Generating PowerPoint Reports from a Shiny App

C. Ashton Drew, PhD
http://www.kdv-decisions.com
April 2016

Objective

Create a Shiny app for a survey admin to upload data, monitor response rate, and generate a standardized PowerPoint report for selected individuals.

Summary

For a recent project (YG360 App), I used the package ReporteRs by David Gohel for the first time and I thought I'd share a few tips and example code snippets. I found this package very easy to use, primarily building from the powerpoint example on the author's website. My biggest challenges were learning to customize the PowerPoint templates and then fine-tuning the layouts align the various parts. However, once I learned to manipulate the templates, I found building presentations with ReporteRs much easier, and far more customizable, than fiddling around with Slidy or other slide packages in R. The one possible downside: ReporteRs requires Java JRE (see special ReportRs install instructions).

Design Criteria for YG360 App

- Ability to compare list of invited participants (Excel) with the respondents in the SenseMaker[1] data output (csv)
- Ability to select individuals from a drop-down list and generate a PowerPoint report
- Reports generated with unique names (Project-SubjectName-Date.pptx)
- Inclusion of some custom graphics, modified from examples in previous reports[2]
- Custom designed template and color palettes
- Expectation for the app audience to be one non-technical clerical staff (the client wants to administer the project and generate reports as-needed without specialized expertise)

The YG360 App Online

The YG360 app is hosted on Shinyappsio but requires two external input files. I've put two faux data files on Github so you can try out the real app. (N.B.: It takes a few moments for the app to generate all the graphics and compile the PowerPoint.)

- List of invited participants: YG360_Invitations_DemoData.xlsx
- SenseMaker responses output: YG360_Responses_DemoData.csv

1. The YG360 App presents data collected via a SenseMaker instrument.
2. The YG SenseMaker 360 instrument was adapted by Yukon Government and Onfoot Consulting from the work of Anne MacMurray and Cognitive Edge and their 360NetImpact instrument.

A Simple Working Example of ReporteRs within Shiny

I designed this example to help others get started building a PowerPoint from within R Shiny. The documentation that comes with ReporteRs provides detailed examples for inserting other types of context (e.g., flex tables) or generating Word documents. The code and template for this example are on my Github:

- app.R
- Project_Template.pptx

Creating a custom Powerpoint template

Windows 10, MS Office 365 template for use by ReporteRs

- Open a blank Powerpoint file.
- Switch to Slide Master view by selecting *View > Slide Master*
- Notice that there are two types of master slide:
    - Office Theme Slide Master – any edits to this slide master will propagate through all the slides
    - Layout series of slides – edits to these slides just modify individual layouts
- Interact with these Master Theme and Layout slides in the same way you'd build and modify a PowerPoint presentation. Add a banner image and logo to the Theme to have this appear throughout the slide – or just add images and logos to a single slide layout. The same instruction applies to changing font styles, rearranging the slide elements, etc.
- Hover over a slide to view its name, right-click to access a dialog to rename, add, delete, or duplicate a slide.
- Choose *Slide Master > Insert Placeholder* to insert new elements for tables, text, media, images, etc.

All of the above was fairly straight-forward once I found the Master Slide commands. What was less intuitive, yet apparently necessary to use the template with ReporteRs:

- When you have a template you like and are ready to save it, FIRST SWITCH BACK TO NORMAL VIEW (click *Close Master View*) and delete any/all slides that are present in the deck.
- DO NOT SAVE AS A TEMPLATE (*.potx) – save it as a regular *.pptx file in the main folder of your Shiny app.

Build a *.pptx Report within Shiny

R 3.3.2 (2016-10-31), R Studio 1.0.136, ReporteRs 0.8.8, and shiny 1.0.0

In the code at the end of this document (pg 4-5), I've focused on the downloadHandler component. The YG360 app contains fileInput, uiOutput and dataTableOutput code to upload files, select subject for reporting, and view summary tables. This code instead uses two tibbles of generated data and substitutes in a simple selectInput and tableOutput. Here are some tips with added explanation or options for certain points in the demonstration code.

Line 33: You create the filename here, so it seems frustratingly counter-intuitive when you run the Shiny app locally and it pops open the file dialog to save a file named "report" – not the nice unique name with a pptx extension. I spent a long time trying to fix this before realizing that when I deployed the app on Shinyappsio, it would use the reactively generated name. I'm sure there is a way to have it also work as expected locally – but it worked where I needed it to work, so I didn't try further fixes.

