PhUSE 2016

Paper TT07

R Beyond Statistics

Namrata Deshpande, Cytel, Pune, India

ABSTRACT

R today has many powerful packages to offer beyond just statistics. One such package is 'gWidgets', a package for toolkit-independent development of interactive graphical user interface (GUI) from R. The other one is 'ReporteRs' which generates well formatted reports as Microsoft Word® or PowerPoint® files from R. These two packages can leverage the power of the established statistical and graphical applications of R.

Automation is the key to higher productivity of a team. It not only saves efforts but also increases precision by minimizing human errors. But using sophisticated programming tools always may not be feasible due to cost constraints. R, with these powerful packages could be highly effective in automation of activities involving data analysis and report generation. The aim of this paper is to highlight the features of these two packages, 'gWidgets' and 'ReporteRs', and demonstrate how they could be put to use to enhance usability of R.

INTRODUCTION

Statistical strength of R is a well established fact. R packages offer a variety of statistical functions keeping up with the latest development in the field. The R graphical packages, especially 'ggplot2', with their high customization power have gained immense popularity in recent times. Thus R is fast being accepted by not only academia but the industry as well.

Since R is free and can be easily installed on any system without any licensing issues, it is a convenient option for exploratory analysis. Recently R has been enriched with some non-statistical packages as well which can leverage the use of its popular statistical packages. With the help of gWidgets, one can build a user friendly GUI that connects with the R statistical and graphical functionalities. This will enable even non-statisticians the execution of these exploratory tasks. Similarly, with ReporteRs we can automate the process of reporting the results that R generates. These two packages can increase efficiency of any repetitive R task.

GUI WITH gWidgets

The Graphical User Interface that R provides is very elementary and not very intuitive. Because of this, user needs a sound knowledge of R, its functions and also a decent programming ability to utilize power of R satisfactorily. gWidgets package authored by John Verzani addresses this concern and helps user to build interactive GUI in R very easily. It provides a toolkit independent API (Application Programming Interface) for building GUIs. In simpler words, it acts as a bridge between R and the GUI toolkits. Thus one of the toolkit packages like gWidgetstcltk, gWidgetsRGtk2, etc. needs to be installed along with gWidgets.

The basic widgets required to build a user friendly GUI such as file browser, button, checkbox, text box, radio buttons etc. are provided as simple built-in functions in gWidgets. These controls help to connect to the powerful R functions at the backend and could build as sophisticated interface as standard software. This enables ease of use of R for non-programmers and reduces dependency on programmers for regular execution of R functions.

EXAMPLE: A SAMPLE GUI

The following example will demonstrate the implementation of gWidgets.

An R program for exploratory studies is used to compare mean responses between two treatments. This is a routine activity and needs regular execution. The inputs to be specified are:

- The study name
- The input data file in CSV (Comma Separated Values) format.
- Column containing Treatment IDs
- Column containing Response values

These inputs are then passed to a function that performs statistical tests to compare the treatments as well as generates relevant graphs.

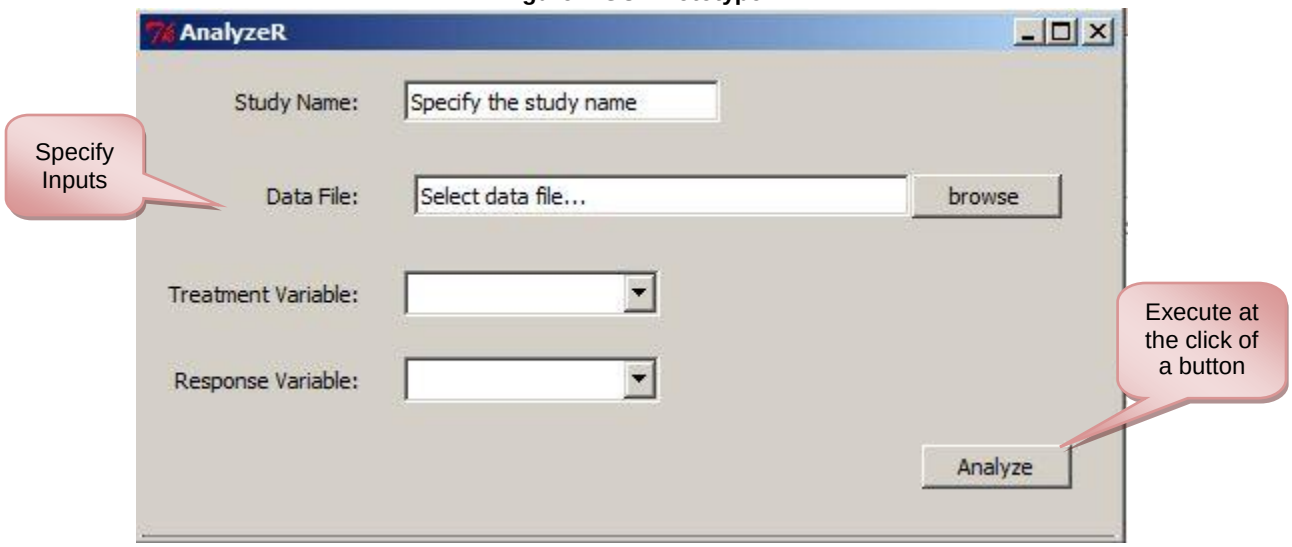
A glimpse of the code is as follows:

```
#A program to analyze raw data comparing two treatments #
#Inputs
Study_name <- "Study1234"
data_raw <- read.csv("E:\\data.csv")
```

```
Treatment_var <- "Treatment"
Response_var <- "Response"
#Extracting variables from data
Treatment <- data_raw[,Treatment_var]
Response <- data_raw[,Response_var]
#Bar plot for mean response
library(ggplot2)
Mean_resp <- aggregate(Response~Treatment, FUN= "mean")
plot1 <- ggplot(Mean_resp, aes(x=factor(Treatment),
         y=Response, fill=factor(Treatment), width = 0.5)) +
         geom_bar(stat="identity", position= "dodge") +
         xlab("Treatment") + ylab("Mean Response") +
         ggtitle("Comparison of Treatments")  +
         scale_fill_discrete(name="Treatment") +
         theme(plot.title = element_text(size = 14,
         lineheight=1, face="bold"),
         axis.text.x = element_text(size = 11),
         axis.text.y = element_text(size = 11))
#Two sample t-test
tout <- t.test(Response~Treatment)
```

