Create laboratory business intelligence dashboards for free using R: A tutorial using the flexdashboard package

Shannon Haymond

Department of Pathology, Northwestern University Feinberg School of Medicine, Chicago, IL, United States

ARTICLE INFO

Keywords: Data analytics Business intelligence

ABSTRACT

This tutorial described use of the R flexdashboard package and its possibilities for customization to develop a dashboard to monitor critical result reporting in a clinical laboratory. Several examples of interactive components were demonstrated. Sample code and exercises were provided to facilitate learning.

Introduction

Background

Data analytics are becoming an increasingly important part of daily operations and quality improvement efforts within clinical laboratories. Business intelligence dashboards are one of the key components of data analytics since they provide decision makers timely access to summarized analyses and visualizations. Dashboards typically display the status of key performance indicators (KPIs) and other metrics or summary statistics on a single screen, providing information for specific objectives at a glance. While there are several commercially available options for business intelligence software, ranging from more general purpose (e.g., PowerBI, Tableau) to those designed for common laboratory applications (e.g., hc1, Visiun), free alternatives also exist. The R programming language is one of the open-source, freely available options. Web-based dashboards can be created in R using the flexdashboard package. This means laboratorians with R skills may create their own, custom business intelligence tools by extending their abilities to include use of the flexdashboard package. Dashboards made with R code are reproducible, transparent, and may be automated. This tutorial demonstrates the creation of an R flexdashboard for monitoring critical result reporting metrics and workflow attributes.

R flexdashboard

The flexdashboard package extends the customization and functionality of R’s main framework for authoring documents, R Markdown [1]. Flexdashboard enables users to group analyses and visualizations to create static (i.e., a standard web page) or dynamic (i.e., responsive web-based documents using R Shiny) dashboards and storyboards. Many features common to contemporary web documents are also available with flexdashboards. These include: graphs and plots, tables, value boxes, gauges, text annotations, images or icons, and interactive JavaScript-based data visualizations and tables. Flexdashboard layouts are highly customizable and can be configured to automatically adapt for optimal display on differently sized screens (i.e., mobile devices). The documentation site for the flexdashboard package shows several example layouts and provides the underlying code, which can be copied and modified as desired [2].

Clinical application

Clinical laboratories designate certain test results as ‘critical’, typically defined as potentially life-threatening if not acted on promptly by a qualified healthcare provider. Although there is variation across institutions as to (i) which test results are deemed critical, (ii) to whom they are reported, and (iii) in what time frame, institutions share the common patient safety goal of timely communication of results to providers. Regulatory standards from The Joint Commission, Clinical Laboratory Improvement Amendments (CLIA), the International Organization for Standardization, and others require documentation and evaluation of the effectiveness of critical result reporting [3]. As such, critical value reporting compliance metrics are commonly tracked as KPIs in clinical laboratories and hospitals. These metrics may include aggregates of the total number of reported critical values in a specified timespan with a measure of the notification time and compliance with established goals. These data may be further evaluated based on test name, patient population, date-time components, ordering department, or workflow-related attributes (e.g., person reporting, mode of communication), for example.

Objective

After completing this tutorial, a user will be able to:

https://doi.org/10.1016/j.jmsacl.2021.12.002

Received 2 July 2021; Received in revised form 7 December 2021; Accepted 8 December 2021

Available online 13 December 2021

2667-145X/© 2021 THE AUTHOR. Publishing services by ELSEVIER B.V. on behalf of MSACL. This is an open access article under the CC BY-NC-ND license
1. Create a web-based business intelligence dashboard using R flexdashboard.
2. Modify the layout of R flexdashboard.
3. Incorporate interactive tables and graphs into R flexdashboard.

Materials and methods

Requirements

This tutorial assumes the user has access to and basic-level skill working with R [4] and RStudio [5]. The tutorial was developed using the following software versions: R version 4.0.2 (2020-06-22); RStudio Version 1.3.1073; flexdashboard version 0.5.2 [6]; rmarkdown version 2.3 [17], tidyverse version 1.3.0 [8], lubridate version 1.7.9.2 [9], dygraphs version 1.1.16 [10], xts version 0.12.1 [11], plotly version 4.9.2.1 [12], highcharter version 0.8.2 [13], DT version 0.16 [14].