Line 37: If you'd rather use the default template, just use `doc <- pptx()`

Line 39: Update the doc object by adding a slide with the addSlide() function. You must provide a layout name from either the default or your custom template. If you cannot remember the layout names, in the console window you could type:

```
doc <- pptx(template="C:/Demos/Project_Template.pptx") # or doc <- pptx()
slide.layouts( doc )
[1] "Project_TitleSlide"  "Project_Scatter&Pie"
```

line 41: Update the doc by adding content to the various elements of that layout. Some elements will have names (e.g. Title, Subtitle, PageNumber) and others are simple generic "Body" elements where you can add text, plots, or tables. If you are unsure what the elements are named and/or what order they are numbered, you can return to the console and type:

```
slide.layouts(doc, 'Project_Scatter&Pie')
```

This will open a diagram of the slide layout in R Studio's plot window. Notice that you do NOT specify which Body element you are providing – the first and second plots added are assigned to the Body 1 and Body 2 elements, respectively.

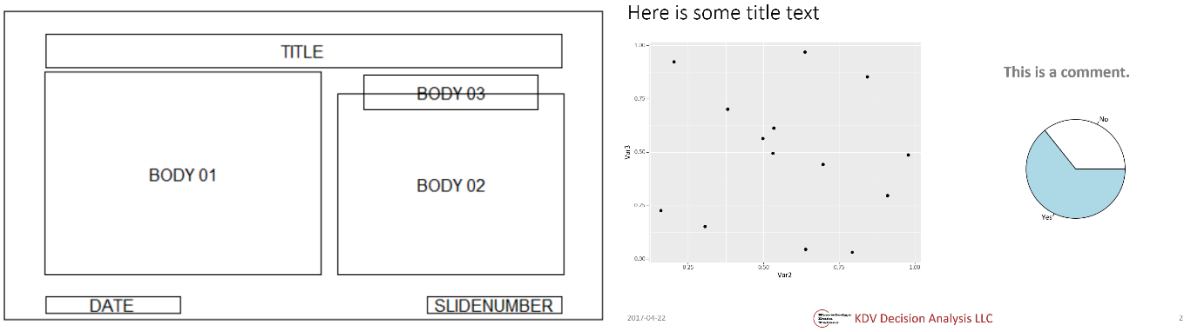

Figure 1. The layout diagram for the 'Project_Scatter&Pie' layout (LEFT) and slide generated with that layout (RIGHT).

Lines 47-49: Notice the difference between adding a ggplot figure and a base figure! The ggplot figure is created as an object that is then printed, while the base plot is created directly in the addPlot() function.

```
library(ggplot2)
library(magrittr)
library(dplyr)
library(ReporteRs)

invites <- tibble(Leaders = c(rep("LeaderX", 8), rep("LeaderY", 6)),
                  Subjects = paste(rep("Subject", 14), 1:14))
responses <- tibble(Leaders = c(rep("LeaderX", 8), rep("LeaderY", 6)),
                    Subjects = paste(rep("Subject", 14), 1:14),
                    Var1 = rep(c("Yes","No"), 7), #rbinom(14, 1, 0.75),
                    Var2 = runif(14, 0, 1),
                    Var3 = runif(14, 0, 1))
dat <- merge(invites, responses, by = c("Leaders", "Subjects"))

piedat <- dat %>%
  group_by(Var1) %>%
  summarize(CntVar1 = n()) %>%
  ungroup()

ui <- fluidPage(
  #  tableOutput("StatusSummary"),
  selectInput("LeaderID", "Select a leader to generate a report:", unique(dat$Leaders)),
  downloadButton("report", "Save Report")
)

server <- function(input, output) {
  output$StatusSummary <- renderTable({
    dat })

  output$report <- downloadHandler(
    # the filename to use
    filename = function() {
      paste0("Demo_", input$LeaderID , "_",Sys.Date(),".pptx") },
    # the document to produce
    content = function(file){
      # use custom template
      doc <- pptx(template = 'Project_Template.pptx')
      # Add a slide by first selecting a layout from your Master Slide layout styles
      doc <- addSlide( doc, slide.layout = 'Project_TitleSlide' )
      # Then fill in the relevant elements for that slide
      doc <- addTitle( doc, input$LeaderID)
```

```
42        doc <- addSubtitle( doc, paste(Sys.Date()))
43
44        # Add slide with figures – notice difference between adding base plots and ggplots
45        doc <- addSlide( doc, slide.layout = 'Project_Bar&PiePlots' )
46        doc <- addTitle( doc, "Here is some title text")
47        PlotA <- ggplot(dat, aes(Var2,Var3))+geom_point()
48        doc <- addPlot( doc, function() print(PlotA))
49        doc <- addPlot( doc, function() pie(piedat$CntVar1, labels=piedat$Var1))
50        doc <- addParagraph (doc, "This is a comment.")
51        doc <- addDate( doc )
52        doc <- addPageNumber( doc )
53
54        writeDoc( doc, file )
55      } # end of report  content function
56    ) # end of downloadHandler function
57
58  } # end of server function
59
60  # Run the application
61  shinyApp(ui = ui, server = server)
```