Paper OS10

Reproducibility of Interactive Analyses

Matthew Kumar, Bayer Inc., Mississauga, Canada
Srinivas Veeragoni, Bayer US LLC, Whippany, USA

ABSTRACT

R and Shiny have seen significant adoption in pharmaceutical data science, owing to the relative ease with which an interactive application can be created and shared to non-technical audiences to engage with. They have been employed in numerous analysis and reporting activities such as delivering insight dashboards, performing exploratory analyses, generating submission dossiers, and aiding in medical review. These apps are evolving from an exploratory state to pivotal decision-making tools in many facets of clinical analysis and operations. This makes it increasingly imperative to being able to replicate the novel insights (derived from user-driven interaction) outside the application in a traceable setting that can be shared with regulators. In this paper, we share our learnings in exploring reproducibility and Shiny. We present our initial use-case, methodology and discuss both practical and technical considerations. Lastly, we outline how leveraging code reproducibility could enable efficiencies in helping programmers to learn open-source languages.

INTRODUCTION

Modern computing technologies have seen considerable adoption in pharmaceutical data science, notably in the statistical programming space, where they have played a transformative role[1] in the analysis, reporting and presentation of clinical study data. The open-source R[2] language is gaining popularity in the industry, in part because of its extensive statistical and graphical capabilities. Additionally, its suite of productivity tools allows for easy creation and dissemination of analyses in a variety of interactive formats.

Shiny† is an R package[3] that provides a framework to author interactive web applications. Using the Shiny package, web apps can be created by means of pure R code and no web-development experience is required. Shiny has been applied in different contexts that enable non-technical users to consume analyses (or data) from the comfort of a web browser. The reactive programming model employed by Shiny makes it particularly well suited for natural experimentation and investigation. Shiny apps respond dynamically to input, permitting users to self-explore the data and test alternate “what if” scenarios to gain insights in real time. Some examples of Shiny apps include study metric or medical review dashboards, performing on-the-fly subgroup analysis, or standalone interactive data visualizations. Additionally, Shiny apps can be used to build general purpose tools to assist in routine tasks such as data quality checking, as well as streamlining existing processes like automated report generation and delivery.

To transition the use of Shiny from an exploratory state to a pivotal decision-making tool used in production[4,5], reproducibility is essential. This poses a challenge since on the surface, user-derived insights are arrived at through a series of user-driven interactions, such as clicking a button, selecting an option from a drop-down menu, or interacting with a filter. Beneath the surface, the R code and the app’s design itself could involve several intricate steps, making it less straightforward to replicate insights in another environment. In this paper, we share our journey in exploring reproducibility and Shiny, with an emphasis on code generation using the *shinymeta* package. We present a data visualization-based case study and discuss practical and technical lessons learned along the way.

CASE STUDY

MOTIVATION

The primary motivation of this case study was to investigate how one might reproduce a result created in a Shiny app in an external environment using code. For the basis of our case study, we chose to focus on reproducing a waterfall plot created using the ggplot2[6] package. The waterfall plot[7] is frequently produced in Oncology trials to summarize change in tumor size and typically requires graphical customization based on a variety of trial design considerations. This made it a leading candidate to explore given the reasonable amount of complexity in specification and coding we would likely encounter.

A secondary motivation of this case study was to explore tangible ways to leverage and integrate any potential outcomes from the primary investigation in day-to-day programming tasks. This is especially important since statistical analysts are increasingly needing to use R to produce deliverables whose prior experience or comfort with R language may vary.

† Shiny is in alpha for Python. See https://shiny.rstudio.com/py/

Together, these two motivations converged on prototyping a Shiny app that allowed statistical analysts to interactively specify a waterfall plot, export the underlying code and seamlessly execute it in a production environment to reproduce the deliverable.

Our first steps for this exploration were to identify key members of our function that could contribute to the technical and content expertise necessary for fruitful exploration. We assembled a small team consisting of computational scientists, statistical analysts, and a data engineer to work collaboratively on different facets of the prototype. Collectively, the experiences included prior R or statistical programming, Shiny app development and deployment, familiarity with internal systems or processes and domain knowledge for producing deliverables and practical code versioning knowledge.

INITIAL SETUP

A major task was to understand how the R instance was setup and configured in the production environment. The production environment represents the destination location of where the final R code will be executed, and where results will be produced for delivery. Briefly, the production environment operates with a controlled version of R and contains a fixed collection of packages that have been validated for use in creating analytical or graphical output. Understanding both components was a necessary first step in structuring a separate, though consistent environment where development efforts could take place. The *renv*[8] package was helpful in facilitating this by producing a snapshot of the R environment and associated package dependencies. The snapshot (i.e. *lockfile)* could then be used as a portable and programmatic basis to recreate the environment elsewhere. Lastly, an internal git repository was used to manage all code versioning during prototype experimentation and development.