Files

Files needed for this tutorial are available for download from a GitHub repository (https://github.com/SHaymond/Rflexdashboard_tutorial_JMSACL). The files include a mock data set derived from actual clinical laboratory data (july_critical_callback_data.csv), an HTML file of a completed flexdashboard (callback_db.html), and the associated R code to create the flexdashboard (callback_db.Rmd).

Data

The provided dataset includes 2167 critical results reported by a fictional laboratory in the month of July 2020. The data are comprised of 10 columns with information about the sample accession number (accession), resulted test (test_code), order location (pt_type, pt_loc_code), result time (result_datetime), phone time (last_phone_datetime), callback turn-around-time (call_lat), individual phoning (tech, tech_location), and individual receiving the call (phoned_title).

Tutorial

Getting started

The user will follow the steps outlined below to complete the tutorial. Prior to beginning the tutorial, the user is encouraged to open the provided HTML file (callback_db.html) to view the example dashboard. The dashboard includes three pages (Overall, Call Center, Non Call Center), accessed via tabs along the top. Each page shows multiple, interactive visualizations. There are differences across the pages, but the overall look and feel of the dashboard is consistent. Users may also want to keep the associated Rmd file (callback_db.Rmd) open as it is referred to frequently throughout the tutorial and can be used to copy/paste code as needed.

Install the flexdashboard package

The flexdashboard package must be installed prior to creating a flexdashboard, this can be done using install.packages ("flexdashboard") in the console or R code editor window. Once the package is installed, flexdashboard will appear as an output format option in R Markdown (Rmd).

Authoring a flexdashboard

Although flexdashboards are authored using R Markdown, the layout is different than that of R Markdown or R Notebook.

Step 1 – open and inspect a new flexdashboard Rmd file

1. From RStudio, select File ➤ New File ➤ R Markdown…
2. Choose From Template and select the Flex Dashboard template.
3. Click OK.

The very top section is the YAML header, which minimally includes the title for the dashboard and parameters for the output. The YAML is a component of all R Markdown documents that specifies how code and content are processed, displayed, and formatted as output [15]. The output format field value is flexdashboard: flex_dashboard. The rendered document can be customized with additional YAML options and sub-options, which are specified as key:value pairs. The default output options indicate the document will render a flexdashboard with column orientation that fills vertically.

The next section of the template is the setup chunk that minimally contains code to load the flexdashboard package into the local environment. The sections that follow define the layout, sizes, and content for the different areas of the default dashboard template (a single page column layout with room for three charts). Layouts are specified in a hierarchical manner as pages, columns or rows, and then boxes that contain the components. In multi-page formats, page separation is indicated using syntax for level 1 header (### or #). New rows or columns are indicated using syntax for a level 2 header (—or #)#) and the containers are specified using syntax for a level 3 header (###). Content is created from scripts within the code chunks of each container.

As with other R Markdown files, the output file is created by rendering (i.e., knitting).

Step 2 – render a flexdashboard Rmd file

1. Render the dashboard: Go to Knit ➤ Knit to flex_dashboard
2. Save the file into your local directory with a filename of your choice.

This generates the HTML output of the dashboard layout resulting from the default template. The title of the dashboard is shown at the top of the page. The text headers for the page (if used) and container level items are displayed in the rendered dashboard. Header text for the row-or column-level (level 2) items is not displayed in the dashboard. There are three designated components arranged into two columns. The first column contains the Chart A component and the second column has places for Chart B and Chart C components. Notice that the width of the columns is not equal. As the browser window is resized, the vertical height of the content adjusts accordingly so everything is visible on a single page.

Selecting a layout

Because dashboards are typically designed as webpages, authors should think about the layouts in terms of viewing in a web browser. Flexdashboards can be arranged in rows or columns or both and can include tabs or multiple pages. Storyboards are another option for displaying visualizations and commentary, but are beyond the scope of this tutorial. Depending on the layout, authors may choose to have the content fill the page vertically, resizing based on the size of the page, or have content maintain its original height through page scrolling. As noted above, the default orientation is columns with vertical fill. The layout can be easily changed by modifying the orientation and vertical_layout arguments of the flex_dashboard call in the YAML.

Row formats. To change the layout to arrange content in rows versus columns, the orientation argument setting is changed to rows. The available space will be divided equally among the number of visualizations/components grouped within a row (or column), unless the author explicitly specifies the size of the component. This is shown in the default template where the {data-width=} attribute was included and resulted in columns with different widths. Similarly {data-height=} can be used for row orientations. Alternatively, authors may...
define the relative sizes of the row and column containers for a dashboard. Users can experiment with removing or modifying the `data-width=` argument and rendering to see the effect on the dashboard layout.

