fille**R**

Filling a (flat) PDF form with data from a CSV file in R

Introduction

Don't get me wrong I am a fan of knitr (and a former user of Sweave), but there are situations when they might just not fit the purpose. Indeed, as LaTeX based tools, they lie on the principle that content is more important than presentation, or better presentation technicalities. The latter should not occupy the researchers' time and be delegated to a mostly predefined model.

Anyway, chances there are that for some documents, the content is very silly, while you need a very detailed control over the final printed output, that is over the position of your printed material. One such a case is filling a form: you have to print words exactly in the blank spaces designed for them.

One simple strategy for the you R user is printing the filling text on a PDF file, as you would for a plot, thereafter you overlay it on the PDF document to be filled. Note the here by PDF form to be filled I mean a 'flat' form, not the special file type hosting form fields.

There are many free tools for overlaying PDFs: a cross platform open source example is Apache PDFBox in Java.

In the following I will show how to read a data from a spreadsheet in CSV format and using them to fill a (many) PDF form(s).

Writing on the border of your page

Saving a figure as a PDF is nothing new for you and most likely you know that you can write a blank plot with `plot.new()` and add text to it with `text()`. The problem is that R adds white spaces every here and there for aesthetic reasons; but, if you need to fill a form, you need to write your text exactly n inch from the borders and not n plus some offset. Obviously you have two alternatives: you adjust your text printing commands to take into account R blank offsets; you set all offsets to zero. The former is impractical also because the offset is in percent, so it is not a matter of simple subtracting a given delta. The latter is a bit tricky (at least, so appeared to me), but you do it once and for all.

With the following code you will create `foo.pdf` in R current directory with `hello` written exactly on the left border of the page.

Note that there are three places (in `par()` and `text()`) where we need to nullify the white space.

```
WIDTH=8.3; HEIGHT=11.7        #Paper size A4, measure  in inches

pdf('foo.pdf', WIDTH, HEIGHT) #Write next plot to 'foo.pdf'
par(mar=c(0, 0, 0, 0))        #Set numbers of lateral blank lines to zero
par(xaxs='i', yaxs='i')       #Does not extend axes by 4 percent for pretty labels

plot.new()                    #Create a blank plot, where we will want to write our text
plot.window(xlim=c(0,WIDTH), ylim=c(0,HEIGHT)) #Fit plot to paper size
text(0, .5, "hello", pos=4, offset=0)   #Write to the right of coords without default .5 off

dev.off()                     #Close device, that is saving for a PDF device
```

Change `WIDTH, HEIGHT` above to your actual paper size.

Since `text` will be used quite often, and we might want to change font and magnification, it is better to define a specialised print function:

```
###Print left aligned
MAGNI=1    #Magnification factor
FONT=1     #Font: 1 is Helvetica regular, 2 Helv. bold, ... 6 Times
ltext=function(x,y, s) text(x,y, s, pos=4, offset=0, cex=MAGNI, font=FONT)
```

Read the R manual for more on plot magnification factors and fonts and set them as you please.

Reading data from a CSV file

So we can now easily place text wherever in the page, let's take the data from a CSV file. The structure will be as follows:

```
x,y,text
3,5,John
3,4,Smith
.....
```

If you think `x,y` are the coordinates and followed by the to print, you guessed it right.

The new code, now reading the overlay material from the CSV data, is:

```
OVER='overlay.pdf'; WIDTH=8.3; HEIGHT=11.7 #Overlay PDF path and size (inches)
DATA='form.csv'                            #CSV data source

pdf(OVER, WIDTH, HEIGHT) #Write next plot to the overlay PDF
```

```
par(mar=c(0, 0, 0, 0))   #Set numbers of lateral blank lines to zero
par(xaxs='i', yaxs='i')  #Does not extend axes by 4 percent for pretty labels

plot.new()                                      #Create the blank plot to write to
plot.window(xlim=c(0,WIDTH), ylim=c(0,HEIGHT)) #Fit plot to paper size
d=read.csv(DATA, as.is=TRUE)                    #Read fill data
ltext(d$x, d$y, d$text)                         #... and print

dev.off()                                       #Save overlay PDF
```

Generating a multipage PDF

You may rightly think that this game makes sense if we have multiple forms to fill.

The CSV could now look something like:

```
x,y,text
3,5,John
3,4,Smith
.....
-1
3,5,Bob
3,4,Sullivan
```

Yes, `-1` tells to skip to a new page. A spreadsheet might set this row as `-1„`, but in R it is the same. You can use any number (but not a letter), as the code just checks that the second field is empty.

```
OVER='overlay.pdf'; WIDTH=8.3; HEIGHT=11.7 #Overlay PDF path and size (inches)
DATA='form.csv'                            #CSV data source

pdf(OVER, WIDTH, HEIGHT) #Write next plot to the overlay PDF
par(mar=c(0, 0, 0, 0))   #Set numbers of lateral blank lines to zero
par(xaxs='i', yaxs='i')  #Does not extend axes by 4 percent for pretty labels

plot.new()                                      #Create the blank plot to write to
plot.window(xlim=c(0,WIDTH), ylim=c(0,HEIGHT)) #Fit plot to paper size

d=read.csv(DATA, as.is=TRUE)                    #Read and print fill data one row per time
for (i in 1:nrow(d)) {
  x=d[i,1]; y=d[i,2]; tx=d[i,3]
  if(is.na(y)){                                 #On -1 start a new plot/page
    plot.new()
```

```
    plot.window(xlim=c(0,WIDTH), ylim=c(0,HEIGHT))
  }
  ltext(x, y, tx)
}

dev.off()                                        #Save overlay PDF
```

Adding multiline entries with automatic left justification

Another interesting thing could be to fill a multilne box. The idea is that in the CSV we set an optional `length` field, where we say how many characters the multiline text should be large. So, in the CSV the row for a multiline box would be like:

```
x,y,text,length
3,4,"Very long text to be split every n characters",10
```

Note the double quotes to mask commas, which often recur in long texts.