CODE GENERATION

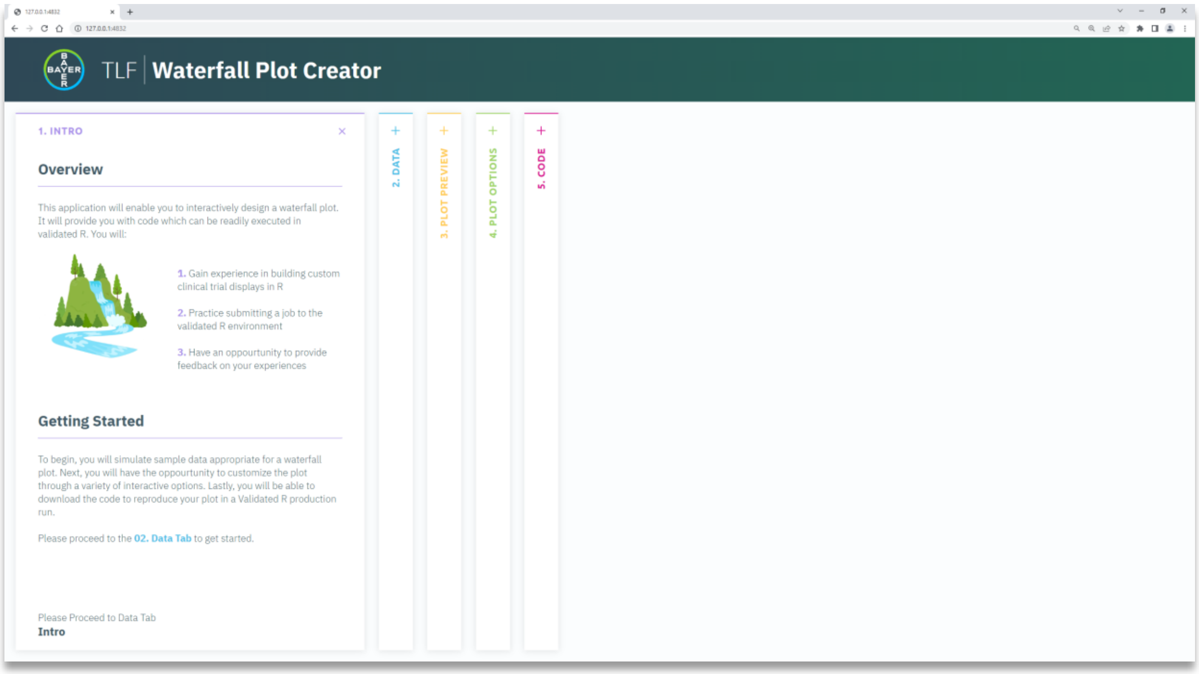
Code generation features of the prototype were implemented using the *shinymeta*[9] package. Shinymeta offers a suite of tools that help capture the essential logic in a Shiny app and 'convert' it to code that can be executed outside of the app environment. The interface of shinymeta presents developers with an alternative set of reactivity components that can be used for programming the app's server logic. These components graciously mimic the familiar reactive components and makes getting started relatively straightforward. The package vignettes provide clearly worked examples and guidance for more advanced usage.

PROTOTYPE: OVERVIEW

This section of the paper shares an overview of the final prototype developed in this case study. Each major section of the Shiny app is described with accompanying snapshots. The overall design of the app consists of a series of *ordered*, collapsible panels[10] which requires a user progress through to create the final plot and produce the associated code.

INTRO

The intro panel expands upon visiting the web app. It provides an overview and intent of the app. It also provides an explicit first set of instructions to proceed to the next panel to orient the user to the interface and flow of the app.



Overview

This application will enable you to interactively design a waterfall plot. It will provide you with code which can be readily executed in validated R. You will:

1. Gain experience in building custom clinical trial displays in R
2. Practice submitting a job to the validated R environment
3. Have an oppourtunity to provide feedback on your experiences

Getting Started

To begin, you will simulate sample data appropriate for a waterfall plot. Next, you will have the oppourtunity to customize the plot through a variety of interactive options. Lastly, you will be able to download the code to reproduce your plot in a Validated R production run.