**Scrolling vertical layout.** For formats that may span more than a single page length, authors can use the scrolling vertical format by setting the `vertical_layout` argument to `scroll`. Unlike the case of `vertical_layout: fill, with vertical_layout: scroll`, the vertical size of the dashboard components does not scale to fill the page when the window is resized. Instead, a scroll bar is added for viewers to move up or down to see content that is outside the page area. Users can test this out by changing the `vertical_layout` parameter, rendering the dashboard, and examining the effect of resizing the browser window.

**Tabset and multiple page layouts.** Sometimes there is too much content for a single page, there are multiple areas of emphasis or focus, or there is a need to toggle between two components in a single dashboard. In these cases, the best layout may be a tabset or multiple page format. Tabsets can be thought of as creating multiple pages within a row or column. To get this effect, the `{tabset}` attribute is included in the section heading to contain the tab (e.g., for a row with a tabset: `## Row (tabset)`) and the level 3 header syntax is used below this to define the number of tabs to appear within the tabset. As noted above, new pages are created with level 1 header syntax. Flexdashboards with multiple pages can have the same or different layouts on each page. Different orientations by page can also be specified using the `vertical_layout` attribute. For this tutorial, a multiple page layout is used to separately visualize the results for the total aggregate and for individual segments based on who is reporting the critical value (i.e., call center or non-call center).

**Setting up the structure for the example.** The critical callback monitor dashboard layout uses three pages, each containing two rows. Row 1 on each page contains three Value Box components. Row 2 on each page contains two items, which vary, depending on the page.

   Step 3 – Specify the dashboard layout and structure

1. Open a new flexdashboard and title it “My Critical Callback Monitor”
2. Configure the dashboard to have rows orientation and the fill vertical layout.
3. Use appropriate syntax to add placeholders and add temporary labels for pages, rows, and components as follows:
   - Page 1: Row 1: Value Box 1, Value Box 2, Value Box 3
   - Page 1, Row 2: Chart 1 and Chart 2
   - Page 2, Row 1: Value Box 1, Value Box 2, Value Box 3
   - Page 2, Row 2: Chart 1 and Table 1
   - Page 3, Row 1: Value Box 1, Value Box 2, Value Box 3
   - Page 3, Row 2: Chart 1 and Table 1
4. Render the dashboard and check that the basic layout matches that in the callback_db.html file.

As the tutorial progresses, users can practice writing code or copy and paste provided code to add the components and customizations to their dashboard.

**Loading the packages and preparing data.**

As with all R scripts, necessary packages and data must be loaded prior to building the dashboard. Note that the visualizations require additional columns to be created from the datetime fields in the data file.

   Step 4 – Insert setup and data code chunks

- Copy/paste (from `r setup` and `r data code chunks in callback_db.Rmd`) or write code to load necessary packages and read and prepare data.

Note: to load, all packages must be previously installed in the user’s R environment.

At this point, users should have a Rmd file that includes the setup and data code chunks and has the correct dashboard layout structure specified. The next steps demonstrate how to build the content for the dashboard.

**Adding components.**

The components of the dashboard are the content that displays within the containers. As mentioned earlier, there are many types of components to choose from, including both static and interactive options. The flexdashboard package documentation [16] and online gallery [2] provide visual examples with code to explore and experiment further. This tutorial will cover some of these components.

**Value boxes.** Dashboards commonly include one or more graphics displaying simple values often with an icon and descriptive title. The `flexdashboard` package includes a `valueBox()` function that creates this type of display. As shown in the `r avg-tat` code chunk of the `call-back_db.Rmd` file, which makes Value Box 1 on Page 1, this function takes arguments for `value`, `icon`, `caption`, and `color`.

**Color**

Value box color can be configured in a static or conditional manner. Colors can be assigned in terms relevant to business intelligence displays, such as “primary” (default color setting), “info”, “success”, “warning”, and “danger”. In these cases, specific colors for a given theme are tied to the terms. Authors can also apply colors by specifying any valid CSS color as name, hexadecimal, or RGB notation.