Imagine the plight of a non-programmer asked to execute this code for a new study. The task of execution could be simplified many folds by building a user friendly GUI for input specifications. Following is a GUI prototype for the above piece of analysis code.

Figure1: GUI Prototype



Following is a step by step implementation of the widgets to build the GUI as per the prototype above:

Step 1: The Framework

- **`gwindow`** defines the outermost window of the tool. It is the base container which holds other widgets as seen in the adjoining figure.
- **`glayout`** helps in laying out the child widgets like text boxes, buttons in a tabular format using matrix notation. It helps in aligning a group of widgets in rows and columns for better appearance. Imagine the layout to be a table as seen in Figure 2.2 below.

Figure 2.1: The Main Window

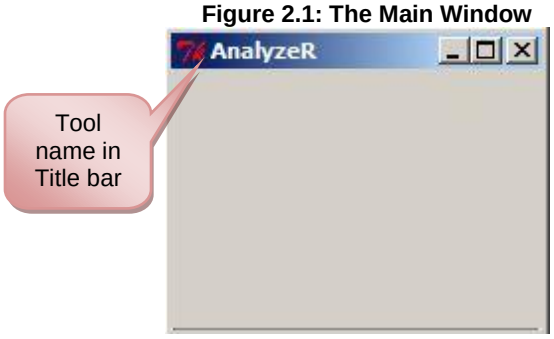
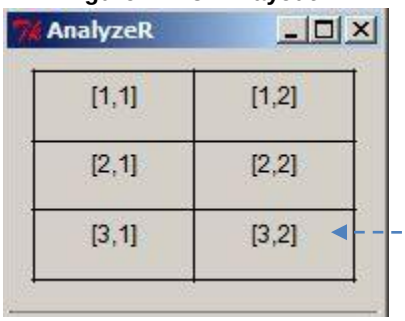


Figure 2.2: 3x2 Layout



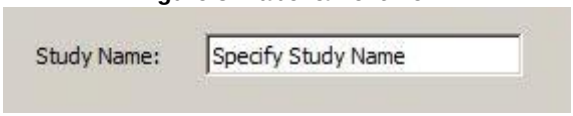
We can add child widgets to each cell of the table using matrix notation, for example `layout[3,2]` refers to the third row and second column in the table.

- **`anchor`** is used to align the widget within the cell of the layout. It is a vector of length two, like `c(-1,1)` . The first argument is for horizontal alignment, -1 for left alignment, 0 for center alignment and 1 for right alignment. Similarly the second argument is for vertical alignment, -1 for top alignment, 0 for center alignment and 1 for bottom alignment. Thus in the above figure, the widget placed in the third row, second column will be centre top aligned.

Step 2: Study name input

- The Study name is expected to be an alpha-numeric value which the user will type in. **`gedit`** is the function used to add the text box widget for such inputs.
- Using **`glabel`** add text on the GUI, mainly representing labels for other GUI components.
  The GUI with the label and text box for Study Name:

Figure 3: Label & Text Box



Step 3: Data specification

- Specification of data file is an integral part of any statistical analysis. Instead of typing in the file path and name it is always convenient to select the file using file browser.
- **`gfilebrowse`** provides the file browser widget to open a file.
- All possible file types for the input file like 'Excel Workbook (*.xlsx)' can be specified for ease of browsing.

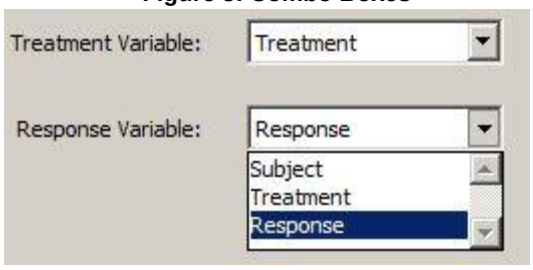
Figure 4: File Browser

Step 4: Mapping Treatment & Response

- The next step is to map the Treatment and Response columns in the input data.
- For this we use combo boxes. Combo boxes provide a list of values for the user to choose from.
- The headers of the data specified will be listed in these combo boxes so that the user can choose the appropriate columns with Treatment and Response values.
- **`gcombobox`** is the function that adds combo box as follows.