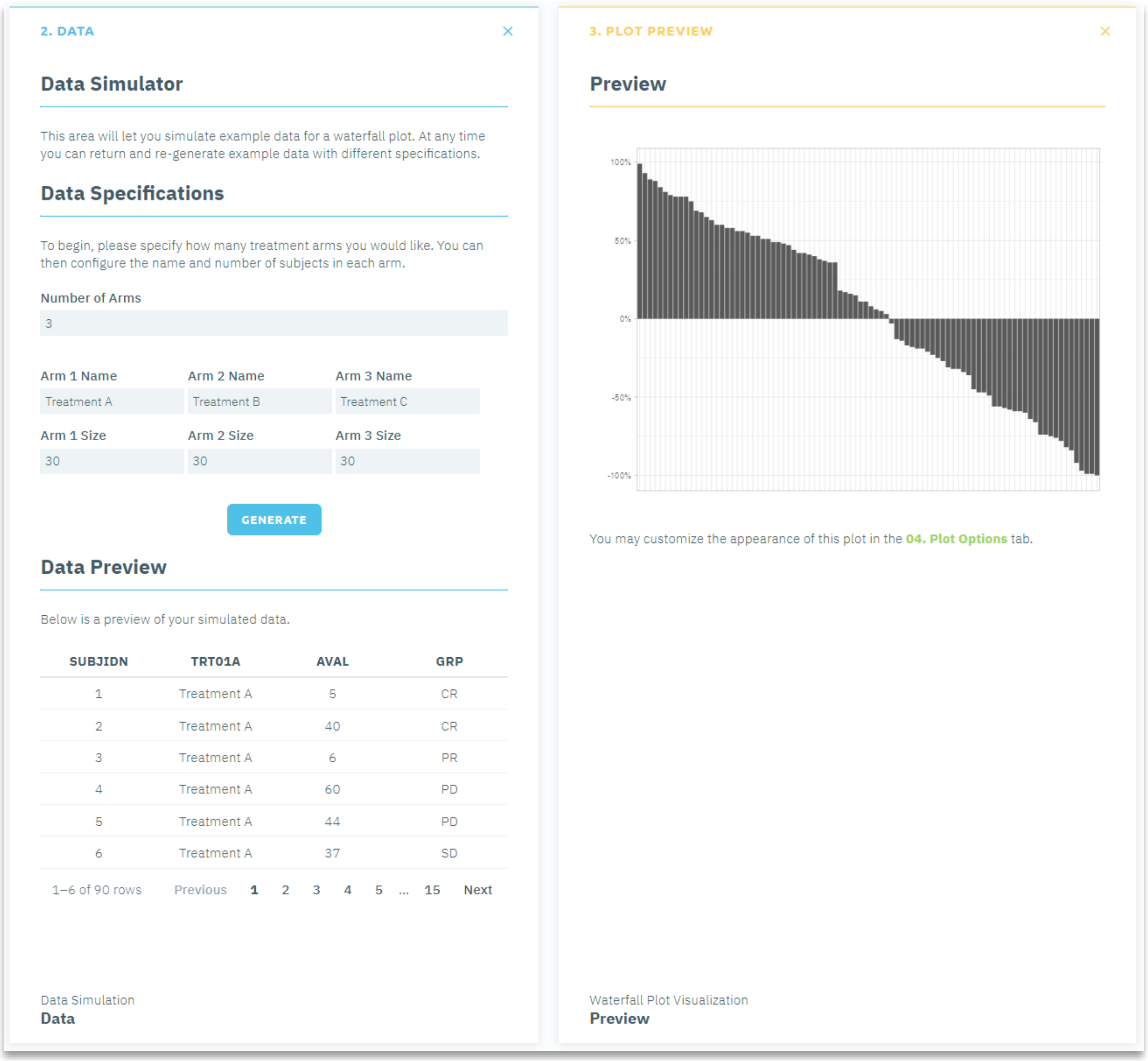
Please proceed to the **02. Data Tab** to get started.

Please Proceed to Data Tab
Intro

DATA
In this panel, users are required to provide input to assist in generating simulated data used by app. The first parameter is the number of treatment arms. Once entered, a corresponding set of controls that let the user specify the names and number of subjects of each arm are generated. When the generate button is clicked, the data simulation is executed and available for use in the app.

A tabular preview of the simulated data is also provided. This serves two purposes: 1) confirmation of the inputs they entered and 2) a feel for what the underlying data structure is (i.e. one row per subject) and variable characteristics are (i.e. names, types, values).

PLOT PREVIEW
In this panel, the base waterfall plot is created using the simulated data and automatically displayed. This panel is continuously updated to provide the user a preview every time a customization is selected (or changed) in the PLOT OPTIONS panel.



Data Simulator

This area will let you simulate example data for a waterfall plot. At any time you can return and re-generate example data with different specifications.

Data Specifications

To begin, please specify how many treatment arms you would like. You can then configure the name and number of subjects in each arm.

Number of Arms: 3

Arm 1 Name: Treatment A
Arm 2 Name: Treatment B
Arm 3 Name: Treatment C
Arm 1 Size: 30
Arm 2 Size: 30
Arm 3 Size: 30

Data Preview

Below is a preview of your simulated data.

SUBJIDN	TRT01A	AVAL	GRP
1	Treatment A	5	CR
2	Treatment A	40	CR
3	Treatment A	6	PR
4	Treatment A	60	PD
5	Treatment A	44	PD
6	Treatment A	37	SD