**Icons**

Authors can include icons from three different icon sets: (Font Awesome [17]; (2) Ionicons [18]; (3) Bootstrap Glyphicons [19].

Icons are specified using their full name, including the icon set prefix (e.g., “fa-stopwatch”, “ion- alarm-outline”, “glyphicon-time”, etc.). Icons can also be added to section headings using the `data-icon=` attribute and supplying the icon name as the argument, as shown in the `callback_db.Rmd` file for the page titles.

1. Modify the page titles and include icons with `data-icon=`, as follows:
   - Page 1: title = Overall; icon = “fa-phone-volume”
   - Page 2: title = Call Center; icon = “fa-users”
   - Page 3: title = Non Call Center; icon = “fa-users”
2. Use `valueBox()` to create Value Box 2 on Page 1, which displays the total number of calls (`calls <- cb_data %>% nrow()`) is greater than or equal to 100.
3. Include the icon “fa-hashtag” and make the color “orange”
4. Title the value box “Total calls” and give the same title to its container header
5. Apply this same code for Value Box 2 on Pages 2 and 3, changing only the input data. Refer to `callback_db.Rmd` file for the completed code to create the remaining value boxes on Pages 1, 2, and 3.
6. Using the same approach in the steps above, create Value Boxes 1 and 3 for all pages.

Note the different ways to assign color and the use of conditional formatting in the `r pct-on-time`, `r pct-on-time-cc`, and `r pct-on-time-non` code chunks of the `callback_db.Rmd` file. The boxes are green if the percent of calls made on time (i.e., within 30 min) is ≥95% and red otherwise.

**R graphics.** Any chart created with standard R graphics can be used in a flexdashboard. The syntax is the same as that used in R scripts or R Markdown files and is specified within the container it will appear. R will scale the graphic to preserve the aspect ratio, but since these are PNG images, they will not automatically fill the bounds of their
containers. Therefore, authors must define the options for `fig.width` and `fig.height` to match the size of the container on the page. Unfortunately, the ideal values are usually determined through experimentation. This is why it may be desirable to render R ggplot2 graphics using the `plotly` htmlwidget as will be described below.

**HTML widgets.** The htmlwidgets framework provides a series of packages that are wrapper libraries to R for JavaScript data visualizations. Charts based on htmlwidgets are well-suited for flexdashboard because they dynamically re-size and are interactive because of the embedded JavaScript. The htmlwidgets website [20] has more information about the available HTML widgets and their R packages and how authors can create new widgets. The following examples are used in this tutorial: dygraphs, plotly, and DT. Authors must install and load the htmlwidget package they wish to use. Each of these packages has its own syntax, but code for htmlwidgets is written within chunks in R Markdown, similarly to what is done for other types of R plots. A review of the available documentation for each htmlwidget package will provide the required syntax and arguments available for customizing output.

The `r plot-daily-volumes` code chunk in the `callback.db.Rmd` file shows the code used to create the daily call volumes graph on Page 1. This was produced using the `dygraphs` widget, which generates nicely formatted, interactive charts for time series data. This plot allows zooming in, highlighting, and tool tips showing the volume for a given date when moused over – all baked into the single `dygraph()` function call. Underlying this function is the JavaScript code to create this plot and make it interactive. A custom color and line width were specified using the `dyOptions()` call. The `dygraphs` widget requires data as a timeseries, which is a special type of R object. The `r plot-daily-volumes` code chunk in the `callback.db.Rmd` file demonstrates a way to convert the provided data to a timeseries.

Step 6 – create the interactive time series chart

- Create the `dygraph` plot in the container for Chart 2 on Page 2. Copy/paste (from `r plot-daily-volumes` code chunk in `callback.db.Rmd`) or write the code to obtain the `daily_vol` timeseries object and pass this to the `dygraph()` function. Make sure to print the plot in your code, if saved as a variable.

Inline code syntax can be used to dynamically update the titles of dashboard components. The title for the `dygraph` plot uses inline code (example shown below) to automatically pull in the start and end dates of the underlying dataset.

```r
### Daily Call Volumes for `r date(min(cb_data$last_phone_datetime))` through `r date(max(cb_data$last_phone_datetime))`
```

A similar approach was used for the other charts and tables in the example dashboard.

Step 7 – make component titles using inline code syntax

- Following the example from the text above, copy/paste or write code to create dynamic titles for the charts and tables using inline code syntax.

