Using BloomR as a Report Generating Tool

What is all this stuff about?

You will be able to generate reports from BloomR in PDF or HTML.
For doing so you will write a file in special format in which:

- The narrative will be formatted with a very very simple markup language, named *Markdown* format (basically the one used when you write a Wikipedia article).
- In the middle of the narrative you can insert special code blocks, denoted as *code chunks*.

When you convert this file to PDF or HTML format the narrative will be pretty formatted, with bold, italic and all the bells and whistles; in place of the code chunks you will find their outputs, that is the tables, the plots or whatever the code generates.

If you want, you might *echo* the code, that is you can let display the R code too in the PDF/HTML report, so your reader will learn about the calculation you implemented to produce the report output.

Since these files include R code chunks inside a markdown narrative, they are named *Rmd* files (short for R-markdown).

In the next sections you will learn how easy is the markdown syntax and the commands needed to convert Rmd files to HTML or PDF files.

Setup BloomR for creating HTML reports

HTML reports are the simplest to create.

When you start BloomR load the following libraries:

```
library(knitr)
library(markdown)
```

As every library command they are per session. They stay loaded in memory until BloomR session is running. If you close and reopen BloomR, to use them you will need to load them again.

Now you are ready to create . . .

Your First R Markdown File

We start by creating an *Rmd* file containing only the narrative and no R code.

Change to some directory, for example `mybloomr` with the tilde:

```
setwd('~')
```

Create an empty file named `my-first-report.Rmd`. Note the `Rmd` extension.

```
file.create('my-first-report.Rmd')
dir() # Check the file was actually created!
```

Warning: For your convenience, inside `mybloomr/examples`, there is already a sample `my-first-report.Rmd`, pay attention not to overwrite it!

We now start editing the file with some Wikipedia-style content (markdown):

```
file.edit('my-first-report.Rmd')
```

The editor window pops up: copy and paste in it the following material:

```
This is a header for the section
================================

**Bold**, *Italic*, _Italic again_

These lines will be
printed as a single line.

To insert a line break leave two spaces after me.
Now I will be printed on a new line!

The following is the typewriter style. Use it if you want to comment/show code templates.

    # This is the main project function
    myfunction(arg1, arg2, arg3)

Leave four spaces before lines and (at least) one blank line above and below the block.
To use the typewriter style *inline*, enclose the text inside backticks,
therefore this `text` will be printed in typewriter style.
If you use dashes, `-`, instead of equals, `=`, you will get a subsection header.

Using Itemisations and Enumerations
-----------------------------------

*  Item 1
*  Item 2
*  ...

1.  Item 1
2.  Item 2
3.  ...
```

After the conversion, your text will be formatted like this:

This is a header for the section

Bold, *Italic*, *Italic again*

These lines will be printed as a single line.

To insert a line break leave two spaces after me.
Now I will be printed on a new line!

The following is the typewriter style. Use it if you want to comment/show code templates.

```
# This is the main project function
myfunction(arg1, arg2, arg3)
```

Leave four spaces before lines and (at least) one blank line above and below the block. To use the typewriter style *inline*, enclose the text inside backticks, therefore this `text` will be printed in typewriter style.
If you use dashes, `-`, instead of equals, `=`, you will get a subsection header.

Using Itemisations and Enumerations

- Item 1
- Item 2
- ...

1. Item 1
2. Item 2
3. ...

Carefully compare the initial markdown input with the formatted output and understand how it works and why.

This should be enough to create pretty documents. Later on you may want to learn more about formatting the narrative here: daringfireball.net.

NB: It is possible (and very common) to use an alternative syntax for section headers, that is:

```
# Section Header
## Subsection Header
```

Anyway I am not emphasizing it, since it can be confused with R syntax for comments.

When you are down with your editing, **save** the file. How?
Well, you might hit `Ctrl-S` on your keyboard, use the save button on BloomR toolbar, or use the menu `File->Save`. Anyway make sure that the editor is selected and not the console, otherwise you will save the command history!

Convert Rmd files to HTML files

Before adding *code chunks* to the narrative, let us see how to operate the conversion for the end-user document (perhaps your employer or your customer).

First of all some before-flight-checks: have you loaded the `knitr` and `markdown` libs? This is easily forgotten, therefore causing errors.
Also, can you see your Rmd-file in the current directory? If so, you avoid to provide the full path to it. To check this type in the console:

```
dir()
```

Now it is time to create your first HTML report, which is as simple as running the following two commands:

```
knit("my-first-report.Rmd") # ->will give: my-first-report.md
markdownToHTML("my-first-report.md", "my-first-report.html")
```

The first function *knits* the Rmd file, that is transforms the R markdown in a standard markdown (like the one used for Wikipedia articles). To be honest, for such a simple file it is pretty useless. The second function is self-explanatory: open with your browser `my-first-report.html` and see the report.

Let us step ahead to for a real Rmd file with code chunks.