The combo boxes will be blank as long as no data is selected. As soon as a data file is selected, the combo boxes will be updated with the headers in the data as seen in Figure 5.

Figure 5: Combo Boxes



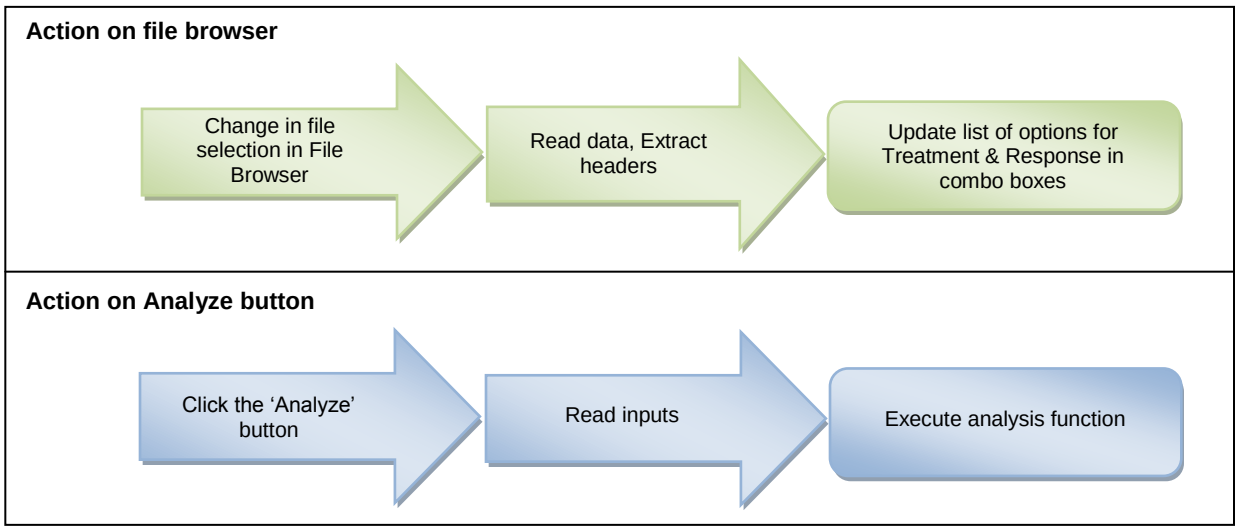
Step 5: Call Analysis Function

- Once all inputs are specified, the analysis/graphical functions need to be called.
- For this we add a button 'Analyze' using **`gbutton`**.
  Clicking this button passes the inputs to the analysis function.
  How? We explain this in the next step.

Figure 6: Button

Step 6: Handle Interdependencies

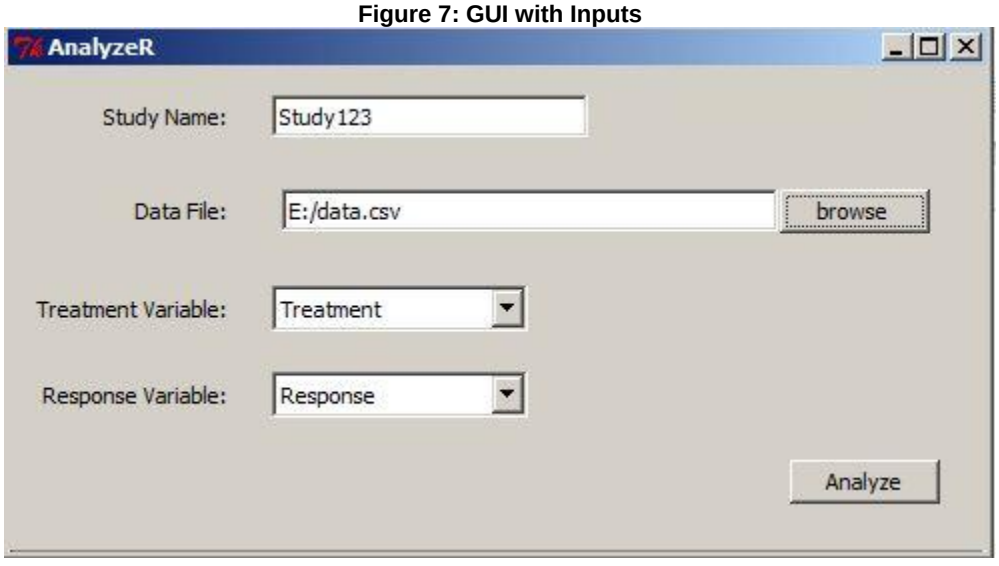
In any GUI there are several dependencies within the widgets. Like changing the selection for some combo box may have an effect on some other input. In the GUI that we have created, following are the interdependent actions to be handled:



Handler functions are used to specify these interdependent actions. These functions can be specified for any widget. They define the action to be taken each time the value of the linked widget is changed.
To handle the above actions in our example, handler functions are defined for the file browser and Analyze button widgets.