As mentioned above, it may be preferable to convert `ggplot` objects into `plotly` objects so they render better on dashboards. This is straightforward to do using the `plotly` package. The `plotly` htmlwidget provides an R interface to the `plotly.js` JavaScript graphing library. Authors write the code for `ggplot` graphics as they normally would and use this as input to `ggplotly()`. As with the `dygraph` example above, the htmlwidgets framework provides the author with a lot of JavaScript functionality with very little additional code and no required familiarity with JavaScript. The `r plot-hourly-by-day-no-weekend` code chunk in the `callback.db.Rmd` file shows the code used in the critical callback example to generate the interactive barplots showing the weekly breakdown of calls by hour and weekday. The `ggplot` bar plot object is created and this is passed to the `ggplotly()` function to yield the interactive version of the tiled barplots.

Step 8 – compare static and interactive plot versions

1. Copy/paste (from `r plot-hourly-by-day-no-weekend` code chunk in `callback.db.Rmd`) or write the code to create the tiled `ggplot` bar plots into the container for Chart 1 on Page 1. Make sure to print the `ggplot` in your code, if saved as a variable.
2. Render your dashboard to view the plot.
3. Modify code to provide the `ggplotly` object as input to the `ggplotly()` function and re-render the dashboard to view the effect.

Step 9 – create the interactive boxplot figures

- Copy/paste (from `r cc-tech` and `r non-tech` code chunks in `callback.db.Rmd`) or write the code to create the boxplot figures on Page 2 and Page 3.

Tables. The last component type covered in this tutorial is tabular data. Tables can be included as simple, static displays or as interactive tables with capabilities for sorting, filtering, searching and pagination. Static tables can be built and styled using the `kable` and `kableExtra` packages. Interactive tables can be created using the `DT` htmlwidget, which is a wrapper for the `datatables` JavaScript library.

Step 10 – Compare static and interactive tables

1. Copy/paste (from `r cc-overdue` code chunk in `callback.db.Rmd`) or write the code to create the table object `cc_late` into the container for Table 1 on Page 2.
2. Format the data into a static table with:

```r
kbl(cc_late) %>% kable_styling()
```

3. Render your dashboard to view the table.
4. Modify the code (as below) to format the data into an interactive table and re-render the dashboard to view the effect:

```r
datatable(cc_late, options = list(pageLength = 20, autoWidth = TRUE))
```

5. Use code above and `r non-overdue` code chunk in `callback.db.Rmd` to make the interactive table on Page 3.

The code to identify the data for the table is the same for static vs interactive tables, but the code to style and render the table differs in the functions used. The tables on Page 2 and Page 3 differ in their input data (i.e., the group making calls).

**Themes**

The aesthetics of flexdashboards are provided by Bootstrap themes. Changing the theme of a flexdashboard allows for additional customization. The critical callback monitor dashboard uses the default theme, cosmo. The `theme` options for flexdashboard are the same ones available for R Markdown [1], which are a set of free themes from the Bootswatch library [21].

The theme is set using the `theme` argument of the `flex_dashboard` call in the YAML output field. Because this tutorial uses the default theme in the sample dashboard, the theme is not specified. To use a new theme, add the `theme` argument and provide the name of a valid theme (i.e., `theme=flatly`). A CSS code file can also be supplied and specified in the `flex_dashboard.css` argument to further customize the look of a flexdashboard. The `bslib` R package [22] allows authors to modify the appearance of dashboards by customizing Bootstrap CSS code directly in R.
**Step 11 – customize flexdashboards with themes**

1. Add an argument for theme to the `flex_dashboard` call in the YAML (at the same level as the orientation argument).
2. Set the argument to one of the available themes.
3. Render the dashboard and compare the style to the cosmo theme (i.e., that in the callback_db.HTML file).

**Flexdashboard HTML files**

The output of a flexdashboard is a standalone, static HTML file that may include interactivity. This has the advantage of being viewed on any web browser without the need for R or a server. The file includes all data and code (i.e., R, CSS, and JavaScript) needed to view the dashboard in a browser. The dashboard is static in that it only includes data available at the time it was rendered, yet it can be interactive if htmlwidgets or other JavaScript features are embedded. Developers commonly combine R Shiny with flexdashboard to make dashboards that are reactive. This means the dashboard dynamically re-renders as a result of a user action.