R Code Chunks

We can now add R code chunks to the narrative. Code chunks have this general template:

~~~
```{r someLabel, option=value}
Put R code lines here
```
~~~

Note the three backticks and don't leave any space before them!
`someLabel` can be whatever, but it should be unique (you cannot reuse it in other chunks). Depending on what you want to obtain, there can be many options for `option = value` or even none.

Normally there is a first chunk setting global options for all document chunks, similar to the following:

~~~
```{r setup}
# set global chunk options: images will be 7x5 inches
opts_chunk$set(fig.width=7, fig.height=5)
```
~~~

These options will apply to all chunks unless explicitly overridden.

Your first code chunk can be as follows:

~~~
```{r myFirstChunk}
x = 1+1 # a simple calculator
set.seed(123)
rnorm(5)  # boring random numbers
```
~~~

When you convert your Rmd file to PDF/HTML, you will see, near the location of the chunk, *both* the code (pretty formatted) and the resulting output, i.e. five normal random numbers. As noted, you can change the label "myfirstchunk" to whatever you want, but the name must be unique.

What if you want only the results of the code (without the code itself)?
You should disable the *echo*.

~~~
```{r  myChunkNoecho, echo=FALSE}
set.seed(123)
rnorm(6)  # boring random numbers
```
~~~

Now, given the option `echo=FALSE`, after the conversion you will see only the random numbers, so without the code producing them.

Is everything clear? If so, you might rewrite `my-first-report.Rmd` with some R code.
In your editor replace the text with the following new one (as usual copy & paste to avoid typo):

~~~
My First Report
===============

Good morning, first of all I will set the default figure dimensions:

```{r setup}
# set global chunk options: images will be 7x5 inches
opts_chunk$set(fig.width=7, fig.height=5)
```

I will implement some calculations. See the code and results below:

```{r myFirstChunk}
x <- 1+1 # a simple calculator
set.seed(123)
rnorm(5)  # boring random numbers
```

I will now generate six random numbers.
Well, you already know the code behind, so I don't show it:

```{r myChunkNoecho, echo=FALSE}
set.seed(321)
rnorm(6)  # boring random numbers
```

And now let me plot my ideas:

```{r myNicePlot, fig.cap="In this plot we compare Miles per Gallon with HP."}
par(mar = c(4, 4, 2, .1))
with(mtcars, {
  plot(mpg~hp, pch=20, col='darkgray', main="My First Plot")
  lines(lowess(hp, mpg))
})
```
~~~

After the conversion (that we are going to do), your text will be formatted like this:

My First Report

Good morning, first of all I will set the default figure dimensions:

```
# set global chunk options: images will be 7x5 inches
opts_chunk$set(fig.width=7, fig.height=5)
```

I will implement some calculations. See the code and results below:

```
x <- 1+1 # a simple calculator
set.seed(123)
rnorm(5)  # boring random numbers
```

```
## [1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
```

I will now generate six random numbers. Well, you already know the code behind, so I don't show it:

```
## [1]  1.7049032 -0.7120386 -0.2779849 -0.1196490 -0.1239606  0.2681838
```

And now let me plot my ideas:

```
par(mar = c(4, 4, 2, .1))
with(mtcars, {
  plot(mpg~hp, pch=20, col='darkgray', main="My First Plot")
  lines(lowess(hp, mpg))
})
```

As you can see code chunks are now followed by their output.

When a code chunk contains plotting commands, here the command `plot(mpg~hp, pch=20, col='darkgray')`, then the report will also show the generated figure. There are some differences concerning figure positioning.
In HTML-format output the figure will be placed *immediately after* the code chunk generating it (so here soon after `myNicePlot` chunk).
That will be different for PDF output. Because PDF's want to resemble the physical paper, figures are automatically placed in the document in such a way to optimise the text flow. In fact, sometimes there is not enough space on the current page to insert the picture, which is therefore placed on another page (perhaps the next one). So, if in this very moment you are reading me from a PDF output, don't be surprised to find the plot placed in what is only apparently a weird place.

A final notice about figure generating chunks. The code for `myNicePlot` uses the not so mysterious `fig.cap` option. Yes, you guessed it: it is the text for the plot caption. It is normally not used in HTML format, so I will talk about this later.