To left justify a string with a given text width, we define:

```
###Left multiline justification at 'width'
justify=function(string, width){

  str.len=nchar(string)
  sp=gregexpr(' ', string)[[1]]                              #Get text spaces
  l=seq(from=width, by=width, length=floor(str.len/width)) #Get limits for every row
  bsp=sapply(l, function(x) max(sp[sp<=x]))                  #Breaking spaces
  rows=substring (string, c(1, bsp), c(bsp, str.len))       #Extract lines
  rows=sub("^ +", "", rows)                                  #Remove leading spaces
  paste(rows, '\n', collapse='')                             #Merge rows, with newlines
}
```

Integrating the previous code will bring too:

```
OVER='overlay.pdf'; WIDTH=8.3; HEIGHT=11.7 #Overlay PDF path and size (inches)
DATA='form.csv'                            #CSV data source

pdf(OVER, WIDTH, HEIGHT) #Write next plot to the overlay PDF
par(mar=c(0, 0, 0, 0))   #Set numbers of lateral blank lines to zero
par(xaxs='i', yaxs='i')  #Does not extend axes by 4 percent for pretty labels
```

```
plot.new()                                      #Create the blank plot to write to
plot.window(xlim=c(0,WIDTH), ylim=c(0,HEIGHT)) #Fit plot to paper size

d=read.csv(DATA, as.is=TRUE)                    #Read and print fill data one row per time
for (i in 1:nrow(d)) {
  x=d[i,1]; y=d[i,2]; tx=d[i,3]; text.width=d[i,4]

  if(is.na(y)){                                     #On -1 start a new plot/page
    plot.new()
    plot.window(xlim=c(0,WIDTH), ylim=c(0,HEIGHT))
  }

  if(!is.na(text.width)) tx=justify(tx, text.width) #Justify left
  ltext(x, y, tx)
}

dev.off()                                       #Save overlay PDF
```

Overlay the text over the form

To finalize our project we need to print on the form, that is to overlay the generated PDF over the original form.

To make a real world example I will use a book I have to fill for tracking my lectures.

The template for the CSV data to print is as follows:

```
x,y,text,length
*,*,Lecture Day/Month
*,*,Lecture Start time
*,*,Lecture End time
*,*,Lecture Description,15
... 4 like this per page
-1
... start again
```

For a usable 2 pages CSV see here.

Here is the form before and after filling (unfortunately not in English).

First of all, to keep track of pages used, we have to add a page counter every time we call `plot.new`. So it could be a good idea to define a new-page function:

```
PAGE.COUNT=0
```

```
....
###Create a new overlay page and update the page counter
new.page=function(page.width, page.height){
    plot.new()               #Create a blank plot, as we just want to write our text
    plot.window(xlim=c(0,page.width), ylim=c(0,page.height)) #Fit plot to paper size
    PAGE.COUNT<<-PAGE.COUNT+1
}
```

Here to overlay our text over we will use the open source and cross platform Apache PDFbox and particularly the java based PDFBox Command Line Tools. Before starting make sure that both pdfbox-app and the single page form to be filled (`form.pdf`) are available from the R working path. As an alternative, you may want to adjust the path occurring in the following code in accordance to yours. For portability reason, we will run the PDFBox shell commands via R.

We start initialising PDFBox and related variables:

```
PDFBOX="java -jar pdfbox-app-1.8.2.jar"    #Modify this lines to match your system, version
TEMP='temp.pdf'; FORM='form.pdf'; FILLED='form-filled.pdf'                  #... and your fo
```

We now create a temporary PDF replicating the single-page form for the number of overlay pages using the PDFMerger command, the synopsis is:

```
java -jar pdfbox-app-x.y.z.jar PDFMerger <Source PDF files (2 ..n)> <Target PDF file>
```

So we run:

```
cmd=paste(rep(FORM, PAGE.COUNT), collapse=' ')
cmd=paste(PDFBOX,  "PDFMerger",  cmd, TEMP)
try(system(cmd, intern = TRUE))
```

We can now overlay the overlay PDF on the temporary PDF by means the homonymous command Overlay:

```
java -jar pdfbox-app-x.y.z.jar Overlay <overlay.pdf> <document.pdf> <result.pdf>
```

That is:

```
cmd=paste(PDFBOX, "Overlay", OVER, TEMP, FILLED)
try(system(cmd, intern = TRUE))
```

Some optional bells and whistles with PDFReader:

```
cmd=paste(PDFBOX, "PDFReader", FILLED)
try(system(cmd, intern = TRUE))
```

Read the full code on github FilleR.

Final considerations

To position your text properly on the PDF you may take advantage of the distance tools present in many PDF applications, including some free ones.

For example, under Windows, this image shows a distance tool in action using the free PDF-XChange Viewer.