Using the figures2 package to assist with large scale figure production

Greg Cicconetti

September 23, 2014

Contents

1 Prerequisistes

1.1 Recommended directory structure

The default.settings function places text objects with default paths to some of these locations.

- Create a parent directory to associated with your project.
- Create subfolders associated with a reporting effort
- With the reporting effort create the following subfolders:
  - code: a directory to hold all r driver files
  - dddata: a directory to hold all .csv files
  - output: a directory to hold outgoing figure files
  - log: a directory to hold log files etc.

1.2 Driver files

The purpose of these files is single fold: Produce a figure. Recall a figure is considered to be a collection of graphics/tables assembled on a page and annotated with headers/footers/titles. The examples that follow divide this process into the following tasks:

- Begin tracking newly created objects, sync with outputplan and start log file

- Address mathematical symbols needed for headers/footers/titles
- Load or create data set(s) for graphic/table components
- Pre-Process data set(s)
- Create graphic object(s)
- Post process graphic object(s)
- A call build.page to assemble object(s) on the page
- A call annotate.page to added headers/footers/titles
- Removal of objects created and close of log file

1.3 The outputplan and the refresh.outputplan function

In practice, the outputplan is read from a .csv file that may be centrally located (to facilitate editing by team members). One can populate the outputplan in excel to facilitate syncing with related files (e.g., the qcplan).

The outputplan holds metadata for each figure. One row is allocated to each figure. The outputplan can have superfluous columns. As a rule, entries in the outputfile are intended to be unique. This allows the driver files, the files stored in the directory and whose names are in stored also stored in the rcode column of the outputplan, to uniquely identify the row associated with the figure being produced allowing the annotate.page function to populate the figure's headers and footers. Superfluous columns are ok (and may prove helpful if you are using the outputplan to help populate driver files). See annotate.page code for the minimum subset of columns required in addition to outputfile and rocde columns.

The figures2 package has a dummy outputplan stored as a data set.

```
require(figures2); require(stringr)
data(outputplan)
# Run the next line to see first 6 rows
# head(outputplan)
```

The figures2::refresh.outputplan function is used to process a data.frame sharing a subset of these column names. In addition to handling some pre-processing tasks when the outputplan is read from a .csv file, this funtion:

- counts the number of lines to allocate for titles: nTitleLines
- counts the number of lines to allocate for footnotes: nFootLines

These can be leveraged to dynamically adapt the graph region to accomadate title and footer requirements.

```
# loadplan=F presumes outputplan exists in the Global environment
refresh.outputplan(loadplan=F)
# Note the additional columns created these will get passed to annotate.page
names(outputplan)
```

```
 [1] "Study"        "Section"      "Number"       "FigureNumber"
 [5] "TableID"      "HarpTitle"    "PlotStyle"    "ulh1"
 [9] "ulh2"         "ulh3"         "urh1"         "urh2"
[13] "urh3"         "fnote1"       "fnote2"       "fnote3"
[17] "fnote4"       "outputfile"   "rcode"        "csv"
[21] "FigureTitle"  "Author"       "Response"     "RunThis"
[25] "FigureTitle4" "FigureTitle3" "FigureTitle2" "FigureTitle1"
[29] "nTitleLines"  "nFootLines"
```

This function can also read an outputplan stored in an external .csv file, preprocess it, and store the resultant data.frame, called outputplan, in the Global Environment. Regarding the uniqueness of names found in the outputfile column mentioned above, one can subset the outputplan at this point in a way that assures this at this point. E.g., if the external outputplan holds metadata for multiple projects and these projects are intended to produce files with common names, one could subset the outputplan to isolate a single project, thus ensuring that the outputplan used by R meets the constraint.

1.4 The run.specific function

This function takes a driver name and optionally outputs it to the screen or any number of file formats. This function presumes the driver is located in the code directory mentioned above and that files are sent to the output directory.

This and all.in.one are wrapper functions for grDevices::pdf and related functions.

1.5 all.in.one

This function loops through an outputplan producing figures2 whenever a row entry for a stated column takes the value "Y". The driver file name used for sourcing is obtained from the rcode column of the outputplan.

