, however, is not always applicable, as the user may not be familiar with LaTeX or may need another format to communicate with a client, who, in many cases, will expect a report that he can edit or to which he can add some details. RTF format is ideal for this type of communication, as it can be opened on many systems and allows some formatting enhancements (bold, tables, ...). Nevertheless, it is not easy to produce and does not enable the user to embed graphs. Another universal format which can achieve our goal is HTML: it is light, readable on any platform, editable, and allows graphs. Moreover, it can easily be exported to other formats.

This documents describes the **R2HTML** package which provides some support for writing formatted HTML output. Although some knowledge of HTML is preferable to personalize outputs, the user may use this package successfully without such knowledge. We will create different web pages, which the reader can find at the following address:
`http://www.stat.ucl.ac.be/R2HTML/`.

Introduction to HTML and R2HTML package

According to 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 to sophisticated WYSIWYG authoring tools. HTML uses so-called tags to structure text into headings, paragraphs, lists, hypertext links and to apply a format. For example the `<b>` tag is used to start bold text, and the `</b>` tag to stop bold text. The tag with the slash (/) is known as the closing tag. Many opening tags need to be followed by a closing tag, but not all of them do.

Consequently, the only required knowledge in order to write basic HTML documents is the list of existing tags and their functionality. By way of illustration, 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 create our first webpage from R: simply using the `cat` function to write text into an external file. In the following example, consider how we call the `cat` function several times having set the `append` argument to `TRUE` in order 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)
```

The library **R2HTML** is simply a list of wrapper fonctions that call the `cat` function 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()` Inserts a `<br>` HTML tag, which requires a new line to be started.
- `HTMLhr()` Inserts an `<hr>` HTML tag, a horizontal rule to separate pieces of text.
- `HTMLInsertGraph()` Inserts an `<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 that behaves like the internal `cat` function. Common arguments are `append` and `file`, whose default value is set by the hidden variable `.HTML.file`. So, it is convenient to start by setting the value of this variable, so that 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 an HTML output on the fly

The first way to use the **R2HTML** library is to generate an automatic HTML output during an interactive session. This is especially convenient for teaching, as students can then keep a log of the commands they asked for and the corresponding outputs, with graphs incorporated. For a dynamic session to be successfull, two commands have to be processed:

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

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

- Three HTML files are written into the temporary directory of the session (or in the directory specified in an option). The main (`index.html`) file is linked to the other two, by incorporating them within HTML frames. It makes it possible to have the commands on the left of the screen and the corresponding outputs ont the right.
- A new environment, called `HTMLenv` is created, where some internal variables are stored. Those variables make it possible to store the path of the output files, and to know which action has been performed with the latest submitted command.
- A new `fix` function is assigned to the global environment, masking the internal one. When calling the "new" `fix`, a boolean is set to `TRUE` in the `HTMLenv` environment, so that we know that the latest action was to edit a function.
- `addTaskCallback` is called, adding a task to each submitted command. This task, handled by the function `ToHTML` (not visible to the user) is the core of the process, as it exports the last manipulated object. This function also tests whether a stored boolean indicates that a function has been edited and, if so, exports the edited function. When doing so, a new file is created, so that at the end of the process, one can keep tracks of all the versions of the function at the different stages of the work.
- Finally, as a side effect, the prompt is changed to `HTML>` so as a signal that outputs are currently beeing redirected.

From this moment on, every command is treated twice: first it is evaluated and then the result goes through the `ToHTML` function which writes it into the HTML output.

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

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

The following example only works in an interactive session with the RGUI. Simply use the following code:

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

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

HTML> sqrt(pi)
[1] 1.772454
HTML> x = rnorm(10)
HTML> x^2
 [1] 2.91248574 0.21033662
 [3] 0.16120327 1.56429808
 [5] 0.02863139 3.47605227
 [7] 1.36348399 0.30152315
 [9] 0.73402896 0.77886722
HTML> myfunction = function(x)
+  return(summary(x))
### try to fix the function
HTML> myfunction(x)
   Min. 1st Qu.  Median    Mean
-1.7070 -0.3017  0.6291  0.3878
3rd Qu.    Max.
 1.0960  1.8640
HTML> plot(x)
HTML> HTMLplot()
[1] TRUE
HTML> HTMLStop()
[1] "C:\\.../dynamic_main.html"
```

Creating personalized reports

Let us start with a simple analysis

For anyone who knows the basics of HTML, the **R2HTML** package offers all the necessary material

to developp fast routines in order to create one's own reports. But even users who have no knwoledge of HTML codes can still easily create such reports. What we propose here is a so-called template approach. Let us imagine that we have to perform a daily analysis whose output consists in some summary tables and graphs.

First, we gather all the material necessary in order to write the report in a list object. An easy way to do so is to create a user function `MyAnalysis` that returns 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)
+ }
```

We then provide a new `HTML` function, based on the structure of our output object and corresponding to its 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 variable")
+  dev.off()
+  HTMLInsertGraph("graph1.png",
+      Caption = "Graph 1 - Histogram",
+      file = file)
+  cat(paste("Report written: ",
+      file, sep = ""))
+ }
```

If we want to write 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:/.../page3.html
```

The advantage of this approach is that we store all the raw material of the analysis within an object, and that we dissociate it from the process that creates the report. If we keep all our objects, it is easy to modify the `HTML.MyAnalysisClass` function and generate all the reports again.

Template scheme to complete the report

What we wrote before is not a real HTML file, as it does not even contain standard headers such as `<html><head>` and `</head><body>`. As we see it, there are two differents ways to handle this, each with its pros and cons. For this personalization, it is indispensable to have some knowledge of acronymHTML.

First, we could have a pure R approach, by adding two functions to our report, 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 the `HTML` function directly, we would consider it as an internal function and, instead, call the `MyReport` function.

```
> out = MyAnalysis(data)
> MyReport(out, file = "page4.html")
Report written: C:/.../page4.html
```

The advantage is that we can even personalize the head and the footer of our report on the basis of some R variables such as the date, the name of the data or anything else.

If we do not need to go further than that and only need hard coded contents, we can build the report on the basis of two existing files, `header.html` and

footer.html, which can be modified to suit our needs. To work properly, the following piece of code supposes that those two files do 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 one step further with CSS

Cascading Style Sheets (CSS) compensates for some of the deficiencies of HTML language. CSS adds to each standard HTML element its own style, which is defined in an external file. Thus, when the house-style book of the report has to be changed, one need only modify the definition of the classes in a single place to change the look of all the reports - past or to come - that rely on the defined classes.

The use of cascading style sheets makes it possible to:

- give a homogeneous look to all generated reports
- change the look of a bunch of reports at one time
- produce lighter reports, as formatting instructions are kept separate
- download and view reports more rapidly

All the details about CSS specification can 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 given sample CSS file, `R2HTML.CSS`. This file is used by `HTMLStart`. In order to work properly, the CSS file has to be lcoated in the same directory as the report and one simply has to add the following line to it `<link rel=stylesheet type=text/css href=R2HTML.css>`. This job is performed byt the `HTMLCSS()` function. It is a good idea to systematically start a report with this function, as CSS files are very powerfull. So, in its last version, our reporting function yields:

```
> 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. Here, we describe here a simple mechanism to use these functions in order to write HTML reports for statistical analysis performed with R. The mechanism is flexible and allows 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 Proceedings in Computational Statistics*, pages 575–580, 2002.

Eric Lecoutre
Institut de statistique, UCL, Belgium
`lecoutre@stat.ucl.ac.be`