1–6 of 90 rows Previous 1 2 3 4 5 … 15 Next

Data Simulation
Data

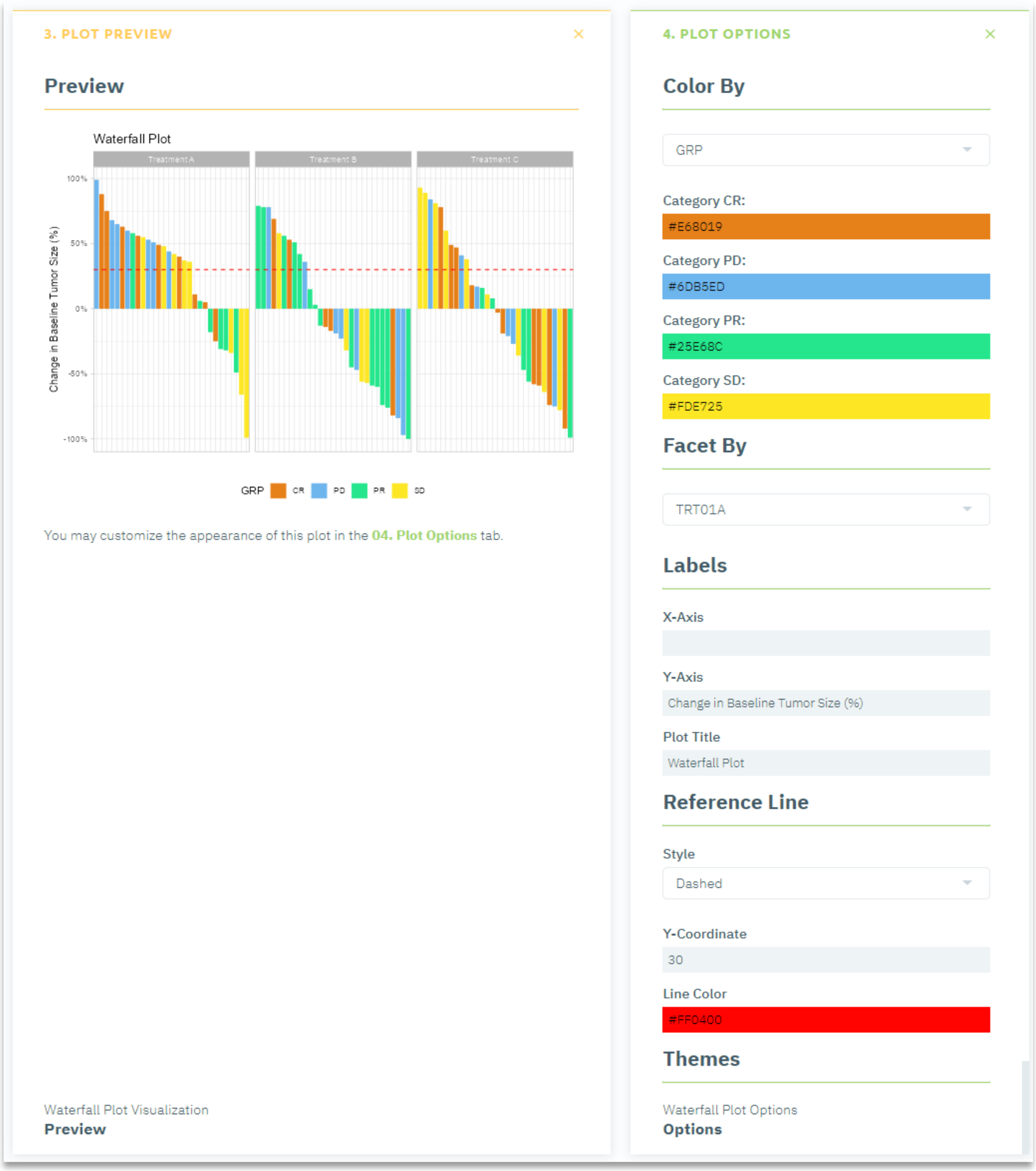
Preview

You may customize the appearance of this plot in the **04. Plot Options** tab.

Waterfall Plot Visualization
Preview

PLOT OPTIONS
In this panel, users can freely enhance the base waterfall plot with a handful of frequent customizations. They include:

- *Color By* – The user selects both the coloring (fill) variable and a color for each of its levels
- *Facet By* – The user selects which variable should be used as a faceting variable, if any
- *Labels* – Display text for the x axis, y axis and title of the plot may be directly entered here
- *Reference Line* – If a horizontal reference line is desired, the y-coordinate, type and color of the line is specified by user
- *Theme* – a user may select from 1 of 4 ggplot2 themes (Not shown below)



You may customize the appearance of this plot in the **04. Plot Options** tab.

Waterfall Plot Visualization
Preview

4. PLOT OPTIONS

Color By

GRP

Category CR: #E68019

Category PD: #6DB5ED

Category PR: #25E68C

Category SD: #FDE725

Facet By

TRT01A

Labels

X-Axis

Y-Axis
Change in Baseline Tumor Size (%)