2 Large scale production workflow

The following outlines the gestalt approach successfully employed in clinical trial reporting.

2.1 Set up local directory structure

Warning the following code will create directories on your computer for the purpose of demonstration!! Ensure your working directory is pointing to a folder that you can use for experimenting.

```
# setwd("path_to_your_parent_folder")

# Creates a folder called workflow_demo
dir.create(file.path(getwd(), "workflow_demo"), showWarnings = FALSE)

# Re-sets the working directory to workflow_demo
setwd(file.path(getwd(), "workflow_demo"))

# Creates code, dddata, output, log folders
dir.create(file.path(getwd(), "code"), showWarnings = FALSE)
dir.create(file.path(getwd(), "dddata"), showWarnings = FALSE)
dir.create(file.path(getwd(), "output"), showWarnings = FALSE)
dir.create(file.path(getwd(), "log"), showWarnings = FALSE)
```

2.2 Populating the data folder with mock .csv files

The following code brings data sets stored in the figures2 packages into Global Environment.

```
data(outputplan)
data(benrisk2.data)
data(summary.lineplot.data)
data(boxplot.driver)
data(demog.data)
data(forest.data)
data(cdf.data)
data(km.data)
data(raw.lineplot.data)
data(lineplot.data)
```

The following saves these datasets as .csv files in the dddata directory. In practice, files are transfered from elsewhere into your dddata folder.

```
write.csv(file=file.path(getwd(), "dddata", "outputplan.csv"), outputplan)
write.csv(file=file.path(getwd(), "dddata", "benrisk2.data.csv"), benrisk2.data)
write.csv(file=file.path(getwd(), "dddata", "summary.lineplot.data.csv"),
          summary.lineplot.data)
write.csv(file=file.path(getwd(), "dddata", "g_bslchar.csv"), demog.data)
write.csv(file=file.path(getwd(), "dddata", "forest.data.csv"), forest.data)
write.csv(file=file.path(getwd(), "dddata", "cdf.data.csv"), cdf.data)
write.csv(file=file.path(getwd(), "dddata", "km.data.csv"), km.data)
write.csv(file=file.path(getwd(), "dddata", "raw.lineplot.data.csv"), raw.lineplot.data)
write.csv(file=file.path(getwd(), "dddata", "lineplot_example.csv"), raw.lineplot.data)
```

2.3 Populating the code directory with driver files

2.3.1 Mimicking the creation of authoring individual driver files

Let's populate with some other prefabricated driver files stored in the figures2 package as datasets. These are first loading them into R's Global Environment.

```
data(continuous_by_visit_and_treatment)
data(category_by_visit)
data(scatter_smooth)
data(scatter_smooth_facet)
data(scatterplot_with_smoother)
data(jitter_weight_faceted)
data(jitter_weight)
data(cdf_weight)
data(priordens)
```

These are now written as text files to the code folder using a .r extenstion. This process would mimic the creation of nine separate driver files.

```
writeLines(driver1, con=paste0(getwd(),"/code/","continuous_by_visit_and_treatment.r"))
writeLines(driver2, con=paste0(getwd(),"/code/","category_by_visit.r"))
writeLines(driver3, con=paste0(getwd(),"/code/","scatter_smooth.r"))
writeLines(driver4, con=paste0(getwd(),"/code/","scatter_smooth_facet.r"))
writeLines(driver5, con=paste0(getwd(),"/code/","scatterplot_with_smoother.r"))
writeLines(driver6, con=paste0(getwd(),"/code/","jitter_weight_faceted.r"))
```

```
writeLines(driver7, con=paste0(getwd(),"/code/","jitter_weight.r"))
writeLines(driver8, con=paste0(getwd(),"/code/","cdf_weight.r"))
writeLines(driver9, con=paste0(getwd(),"/code/","priordens.r"))
```

2.3.2 Getting R to build drivers with pattern replacement

This approach might be considered if you need to create 15+ driver files that differ only in a few locations. The following code brings a skeletonized driver for building box plots into the code directory.

```
# Write boxplot.driver file to code directory
writeLines(boxplot.driver, con=file.path(getwd(), "code", "boxplot.driver.txt"))
```

By making use of columns included in the outputplan to this task and the skeletonized driver, we can have R populate the driver files and store these the code directory. The idea here is to search through the skeltonized driver for a predefined set of text patterns, replace those patterns with values taken from columns of the outputplan and save the result with a predefined filename.