With this we complete the GUI for our analysis as per the prototype in Figure 1. An example of the GUI with inputs duly filled in:

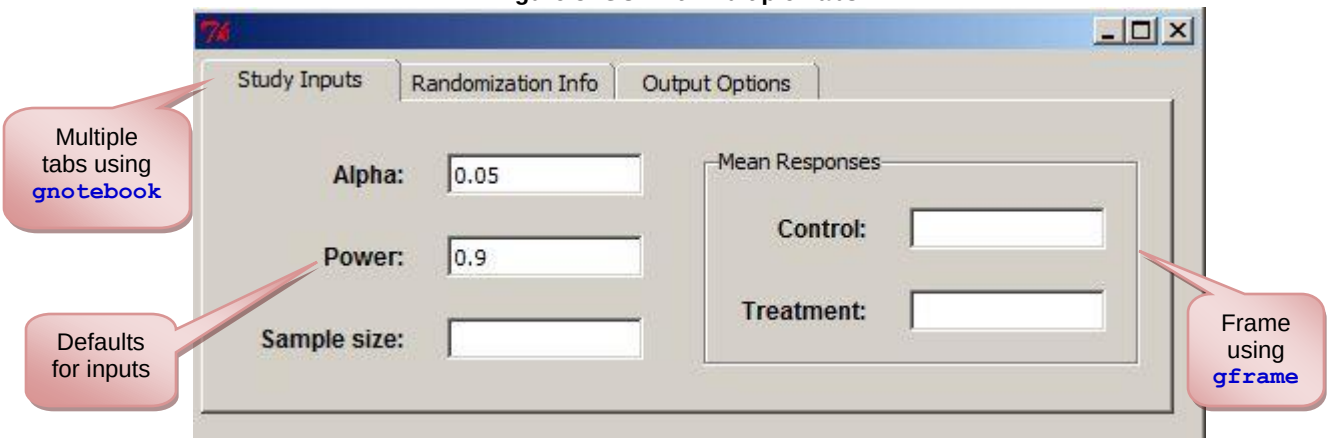
Figure 7: GUI with Inputs



- Note that we have covered just some basic functionality of gWidgets.
- This is a simplified GUI where we assume no error in data specification, etc. With gWidgets we can handle errors efficiently by popping up error messages in the form of child windows of the main window.

- gWidgets has many more functional as well as cosmetic features. For example, we could build more fancy GUIs using functionalities like **`gframe`** to add a frame to a set of related inputs or **`gnotebook`** to add multiple tabs to a window as seen below:

Figure 8: GUI with Multiple Tabs



REPORTING WITH ReporteRs

R these days is extensively used for analysis and mainly for graphics. But extracting the outputs, tables, graphs from R and putting them in a well formatted report could be a tedious job. Especially if a similar kind of analysis is to be repeated periodically then it is worthwhile automating the process of report generation in a standardized format. Package 'ReporteRs' authored by David Gohel et al. provides tools to export R outputs to Microsoft Word, Microsoft PowerPoint and html documents. Most of the functions in ReporteRs can be used with any of the document types. We will mainly look at the creation of a PowerPoint report in this paper. We will discuss some of the key functionalities of ReporteRs like using saved templates, creating slides with customized text, tables and graphs.

EXAMPLE: POWERPOINT REPORT

Study Summary:
This is a consumer study where each subject assesses 3 prototypes in terms of 5 attributes. Some study details:

- Study: MKN-2344
- Number of Prototypes: 3
- Number of Subjects: 21
- Attributes: Mint, Sweet, Bitter, Citrus, Aroma

Analysis:

- Prototype wise mean grades for attributes
- Bar chart comparing Prototypes

A Glimpse of Grade data

Subject	Prototype	Mint	Sweet	Bitter	Citrus	Aroma
1	OL-46	5.0	5.5	0.0	3.0	0.0
2	OL-46	5.0	6.0	0.0	3.0	2.0
3	OL-46	4.5	6.5	1.5	3.5	2.0
4	OL-46	5.0	6.0	0.0	3.0	1.5
5	OL-46	4.5	5.5	0.0	3.0	2.0

Steps to generate PowerPoint report for the analysis:

Step 1: Apply Templates

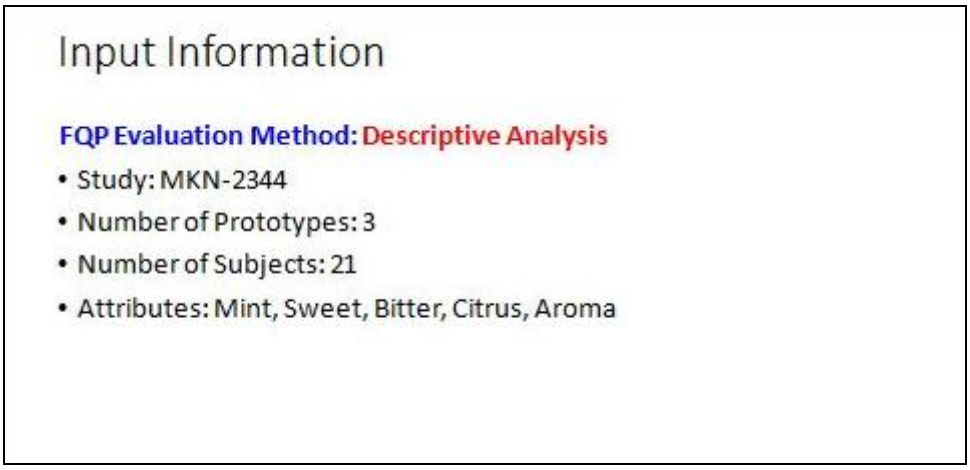
In most of the organizations it is mandatory to generate reports with company templates. Templates are standardized files with pre-specified font size, font style, etc. fixed and may even contain the company logo. While generating Word or PowerPoint reports, ReporteRs enables the user to specify the template file. Though a trivial functionality, its utility for routinely generated reports is very critical.

