Paper OS03

Pharmula 1 – Driving Innovation in Reporting

Ellis Hughes, GSK, Seattle, USA

ABSTRACT

In the rapidly evolving landscape of Pharma, clinical programmers develop key insights in Tables, Listings, and Figures that help teams make decisions about clinical trial results. Taking these important insights to the rest of the team, especially to reuse them, is not trivial. To address this, we developed the BEJEWELED suite of tools, a collection of R packages designed to bridge the gap between our programming teams and non-programming staff, particularly our medical writers. This suite in part comprises of the recently open-sourced sparkle, mirage, and polish packages, each providing essential features to insert and format content seamlessly into Word and PowerPoint documents. In my talk I will share how we created, innovated, and deployed tools to reduce drag, saving valuable time between the creation of outputs and delivery of key reports and presentations.

INTRODUCTION

In the fast-paced world of the pharma industry, while everyone is working towards a common goal of delivering sfe and effective treatments, the way we work and how we work across different teams can have a massive impact on how well we work together. Handoffs between programming, statistics, and medical writing functions can be a source of pain when everyone is speaking a different language.

One common source of pain is data sharing, and making everything work nicely together. This is because data presentation tools common to the programming function – mainly PDF, is not an ideal data storage format, nor is it well know for interoperability with other reporting tools. Medical writers author their work in Microsoft Word. And statisticians present to stakeholders using Microsoft PowerPoint. Screen grabs work, copy paste work, but it is painful process especially when data accuracy is paramount.

Enter BEJEWELED and the R packages {polish}, {sparkle}, and {mirage}. These packages aim to resolve the common pains of data reuse across multiple platforms.

INTRODUCING BEJEWELED

BEJEWELED is a collection of packages are built to work off one another as an extensible set of packages that anyone can customize to fit their use cases. Microsoft office products are based on the “open office xml” (OOXML) standard, which means that the document source code can be easily modified, content added, and the output saved with simple code.

Integrating these packages into workflows can simplify processes, be integrated into other tools like R Shiny applications, and empower teams across the organization. Lets go into some details about the packages and look at some examples.

POLISH

The nucleus of BEJEWLED is in {polish}, where it introduces the core concept of the collection – polishing. Polishing is the conversion of an R object into the “open office xml” (OOXML) standard, so that its companion packages, {sparkle} and {mirage} can add the content into Microsoft Word documents and Microsoft PowerPoint presentations or completely make new ones.

The main exported function of {polish} is “polish_content()”. This implements two s3 methods, “polish_content_word()”, or “polish_content_pptx()”. It is these s3 methods that convert R objects into the OOXML that the Microsoft Office products understand. {polish} comes with a variety of common R object polishing methods fully implemented in word and powerpoint, including basic character, numeric, data.frames, to ggplot and flextable objects. R Objects can be easily converted into their OOXML representations via a call to “polish_content()” and passing the type (word or pptx).

```
library(polish)
polish_content(mtcars, type = “word”)
polish_content(mtcars, type = “pptx”)
```

In addition to basic R objects, {polish} includes some helpers to do things like pass strings “as is”, use markdown syntax for text, and pass common files (png, jpeg, pdf) as inputs.

It is important to note that the OOXML for Microsoft Word and Microsoft PowerPoint are different and use different schema, which is why there are different methods for the two.

This package is hosted on GitHub at https://github.com/GSK-Biostatistics/polish.

SPARKLE

If {polish} is the translator, you still need to add your outputs into your Microsoft Word documents, and tell it where to add it in. This is the remit of {sparkle}.

To “sparkle” a document, you need to tell the package what you want to add, and where to add it. {sparkle} takes advantage of the comment functionality to do just that, meaning content can be easily added either in-line, or as its own paragraph.

Preparing a document is as easy as adding one or multiple comments to the word document wherever the author wants new content aded. Once that is done, load it into your R session.

```
library(sparkle)
sparkle_doc_path <- system.file("basic_sparkle_doc.docx", package = "sparkle")
my_doc <- load_docx(path = sparkle_doc_path)
```

Once the document is loaded in, you can see locations {sparkle} found to potentially add content to by calling “comments_df()”.

```
comments_df(my_doc)
```

Now, to add programmed content to the document, we use the “add_sparkle()” function. This function takes at minimum two arguments, the comment_id, which is the id of the comment, which can be found by looking at the results of “comments_df()”, and the R object to be added in. “add_sparkle()” calls {polish} and the “polish_content_word()” function for you!

```
my_doc |>
  add_sparkle(comment_id = 0,value = mtcars[1:5,])
```

In addition to the basic object addition, you can also pass named arguments to “add_sparkle()” that also get passed on to the polishing methods. The document object is modified in-place, so you don’t have to assign it, though this is also valid.