WARNING: in practice it would be a good idea to have these populated in a staging directory to avoid inadvertently overwriting files!

```
outputplan.bp <- subset(outputplan, PlotStyle == "Boxplot" & Response != "")

 for(i in 1:nrow(outputplan.bp )){
   tx  <- boxplot.driver
   tx  <- gsub(pattern = "@ProgramName", replace = paste(outputplan.bp$Study[i],
                    outputplan.bp$ULHead1[i]), x = tx)
   tx  <- gsub(pattern = "@DataFileName", replace = outputplan.bp$csv[i], x = tx)
   tx  <- gsub(pattern = "@OutputName", replace = outputplan.bp$outputfile[i], x = tx)
   tx  <- gsub(pattern = "@RESPONSE", replace = outputplan.bp$Response[i], x = tx)
   tx  <- gsub(pattern = "@ProtocolID", replace = outputplan.bp$ulh1[i], x = tx)
   tx  <- gsub(pattern = "@PlotStyle", replace = outputplan.bp$PlotStyle[i], x = tx)
   tx  <- gsub(pattern = "@FigureTitle", replace = outputplan.bp$HarpTitle[i], x = tx)
   tx  <- gsub(pattern = "@TableRef", replace = outputplan.bp$TableID[i], x = tx)
   tx  <- gsub(pattern = "@Author", replace = outputplan.bp$Author[i], x = tx)
   tx  <- gsub(pattern = "@Date", replace = Sys.Date(), x = tx)
   writeLines(tx, con=paste0(getwd(),"/code/",outputplan.bp$rcode[i]))
 }
```

2.4 Simulating a session

At this point, all of the components are in place to create figurs. First, items are cleared from the Global Environment to mimic a fresh session start.

```
remove(list=ls())
```

```
require(figures2)
# ./workflowdemo should still be the working directory.
#setwd(file.path(getwd(), "workflowdemo"))
default.settings()
```

2.4.1 Importing the outputplan

In practice, the outputplan would be read in from .csv file.

```
data(outputplan)
head(outputplan)
refresh.outputplan(loadplan=F)
```

2.4.2 Running individual .pdf files

Apply run.specific:

```
run.specific(as.character(outputplan$rcode[1]), toPDF=T)
run.specific(as.character(outputplan$rcode[2]), toPDF=T)
run.specific(as.character(outputplan$rcode[3]), toPDF=T)
run.specific(as.character(outputplan$rcode[4]), toPDF=T)
run.specific(as.character(outputplan$rcode[5]), toPDF=T)
run.specific(as.character(outputplan$rcode[6]), toPDF=T)
run.specific(as.character(outputplan$rcode[7]), toPDF=T)
run.specific(as.character(outputplan$rcode[8]), toPDF=T)
run.specific(as.character(outputplan$rcode[9]), toPDF=T)
run.specific(as.character(outputplan$rcode[10]), toPDF=T)
run.specific(as.character(outputplan$rcode[11]), toPDF=T)
run.specific(as.character(outputplan$rcode[12]), toPDF=T)
run.specific(as.character(outputplan$rcode[13]), toPDF=T)
run.specific(as.character(outputplan$rcode[14]), toPDF=T)
# In practice it may be easier to work with the filename, e.g.,
run.specific("continuous_by_visit_and_treatment.r", toPDF=T)
```

At this point, the output directory should have 15 pdf files.

2.4.3 Populating a single .pdf file with multiple figures2

Finally, here is a call to produce a single pdf with all figures2.

```
all.in.one(UseSubset = "RunThis", filename= "allinone.PDF" , reportNR=FALSE)
```