Step 2: Add Title Slide

- Add title to the slide using function **`addTitle`**.
- Add subtitle using function **`addSubtitle`**.
- Add date to the slide using function **`addDate`**.

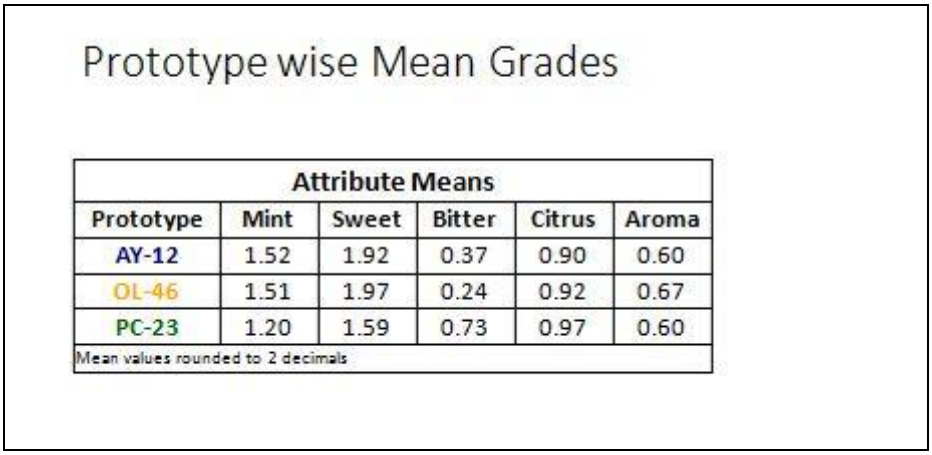
Note that to use these direct functions to add specific details, the slide layouts need to have text area for those details. For example, to add a subtitle using function **`addSubtitle`** the layout needs to have a provision for a subtitle.

Step 3: Add Text



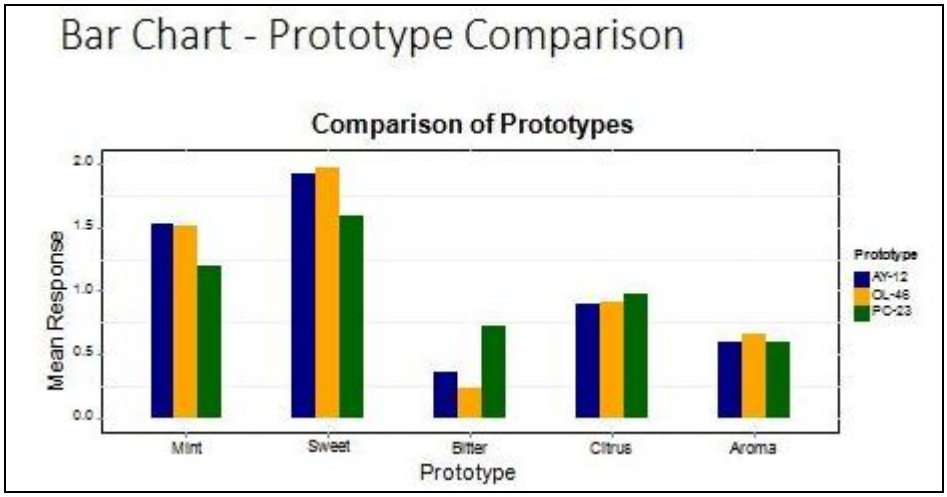
- Add text to the content slides using **`addParagraph`**.
- Use **`set_of_paragraphs`** to create bullet points.
- **`textProperties`** define text format like font size, font color, etc. **`parProperties`** define paragraph format like alignment, padding, border etc.
- **`pot`** is a very useful function which stores text as well as its formatting properties. It can be used to create a text with varied formatting properties. For example '**An *example* of pot function**.'