To save the output, use “save_docx()”.

```
doc_out <- tempfile(fileext = ".docx")
save_docx( my_doc, doc_out)
```

While this example is relatively trivial, and you probably could have added your table manually in about a similar time, imagine adding 30-40 tables manually. It really adds up in time. This, and some logic around the outputs of “comments_df()” means users can potentially write for-loops around different text, either in the document or in the comments to inform your sparkling of your document, what files to grab, modifications to the tables to make, etc. This could also be embedded potentially into shiny applications to support non programming staff.

This package is hosted on GitHub at https://github.com/GSK-Biostatistics/sparkle.

MIRAGE

Where {sparkle} focuses on Microsoft Word, {mirage} is fully focused on Microsoft PowerPoint, and its unique needs. As was mentioned earlier, the OOXML that goes into Microsoft PowerPoint is different, but that is not the only difference. Content is not linear, it is placed onto slides, and the locations on these slides are non-linear.
To support this, {mirage} in turn does much more than {sparkle}. Not only does content get added, but users have the ability to completely construct a presentation, including new slides, via {mirage}, and identify where the content is to be added anywhere on the slide!

Just as before, the process starts with loading in a presentation. However, in addition to loading in an existing one, {mirage} comes built in with a basic PowerPoint template file too.

```
library(mirage) one;
pptx <- example_pptx()
# To load an existing presentation, use load_pptx()
# pptx <- load_pptx("Path/to/your/presentation.pptx")
```

Once the presentation is loaded, you can add, move, or delete slides. Here lets add three new slides. To add a slide, you need to specify at least the "layout" of the slide. These are the same names as you see in the Microsoft PowerPoint application when adding a new slide layout. Read the help pages for these functions, "add_slide()", "delete_slide()", and "move_slide()"

```
pptx |>
  add_slide("Title Slide") |>
  add_slide("Two Content 1") |>
  add_slide("Two Content 2")
```

To add a slide and add content, the same "add_slide()" function is used, but this time, we identify new content to be added by passing at least one "content()" function for each piece of data to be added to the slide. "content()" expects two arguments, the value to be added, and a location on the slide. This location, called a "placeholder" is where the content will be added to.

You are familiar with placeholders already, as they are the dashed-transparent boxes where you click and type in Microsoft PowerPoint application to put titles, bulleted sentences, and gifs. {mirage} comes with some basic helper functions to identify common placeholders:

- "ph_body()" is the largest placeholder on the slide
- "ph_title()" finds the title placeholder,
- "ph_body_left()" and "ph_body_right()" find the the largest placeholders on the slide and if there is a tie, selects the left or right most placeholder.

Lets add a title and the "mtcars" dataset to our new slide.

```
pptx |>
  add_slide(
     layout = "Title and Content",
     content("My Title", ph = ph_title()),
     content(mtcars, ph = ph_body())
  )
```

However, sometimes you may already have some slides, and just need to add some content. In this case, the "update_slide()" function is your friend. It works just like "add_slide()", but instead of passing a new slide layout, you provide the index of the slide to update.

```
pptx |>
  update_slide(
     index = 1,
     content("My Updated Title", ph = ph_title()),
     content(mtcars, ph = ph_body())
  )
```

In addition to using the pre-defined locations, users can define new placeholders to add content to if the shape does not fit their vision, or they want something unique. Rather than using a pre-defined "ph_*" function, they use "new_placeholder()", and define the height, width, x offset and y offset of the new spot to add content. Height and width are the height and width of the newly defined location, in either absolute units (inches, centimeters) or as a percentage of the slide. The x and y offset are how far from the top left corner you want the box to be from the left(x offset) and top (y offset), also in either absolute units (inches, centimeters) or as a percentage of the slide.
To add a ggplot of mtcars disp vs mpg and make it go across the page and to the bottom, we would do the following:

```
library(ggplot2)
pptx |>
  add_slide(
     layout = "Title and Content",
     content("My mtcars plot", ph = ph_title()),
     content(ggplot(mtcars) + geom_point(aes(x = disp, y = mpg)),
        ph = new_placeholder(height = "85%", width = "90%",
                             y_offset = "10%", x_offset = "5%"))
  )
```

Finally, to save this presentation, use "save_pptx()"

```
pptx_out <- tempfile(fileext = ".pptx")
save_pptx( pptx, pptx_out)
```

As you might imagine, setting up loops to quickly load in content, identify figures or other content to add and rapidly either update slides or make new slides with content can be easily done. Once templates exist, the easy by which new presentations can be created is a breeze.

This package is hosted on GitHub at https://github.com/GSK-Biostatistics/mirage.

CONCLUSIONS

These three packages represent an opportunity to connect three important teams that need to work in lock step together. These packages can also be built on and extended to support different functions across pharma, different data sources, and build connections across teams that previously found friction.

By making these packages open source, additional users are encouraged to use and contribute to them to make them more powerful.

DISCLAIMER

The contents of this paper is my personal opinion and not that of GSK.

CONTACT INFORMATION

(In case a reader wants to get in touch with you, please put your contact information at the end of the paper.)
Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Ellis Hughes
Company: GSK
Email:ellis.h.hughes@gsk.com
Website:gsk.com