Plot Title
Waterfall Plot

Reference Line

Style
Dashed

Y-Coordinate
30

Line Color
#FF0400

Themes

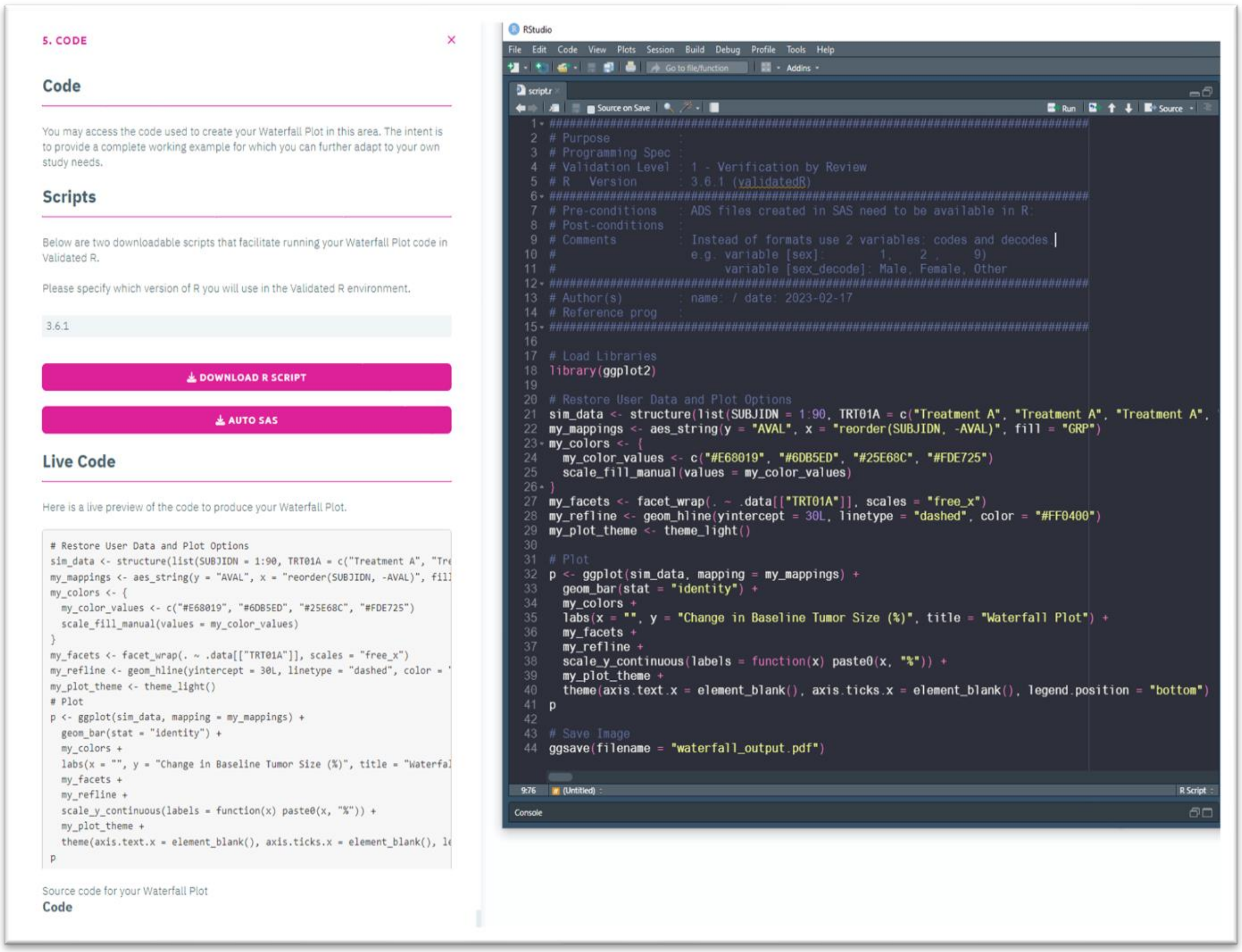
Waterfall Plot Options
Options

CODE
In this panel, users can view the code associated with the waterfall plot. Like the PLOT PREVIEW panel, it is continuously updated as a user makes changes in the PLOT OPTIONS panel.

Users may also download the code as an executable R script file. The R script file has a few formatting features:

- Clearly delineated sections of the R script, with comments
- Contains a standardized programming header and with a date stamp (lines 1-15)
- Setup code that loads the required libraries (line 18)
- Code to facilitate exporting results to a physical output file (line 44)

Lastly, the simulated data is recreated verbatim (line 21). This helps programmers reproduce the result in an end-to-end fashion and provides initial direction on how to adapt the example in their own work.



Code

You may access the code used to create your Waterfall Plot in this area. The intent is to provide a complete working example for which you can further adapt to your own study needs.

Scripts

Below are two downloadable scripts that facilitate running your Waterfall Plot code in Validated R.

Please specify which version of R you will use in the Validated R environment.

3.6.1

Live Code

Here is a live preview of the code to produce your Waterfall Plot.

```
# Restore User Data and Plot Options
sim_data <- structure(list(SUBJIDN = 1:90, TRT01A = c("Treatment A", "Tre
my_mappings <- aes_string(y = "AVAL", x = "reorder(SUBJIDN, -AVAL)", fill
my_colors <- {
  my_color_values <- c("#E68019", "#6DB5ED", "#25E68C", "#FDE725")
  scale_fill_manual(values = my_color_values)
}
my_facets <- facet_wrap(. ~ .data[["TRT01A"]], scales = "free_x")
my_refline <- geom_hline(yintercept = 30L, linetype = "dashed", color = '
my_plot_theme <- theme_light()
# Plot
p <- ggplot(sim_data, mapping = my_mappings) +
  geom_bar(stat = "identity") +
  my_colors +
  labs(x = "", y = "Change in Baseline Tumor Size (%)", title = "Waterfal
  my_facets +
  my_refline +
  scale_y_continuous(labels = function(x) paste0(x, "%")) +
  my_plot_theme +
  theme(axis.text.x = element_blank(), axis.ticks.x = element_blank(), le
p
```