Step 4: Add Table



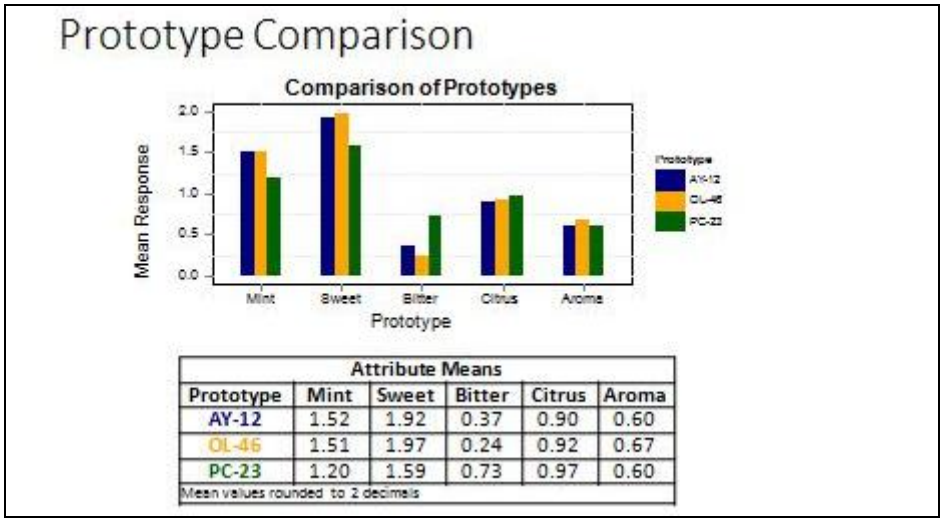
- Prepare well formatted tables using function **`FlexTable`**.
- Ease of formatting in terms of font, alignment, cell merging, etc.
- In addition to **`textProperties`** and **`parProperties`** as described above, tables also have **`cellProperties`** that define cell format like alignment, padding within the cell.
- Some more table functions: **`addHeader`** to add a header row, **`addFooter`** to add footer row, **`setFlexTableWidths`** to adjust column widths.
- Use **`addFlexTable`** to insert the table created using **`FlexTable`** into the document.

Step 5: Add Plot



- Add plots with the simple command **`AddPlot`**.
- We can specify the width, height of the plot area to be pasted in the document.
- The plot added to the document could be kept editable.

With this, all the outputs from our example have been added to the PowerPoint report. Once all objects are inserted we just need to save the document using **`writeDoc`**. These were some basic capabilities of ReporteRs. It has many more advanced features to create more intricate reports. For example, ReporteRs enables us to generate customized slides with multiple objects using X-Y coordinate positions of the slide area as follows:

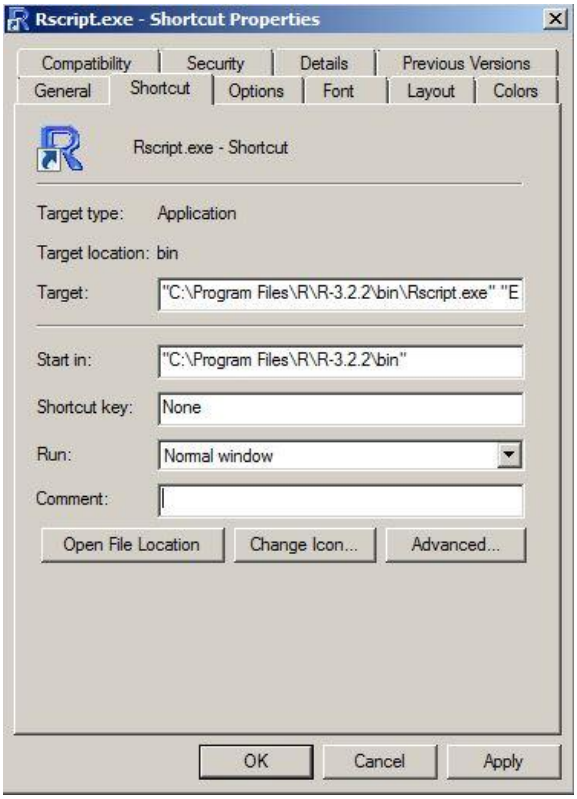


Attribute Means					
Prototype	Mint	Sweet	Bitter	Citrus	Aroma
AY-12	1.52	1.92	0.37	0.90	0.60
OL-46	1.51	1.97	0.24	0.92	0.67
PC-23	1.20	1.59	0.73	0.97	0.60

Mean values rounded to 2 decimals

DESKTOP APPLICATION

- Create a shortcut of Rscript.exe found in the 'bin' folder at the R installation path. For example: "C:\Program Files\R\R-3.2.2\bin"
- Right click on this shortcut and select Properties. The Properties window will pop up like in the adjacent image.
- In the 'Target' field add the RScript.exe file and path followed by the R program file and path to be executed through this shortcut. For example: "C:\Program Files\R\R-3.2.2\bin\Rscript.exe" "E:\gui_code.R"
- In the 'Start in' field add the Rscript.exe path. For example: "C:\Program Files\R\R-3.2.2\bin"
- Click on 'Apply' and close the Properties window.
- The shortcut is now an executable file. You could rename it with an appropriate name.
- With just a double click of this .exe file, the R program mentioned in Target above will be executed automatically.



A shortcut of this kind for the gWidgets code would just bring up the GUI for inputs. The analysis functions could be called at the click of a button on the GUI.
Moreover if ReporteRs is used for report generation at the end of the analysis, the task of execution could be completely achieved without even opening the R console.

CONCLUSION

Functions offered by gWidgets are intuitive and do not need any core software development skills for GUI programming. An R programmer with basic designing skills can come up with an efficient GUI with gWidgets. A simple interface to accept inputs to a complex function and execution with the click of a button could ease the task of execution, particularly for non-programmers. Thus R in the form of a tool would definitely increase the scope of its usage.

Similarly, ReporteRs could help automate the process of report generation from R like any standard software. Especially if an analysis is performed routinely through R, the effort to develop a ReporteRs function to generate a Word or PowerPoint report instantly would definitely be worthwhile. For example, consider a situation where such an activity is to be performed once a week and takes 4 hours each time to prepare the report. Then a onetime investment of approximately 20 hours of efforts in developing a function to automate this report generation process using ReporteRs could reduce these 4 hours to 2 minutes, saving around 200 hours of efforts per year.