**Declaration of Competing Interest**

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

**References**

[1] Y. Xie, J.J. Allaire, G. Grolemund, R Markdown: The Definitive Guide [Internet]. Chapman and Hall/CRC, Boca Raton, Florida, 2018. Available from: https://bookdown.org/yihui/markdown/.

[2] Example projects • flexdashboard [Internet]. [cited 2021 Jun 30]. Available from: https://pkgsrc.rstudio.com/flexDashboard/articles/examples.html.

[3] G. Lippi, C. Mattiuzzi, Critical laboratory values communication: summary recommendations from available guidelines, Ann Transl Med. 4 (20) (2016) 400.

[4] R Core Team. R: A Language and Environment for Statistical Computing [Internet]. Vienna, Austria: R Foundation for Statistical Computing; 2020. Available from: https://www.R-project.org/.

[5] RStudio Team. RStudio: Integrated Development Environment for R [Internet]. Boston, MA: RStudio, PBC; 2020. Available from: http://www.rstudio.com/.

[6] R. Iannone, J.J. Allaire, B. Borges, flexdashboard: R Markdown Format for Flexible Dashboards [Internet]. 2020. Available from: https://CRAN.R-project.org/package=flexdashboard.

[7] J.J. Allaire, Y. Xie, J.J. McPherson, J. Larsen, K. Ushey, A. Atkins, et al. rmardown: Dynamic Documents for R [Internet]. 2020. Available from: https://github.com/rstudio/rmarkdown.

[8] H. Wickham, M. Averick, J. Bryan, W. Chang, L. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. Lethen, E. Miller, S. Bartsch, K. Müller, J. Ooms, D. Robinson, D. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, H. Yutani, Welcome to the tidyverse, J. Open Sour. Software. 4 (43) (2019) 1686, https://doi.org/10.21105/joss.01686.

[9] G. Grolemund, H. Wickham, R Markdown: The Definitive Guide [Internet]. Chapman and Hall/CRC; 2020. Available from: https://bookdown.org/yihui/rmarkdown.

[10] D. Vanderkam, J.J. Allaire, J. Owen, D. Gromer, B. Thiermeil, dygraphs: Interface to ‘Dygraphs’ Interactive Time Series Charting Library [Internet]. 2018. Available from: https://CRAN.R-project.org/package=dygraphs.

[11] J.A. Ryan, J.M. Ulrich, xtc: eXtensible Time Series [Internet]. 2020. Available from: https://CRAN.R-project.org/package=xtx.

[12] C. Sievert, Interactive Web-Based Data Visualization with R, plotly, and shiny [Internet]. Chapman and Hall/CRC, 2020. Available from: https://plotly-r.com/.

[13] J. Kunst, highcharter: A Wrapper for the “Highcharts” Library [Internet]. 2020. Available from: https://CRAN.R-project.org/package=highcharter.

[14] Y. Xie, J. Cheng, X.D.T. Tan, A Wrapper of the JavaScript Library “DataTables” [Internet]. 2020. Available from: https://CRAN.R-project.org/package=DT.

[15] Y. Xie, C. Dervieux, E. Riederer, R markdown cookbook, 1st ed., Taylor and Francis, CRC Press, Boca Raton, 2020.

[16] R Markdown Format for Flexible Dashboards • flexdashboard [Internet]. [cited 2021 Jun 30]. Available from: https://pkgsrc.rstudio.com/flexDashboard/index.html.

[17] Icons | Font Awesome [Internet]. [cited 2021 Jun 30]. Available from: https://fontawesome.com/v5.15/icons?d=gallery&p=2.

[18] Ionic: The premium icon pack for Ionic Framework [Internet]. [cited 2021 Jun 30]. Available from: https://ionic.io/ionicicons.

[19] Components - Bootstrap [Internet]. [cited 2021 Jun 30]. Available from: https://getbootstrap.com/docs/3.3/components/.

[20] htmlwidgets for R [Internet]. [cited 2021 Jun 30]. Available from: https://www.htmlwidgets.org/.

[21] Bootswatch: Free themes for Bootstrap [Internet]. [cited 2021 Jun 30]. Available from: https://bootswatch.com/.

[22] C. Sievert, J. Cheng, bslib: Custom ‘Bootstrap’ ‘Sans’ Themes for ‘shiny’ and ‘rmarkdown’ [Internet]. 2021. Available from: https://CRAN.R-project.org/package=bslib.