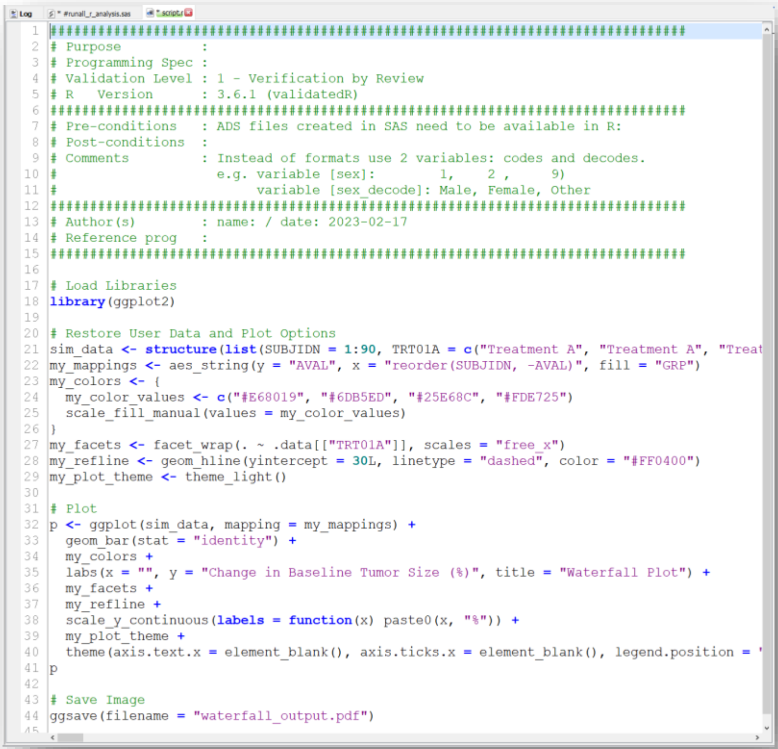
Source code for your Waterfall Plot
Code

```
 1 ################################################################################
 2 # Purpose           :
 3 # Programming Spec  :
 4 # Validation Level  : 1 - Verification by Review
 5 # R   Version       : 3.6.1 (validatedR)
 6 ################################################################################
 7 # Pre-conditions    : ADS files created in SAS need to be available in R:
 8 # Post-conditions   :
 9 # Comments          : Instead of formats use 2 variables: codes and decodes.
10 #                     e.g. variable [sex]:          1,    2 ,     9)
11 #                          variable [sex_decode]: Male, Female, Other
12 ################################################################################
13 # Author(s)         : name: / date: 2023-02-17
14 # Reference prog    :
15 ################################################################################
16
17 # Load Libraries
18 library(ggplot2)
19
20 # Restore User Data and Plot Options
21 sim_data <- structure(list(SUBJIDN = 1:90, TRT01A = c("Treatment A", "Treatment A", "Treatment A",
22 my_mappings <- aes_string(y = "AVAL", x = "reorder(SUBJIDN, -AVAL)", fill = "GRP")
23 my_colors <- {
24   my_color_values <- c("#E68019", "#6DB5ED", "#25E68C", "#FDE725")
25   scale_fill_manual(values = my_color_values)
26 }
27 my_facets <- facet_wrap(. ~ .data[["TRT01A"]], scales = "free_x")
28 my_refline <- geom_hline(yintercept = 30L, linetype = "dashed", color = "#FF0400")
29 my_plot_theme <- theme_light()
30
31 # Plot
32 p <- ggplot(sim_data, mapping = my_mappings) +
33   geom_bar(stat = "identity") +
34   my_colors +
35   labs(x = "", y = "Change in Baseline Tumor Size (%)", title = "Waterfall Plot") +
36   my_facets +
37   my_refline +
38   scale_y_continuous(labels = function(x) paste0(x, "%")) +
39   my_plot_theme +
40   theme(axis.text.x = element_blank(), axis.ticks.x = element_blank(), legend.position = "bottom")
41 p
42
43 # Save Image
44 ggsave(filename = "waterfall_output.pdf")
```

5

PROTOTYPE: TEST

The next part of the workflow is to transfer and execute the generated R script in the production environment. The corresponding output is produced and displayed.



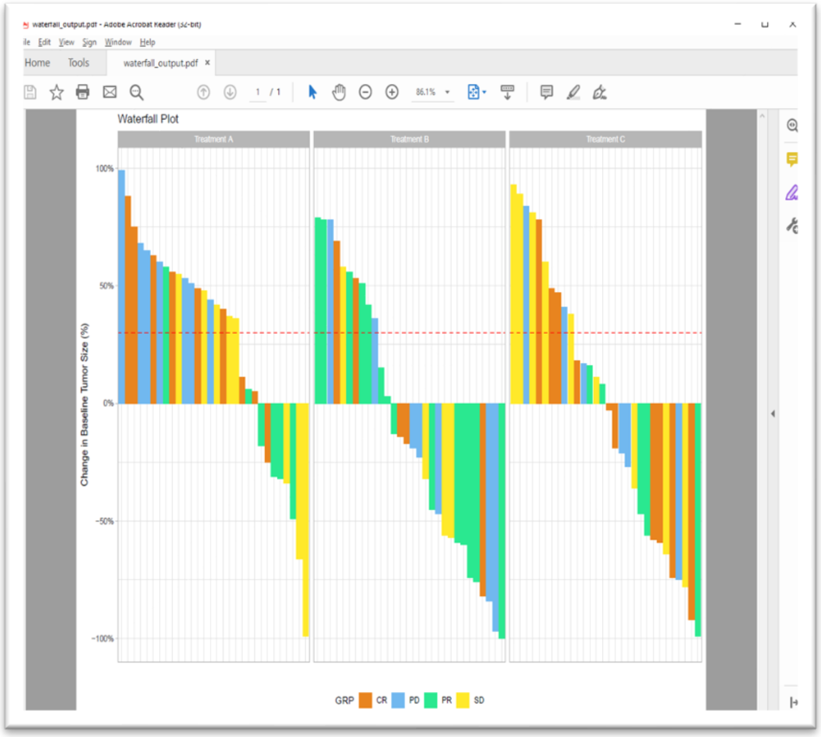
```r
################################################################################
# Purpose           :
# Programming Spec :
# Validation Level : 1 - Verification by Review
# R   Version       : 3.6.1 (validatedR)
################################################################################
# Pre-conditions    : ADS files created in SAS need to be available in R:
# Post-conditions   :
# Comments          : Instead of formats use 2 variables: codes and decodes.
#                     e.g. variable [sex]:         1,     2 ,      9)
#                          variable [sex_decode]: Male, Female, Other
################################################################################
# Author(s)         : name: / date: 2023-02-17
# Reference prog    :
################################################################################

# Load Libraries
library(ggplot2)

# Restore User Data and Plot Options
sim_data <- structure(list(SUBJIDN = 1:90, TRT01A = c("Treatment A", "Treatment A", "Treat
my_mappings <- aes_string(y = "AVAL", x = "reorder(SUBJIDN, -AVAL)", fill = "GRP")
my_colors <- {
  my_color_values <- c("#E68019", "#6DB5ED", "#25E68C", "#FDE725")
  scale_fill_manual(values = my_color_values)
}
my_facets <- facet_wrap(. ~ .data[["TRT01A"]], scales = "free_x")
my_refline <- geom_hline(yintercept = 30L, linetype = "dashed", color = "#FF0400")
my_plot_theme <- theme_light()

# Plot
p <- ggplot(sim_data, mapping = my_mappings) +
  geom_bar(stat = "identity") +
  my_colors +
  labs(x = "", y = "Change in Baseline Tumor Size (%)", title = "Waterfall Plot") +
  my_facets +
  my_refline +
  scale_y_continuous(labels = function(x) paste0(x, "%")) +
  my_plot_theme +
  theme(axis.text.x = element_blank(), axis.ticks.x = element_blank(), legend.position = '
p

# Save Image
ggsave(filename = "waterfall_output.pdf")
```