R today is gaining wider acceptance and will surely see more applications in the near future. gWidgets and ReporteRs together can end-to-end automate any R process and hence leverage the strength of R's statistical packages. This will certainly add sophistication to R's user experience taking it to the next level.

REFERENCES

https://cran.r-project.org/web/packages/gWidgets/index.html

https://cran.r-project.org/web/packages/gWidgets/vignettes/gWidgets.pdf

https://cran.r-project.org/web/packages/ReporteRs/index.html

http://davidgohel.github.io/ReporteRs/

http://www.coppelia.io/2012/06/r-creating-a-shortcut-to-run-a-gwidgets-gui/

ACKNOWLEDGMENTS

I would like to thank my managers and colleagues at Cytel (India) for their support and cooperation. Special thanks to Manjusha Gode for her valuable review and inputs.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Namrata Deshpande
Cytel Statistical Software & Services Pvt. Ltd.
'Lohia Jain IT Park', A wing, 6th Floor
Pune – 411038
India

Work Phone: +91-20-67090153
Email: namrata.deshpande@cytel.com
Web: www.cytel.com

APPENDIX

Code for GUI with gWidgets:

```
#Step 1: The Framework
library(gWidgets)                                  #Load package gWidgets
options("guiToolkit"="tcltk")                      #Choose 'tcltk' as the toolkit option
main_window <- gwindow(title = "AnalyzeR")         #Main window for the GUI
main_layout <- glayout(container = main_window)    #Layout for child widgets

Step 2: Text Box and Labels
Study_name_label <- glabel(text = "Study Name:",       #Define label for
                           cont = main_layout)         #Study Name text box
Study_name <- gedit(text = "Specify Study Name",       #Text box
                    cont = main_layout)
main_layout[1,1, anchor = c(1,1)] <- Study_name_label  #Insert label in the layout
main_layout[1,2,anchor = c(-1,1)] <- Study_name        #Insert text box in the layout

#Step 3: File Browser
Specify_file_label <- glabel("Data File:",                 #Label for File browser
                      cont = main_layout)
main_layout[2,1, anchor = c(1,1)] <- Specify_file_label    #Insert label in the layout.
main_layout[2,2, anchor = c(-1,1)] <- gfilebrowse(         #File browser to select data
                    text="Select data file...",            #file of type *.csv. Handler
                    width = 30,                            #function 'Update_Fields'
                    filter = list("CSV (Comma delimited)" #updates dependent fields
                    = list(patterns = c("*.csv"))),        #each time input file is
                    container = main_layout,               #modified.
                    handler = function(h,...)
                    Update_Fields())

#Step 4: Combo Boxes
Treatment_var_label <- glabel("Treatment Variable:",     #Label for Treatment variable
                       cont = main_layout)
Response_var_label <- glabel("Response Variable:",       #Label for Response variable
                      cont = main_layout)
Treatment_combo <- gcombobox(items = "", width = 100,    #Combo box for selecting
                   selected = 0, cont = main_layout)     #treatment variable
Response_combo <- gcombobox(items = "", width = 100,     #Combo box for selecting
                  selected = 0, cont = main_layout)      #response variable
#Insert the labels and combo boxes in the layout
main_layout[3,1, anchor = c(1,1)]  <- Treatment_var_label
main_layout[3,2, anchor = c(-1,1)] <- Treatment_combo
main_layout[4,1, anchor = c(1,1)]  <- Response_var_label
main_layout[4,2, anchor = c(-1,1)] <- Response_combo

#Handler Functions
Update_Fields <- function()
{
  data_file_name <- gsub("/","//",                    #File path in R format
                    svalue(main_layout[2,2]))
  if(data_file_name != "" | !is.na(data_file_name)) #Check for invalid input
  {
    input_data <- read.csv(data_file_name)           #Read input *.csv file
    column_names <- colnames(input_data)             #Read column names
    Treatment_combo[] <- column_names                #Assign list of column names to
    Response_combo[] <- column_names                 #Treatment & Response combo boxes.
  }else
  {                                                  #If the relevant column names are
    Treatment_combo[] <- ""                          #not available then keep the
    Response_combo[] <- ""                           #selection blank. User will then
                                                     #manually make a selection.
  }
}
#Buttons
Analyze_button <- gbutton(text = "Analyze",          #Define the button with label
```

```
                  container = main_layout )          #'Analyze'
main_layout[5,2, anchor = c(1,1)] <- Analyze_button #Insert the button in the layout

#Handler function for Analyze_button
addhandlerchanged(Analyze_button, handler=function(h,...)
{
  Study_name <- svalue(Study_name)                    #Each time the Analyze button is
  data_file_name <- gsub("/","//",                    #clicked this handler function
                    svalue(main_layout[2,2]))         #will be called. It will read the
  input_data <- read.csv(data_file_name)              #input file, fetch the study
  Treatment_var <- svalue(Treatment_combo)            #name, selection for Treatment
  Response_var <- svalue(Response_combo)              #and Response variables and pass
  Analysis_function(Study_name,                       #these to the main analysis
                    input_data,                       #function, in this case
                    Treatment_var,                    #'Analysis_function'.
                    Response_var)
})
```