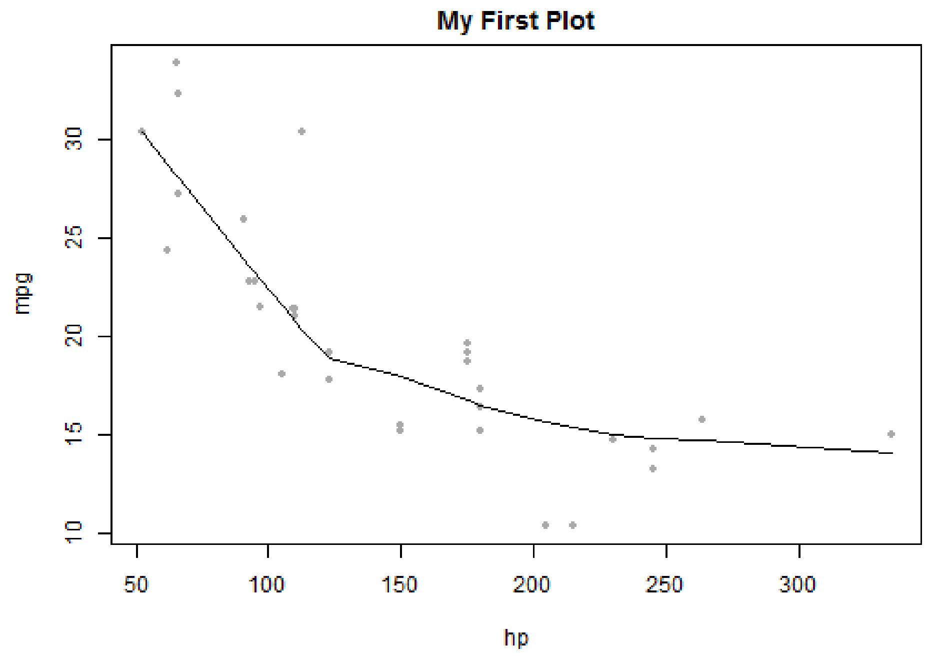

Figure 1: In this plot we compare Miles per Gallon with HP.

Maths

You can use LaTeX maths, therefore:

```
$f(\alpha, \beta) \propto x^{\alpha-1}(1-x)^{\beta-1}$
```

will be rendered as: $f(\alpha, \beta) \propto x^{\alpha-1}(1-x)^{\beta-1}$.

Convert the rich Rmd document to an HTML report

As usual, when you are done with editing, save `my-first-report.Rmd` and type in the console:

```r
knit("my-first-report.Rmd") # ->will give: my-first-report.md
markdownToHTML("my-first-report.md", "my-first-report.html")
```

Now let us check the file produced:

```r
dir() # a strage "figure" directory will appear
```

This time, together with `my-first-report.html`, you see also a mysterious `figure` directory listed.

As you might already know, the HTML files are not technically designed to store figures (but only markup text to be interpreted by browsers). So HTML files just tell the browser where to go and find the figures to be displayed.
In our case the report's figures are placed in the in the `figure` directory. Do explore yourself this directory: you will find the list of plots used by the report as PNG pictures (a format better than the best known JPEG).

There is an important thing to remember now: whenever you move/share the HTML report, don't forget to *bring the figure directory alongside the file*!

Setup BloomR for creating PDF reports

Creating PDF reports implies downloading the BloomR LaTeX addons. Since they take huge space, they are not added by default to BloomR.

You will need approx. 700 MB of free space on the storage device where BloomR is installed, and you will also need an Internet connection to properly download the addons.
Note that you *do not* need to be logged to the Bloomberg Service (or anyway working on a Bloomberg terminal) to install the addons, you will need Bloomberg only later, when generating the reports.

Assuming that you have the space and an Internet connection, the setup consist of this line of code:

```
 br.getLatexAddons()
```

The process can be long depending on the speed of your Internet connection and of the device where BloomR is stored. During the setup you will see a number of windows popping up. They will automatically close. *Don't touch them* or you will crash the installation.
Of course this long process is to be done only once.

Convert the rich Rmd document to a PDF report

Since we have already generated the HTML report and so we have *knitted* the Rmd file, we already have generated the markdown file `my-first-report.md`. We need to transform this file in a PDF. To tranform an `.md` file to a PDF we type in the console:

```
dir() # Check that your '.md' file is here
br.md2pdf("my-first-report.md", "my-first-report.pdf")
```

As we have observed above, in LaTeX (PDF) the plot is automatically placed in such a way to optimise the text flow. Figures and tables are therefore named by those who speak properly *floats*.

One thing to note is that, *the order by which floats are displayed in the report reflects that of the code chunks generating them.* Besides, below them, there is an automatic generated sequential identifier, e.g.: "Figure 1". You will use this number to address the floats inside the report body text.

As hinted above, you can do even more. Set the `fig.cap` property in your plot generating chunks as follows:

~~~
```{r myNicePlot, fig.cap="In this plot we compare Miles per Gallon with HP."}
    ...
```
~~~

Then in the PDF output (but not in HTML output), after the plot number, you get also the given caption, which adds more insights and makes the report more eye catching.

I am not a literate girl

If one emphasises the code side of this document vs. the English description, she speaks of *literate programming*; but there are situations in which you want to *purl* the document, that is stripping the narrative and leave only the code, in other words we want to convert the Rmd file containing words+code in a standard R script where only the code survives. That's as simple as that:

```r
purl("my-first-report.Rmd")
```

Conclusion

That's it. Now you can fully appreciate the difference between a spreadsheet, which is undisclosed and unrepeatable, and *reproducible research.*
For more info go to knitr homepage.