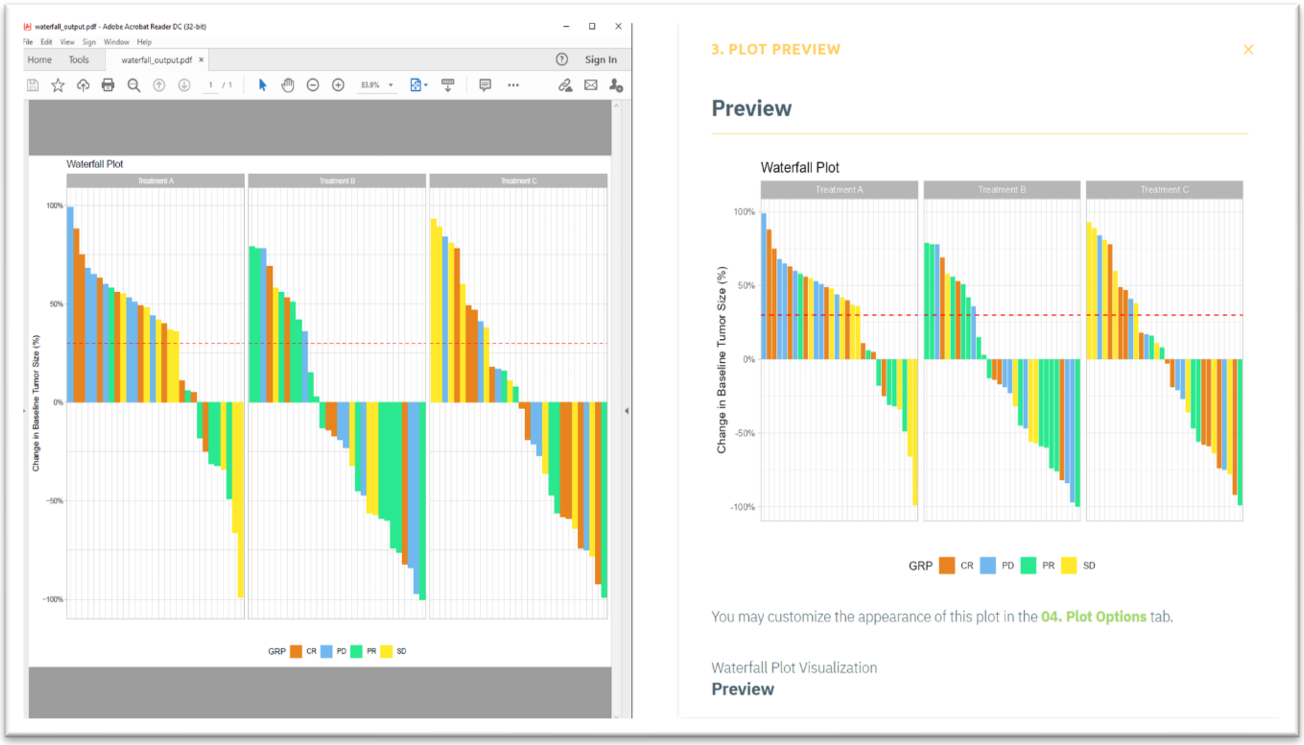


DISCUSSION

In this section of the paper, we consider the outcome of the current case study and discuss more broadly Shiny and reproducibility from different perspectives.

OUTCOME

The outcome of the case study demonstrates the *feasibility* of reproducing a result created in a Shiny app in an external environment using code. In the figure below, the left panel corresponds to the waterfall plot created in the production environment, whereas the right panel displays what was initially designed in the Shiny app.


waterfall_output.pdf - Adobe Acrobat Reader DC (32-bit)
File Edit View Sign Window Help
Home Tools waterfall_output.pdf
Sign In
Waterfall Plot
Treatment A
Treatment B
Treatment C
Change in Baseline Tumor Size (%)
100%
50%
0%
-50%
-100%
GRP CR PD PR SD
3. PLOT PREVIEW
Preview
Waterfall Plot
Treatment A
Treatment B
Treatment C
Change in Baseline Tumor Size (%)
100%
50%
0%
-50%
-100%
GRP CR PD PR SD
Waterfall Plot Visualization
Preview

You may customize the appearance of this plot in the **04. Plot Options** tab.

While visual inspection might be sufficient for verifying graphical output, more formal approaches such as visual regression testing can also be investigated in the future (See *vdiffr*[11] package). Use cases involving complex statistical calculations may warrant greater attention to environment factors and rigorous evaluation of outputs. In the present case study, we made every effort to recreate the destination (i.e. production) environment with respect to the version of R and R packages used. However, in other situations, system-level dependencies (such as the operating system or external software libraries) may need to be considered. One possible approach to tackle this issue is containerization, which could enable a more precise specification of environments and even future reuse.