Code for report generation with ReporteRs:

```
# Load packages
library(dplyr);
library(ggplot2);
library(ReporteRs)

#Analysis code
data_raw <- read.csv("data.csv")
melted <- melt(data_raw,id.vars =c("Subject","Prototype"))
grouped <- group_by(melted, Prototype, variable)

#Mean computation
summ <- as.data.frame(summarise(grouped, mean=mean(value)))
names(summ)[2] <- "Attribute"
summ$mean <- round(summ$mean,2)
summ_tab <- reshape(summ, idvar = "Prototype", timevar = "Attribute",
            direction = "wide")

#Bar plot for mean response
plot1 <- ggplot(summ, aes(x=factor(Attribute), y=mean, fill=factor(Prototype),
         width = 0.5 )) +  scale_fill_manual(name="Prototype", values=c("navy",
         "orange", "darkgreen")) +   geom_bar(stat="identity", position= "dodge") +
         xlab("Prototype") + ylab("Mean Response") + ggtitle("Comparison of
         Prototypes") + scale_y_continuous(limits = c(0,2)) + theme(plot.title =
         element_text(size = 14, lineheight=1, face="bold"), axis.text.x =
         element_text(size = 11), axis.text.y = element_text(size = 11),
         panel.background = element_rect(fill = 'white', colour='black'))

#Report generation:
mydoc <- pptx( )                                      #Define a presentation object
mydoc <- addSlide( mydoc, "Title Slide")              #Add the Title Slide
mydoc <- addDate(mydoc, Sys.time())                   #Add date for reference
mydoc <- addTitle( mydoc, "MKN-2344 Report")          #Add title
mydoc <- addSubtitle( mydoc, "Exploratory Analysis")  #Add title

#Slide 1: Text slide
mydoc <- addSlide( mydoc, "Title and Content")              #Add Title & Content slide
mydoc <- addTitle( mydoc, "Input Information")              #Add title
text1 <- pot(paste("FQP Evaluation Method: "),              #An example of pot.
         format = textProperties(font.family = "Calibri",   #Multiple formatting
         font.size = 28, font.weight = "bold", color =      #properties.
         "blue")) + pot(paste("Descriptive Analysis"),
         format = textProperties(font.family = "Calibri",
         font.size = 28, font.weight = "bold", color =
         "red"))
mydoc <- addParagraph(mydoc, text1, par.properties =        #Add text to doc
```

```
         parProperties(text.align="left"))
text2 <- "Study: MKN-2344"                                  #Set of texts to be added
text3 <- "Number of Prototypes: 3"
text4 <- "Number of Subjects: 21"
text5 <- "Attributes: Mint, Sweet, Bitter, Citrus, Aroma"
my.par <- set_of_paragraphs(text2, text3, text4, text5)     #Bullet points
mydoc <- addParagraph(mydoc, my.par,append=TRUE)            #Add bullet points to doc

#Prepare customized table for Means
mean_table <- FlexTable(numrow = nrow(summ_tab),
              numcol = ncol(summ_tab), header.columns = FALSE,
              body.cell.props = cellProperties(padding = 0, vertical.align = "middle"),
              body.par.props = parProperties(text.align = "center"),
              body.text.props = textProperties(font.family = "Calibri",
              font.size = 20))

#Cell wise formatting
mean_table[1,1] <- pot(summ_tab[1,1], format = textProperties(font.family =
                   "Calibri", font.size = 20, font.weight = "bold", color = "navy"))
mean_table[2,1] <- pot(summ_tab[2,1], format = textProperties(font.family = "Calibri",
                   font.size = 20, font.weight = "bold", color = "orange"))
mean_table[3,1] <- pot(summ_tab[3,1], format = textProperties(font.family = "Calibri",
                   font.size = 20, font.weight = "bold", color = "darkgreen"))
mean_table[,2:6] <- summ_tab[,2:6]
#Add first line of headers
mean_table <- addHeaderRow(mean_table, value="Attribute Means", colspan = 6,
              par.properties = parProperties(text.align = "center"),
              text.properties = textProperties(font.family = "Calibri",
              font.size = 24, font.weight = "bold"))
# Add second line of headers
mean_table <- addHeaderRow(mean_table,
              value = c("Prototype", "Mint", "Sweet", "Bitter", "Citrus", "Aroma"),
              par.properties = parProperties(text.align = "center"),
              text.properties = textProperties(font.family = "Calibri",
              font.size = 20, font.weight = "bold"))
#Set table column widths
setFlexTableWidths(mean_table, c(1.5,rep(1,5)))
#Add footer row
footerRow <- FlexRow("Mean values rounded to 2 decimals", colspan=6,
             text.properties = textProperties(font.family = "Calibri", font.size = 16))
addFooterRow(mean_table, footerRow)

#Add table to document
mydoc <- addSlide(mydoc, "Title and Content")
mydoc <- addTitle(mydoc, "Prototype wise Mean Grades")
mydoc <- addFlexTable(mydoc, mean_table)

#Add graph to presentation
mydoc <- addSlide( mydoc, "Title and Content")
mydoc <- addTitle( mydoc, "Bar Chart - Prototype Comparison")
mydoc <- ReporteRs::addPlot(mydoc, fun = print, x = plot1)

#Save PowerPoint file
writeDoc(mydoc, file = "Analysis Report.pptx")
```