GENERAL CONSIDERATIONS AND APPLICATIONS

In the current prototype, code generation played a crucial role in replicating a plot developed in a Shiny app elsewhere. However, code generation can also have smaller yet valuable applications. For example, imagine a safety review dashboard where users can perform data exploration by applying complex filters to identify interesting subsets of patients or patterns of events. Code generation could help capture the underlying logic used to identify such subsets, thus enabling their replication in another environment where they can be used in advanced analytics or machine learning.

A key practical consideration when considering code generation and reproducibility functionality is whether it will be integrated into an existing app or developed as part of a new app. While the *shinymeta* interface allows the back-porting of an existing app to enable code generation, this may still require a significant amount of time and effort, depending on the app's size and complexity. Therefore, it's better to discuss the need for code-generation capabilities during the requirements gathering phase of a new app's development to determine whether it is ultimately necessary.

It's also important to note that in some circumstances, the production environment can firmly limit *what is possible* through package and package version availability. Careful thought should then be given to how to construct the core code, since typically more freedom will exist in development environments. For example, accessory and helper packages used in plotting might not be at your disposal in production as was the case in this exercise. Similarly, the *version* of an available package used in production might not have a specific functionality or behaves differently from recent versions.

During the development and piloting of the prototype, the potential of code generation as a tool to introduce or enhance the skills of statistical programmers new to R became apparent. The case study of the waterfall plot offered a digestible example of how to create a relevant deliverable in R. As users tweak the waterfall plot interactively, they receive real time feedback on not only the visual output, but also how those changes impact and integrate in the underlying code. Thus, only a handful of prescribed customizations were offered in the final prototype, and careful consideration was given to the data simulation component. The intent of the prototype was not to be exhaustive, but perhaps promote further learning by providing a tractable, reproducible starting point programmers could replicate as and extend further. For example, adding additional graphical customizations and adapting the code to use real study data.

CONCLUSION

Reproducibility is a fundamental cornerstone of our industry. As the pharmaceutical data science landscape expands to adopt open-source technologies (such as Shiny), it becomes even more critical to ensure that the insights and decisions generated by these technologies are reproducible and trustworthy. In this case study, we demonstrated the feasibility of reproducing a result from a Shiny app in an external environment. This is particularly important in the current data science evolution of clinical trial analysis, where statistical analysts are broadening their skill set to incorporate different tools and approaches for analysis, visualization, and reporting purposes. By serving as a catalyst for the transition and encouraging collaboration among multiple stakeholders, Shiny can help realize potential efficiency benefits in current programming or review processes.

REFERENCES

[1] Sascha Ahrweiler (2014). Managing the Change: Evolving from Statistical Programmers to Clinical Data Scientists. PhUSE EU CONNECT 2014. URL https://phuse.s3.eu-central-1.amazonaws.com/Archive/2014/Connect/EU/London/PAP_PD08.pdf

[2] R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/

[3] Winston Chang, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert and Barbara Borges (2022). shiny: Web Application Framework for R. R package version 1.7.4. https://CRAN.R-project.org/package=shiny

[4] R Consortium (2021). Successful R-based Test Package Submitted to FDA. URL https://www.r-consortium.org/blog/2021/12/08/successful-r-based-test-package-submitted-to-fda

[5] R Consortium (2022). UPDATE: Successful R-based Package Submission with Shiny Component to FDA. URL https://www.r-consortium.org/blog/2022/12/07/update-successful-r-based-package-submission-with-shiny-component-to-fda

[6] H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.

[7] Antonio Nieto, Javier Gómez (2010). Waterfall plots: A beautiful and easy way of showing a whole picture of an interesting outcome. PhUSE EU CONNECT 2010. URL https://phuse.s3.eu-central-1.amazonaws.com/Archive/2010/Connect/EU/Berlin/PAP_SP08.pdf

[8] Kevin Ushey (2022). renv: Project Environments. R package version 0.16.0. https://CRAN.R-project.org/package=renv

[9] Joe Cheng and Carson Sievert (2021). shinymeta: Export Domain Logic from Shiny using Meta-Programming. R package version 0.2.0.3. https://CRAN.R-project.org/package=shinymeta

[10] Juan Pablo Marin Diaz (2020). shinypanels: Shiny Layout with Collapsible Panels. R package version 0.5.0. http://github.com/datasketch/shinypanels

[11] Lionel Henry, Thomas Lin Pedersen, T Jake Luciani, Matthieu Decorde and Vaudor Lise (2023). vdiffr: Visual Regression Testing and Graphical Diffing. R package version 1.0.5. https://CRAN.R-project.org/package=vdiffr

ACKNOWLEDGMENTS

The authors would like to extend a warm thank you to Siqi Zhang, Kangjie Zhang and Estella Dong for their diligent efforts in the conception and development of the prototype. A warm thank you to all statistical programming and medical statistics colleagues who volunteered to pilot the prototype and provide feedback.

CONTACT INFORMATION

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

Matthew Kumar
Bayer Inc
Medical & Scientific Affairs, Pharmaceuticals
Oncology Data Analytics (SBU)
2920 Matheson Blvd E
Mississauga, ON L4W 5J4
Canada
Matthew.Kumar@bayer.com