PharmaSUG China 2023 - Paper AA-139

Practice On Using Card-Based Design to Develop Dashboard Via R Shiny

Zhiwei Luo, Pfizer (China) Research and Development Co., Ltd. China;

ABSTRACT

Nowadays, card-based UI design style has adopted by many IT companies like Apple, Microsoft, and Google given its clean and minimalist appearance. This type of UI design utilizes small, self-contained units of content or functionality organized into cards, making it easier for users to navigate and interact with the interface, which provides seamless and intuitive user experiences.

Many pharmaceutical companies are currently using R Shiny to analyze and explore clinical trial data. With the increasing popularity of card-based design language, multiple packages such as {blisb}, {summarybox}, {shinydashboard}, {shinywidgets}, and {shinydashboardPlus} have been developed to enhance traditional R Shiny UI. These packages enable users to design beautiful R Shiny card-based apps without front-end knowledge like CSS or JavaScript.

However, with so many options available, it can be confusing to decide which package to use for layout design, theme adjusting, or card type presentation. This article aims to simplify the workflow of building modern design R Shiny Apps by utilizing these packages based on practical experience.

INTRODUCTION

In daily practice, I often divide the development process of a card-based design dashboard into five steps, as shown in the following
diagram:

Figure 1. General workflow of card-based design dashboard

The specific content outline of the five development steps is as follows:

- **Data Source Management:** Before exploring the data, we first need to understand it deeply, starting from the requirements and identifying the subject of data analysis. This subject can be a set of laboratory data, a project progress, etc. Then, we explore which data are needed to meet the analysis requirements of the subject and form an application driven by the analysis subject.
- **Divide Contents Into Units & Assign Cards:** We classify and divide the required analysis data into individual analysis units. Then, we analyze each analysis unit to explore which data display method is suitable for presentation, such as various visualization graphics, statistical tables, cards for diverse interaction, or simultaneous display of graphics and data. Depending on the requirements.
- **Organization:** Organization means we need to find a proper way to organize those cards, we can start with hand-drawing, classifying, organizing, and adjusting the layout of the card content to make the entire page logical and orderly. Then, we select an appropriate layout method from Shiny.
- **Decorate:** After completing these three steps, the majority of the work for the entire app has been completed. At this stage, what we need to do is to decorate the app. During the decoration process, we will be dealing with ICONs, themes, and so on.
- **Deploy:** After creating the application locally, we need to consider the deployment issue. Depending on the requirements, we can choose Docker or cloud deployment, etc.

This paper will primarily focus on introducing the middle three steps in detail.

DIVIDE CONTENTS INTO UNITS & ASSIGN CARDS

CARDS TYPE SUMMARY

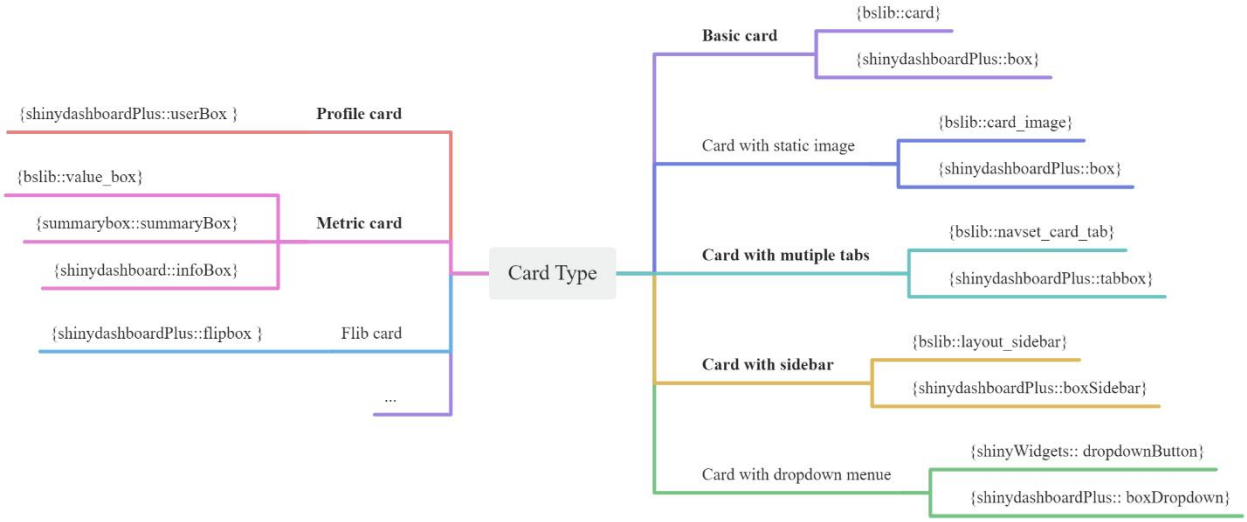

Figure 2. Card Type Summary

Different analysis requirements require breaking down data into different analysis units. This needs to be discussed collectively before designing the app. However, the presentation format of the analysis results is universal, which mainly includes various types of dynamic charts and tables. This section will introduce several card types used to display these analysis results. The above figure summarizes the commonly used card types and indicates which functions of which packages can create them. Due to the limitations of space, the following will introduce the key card types highlighted in bold in the above figure. Let's explore them.

BASIC CARD

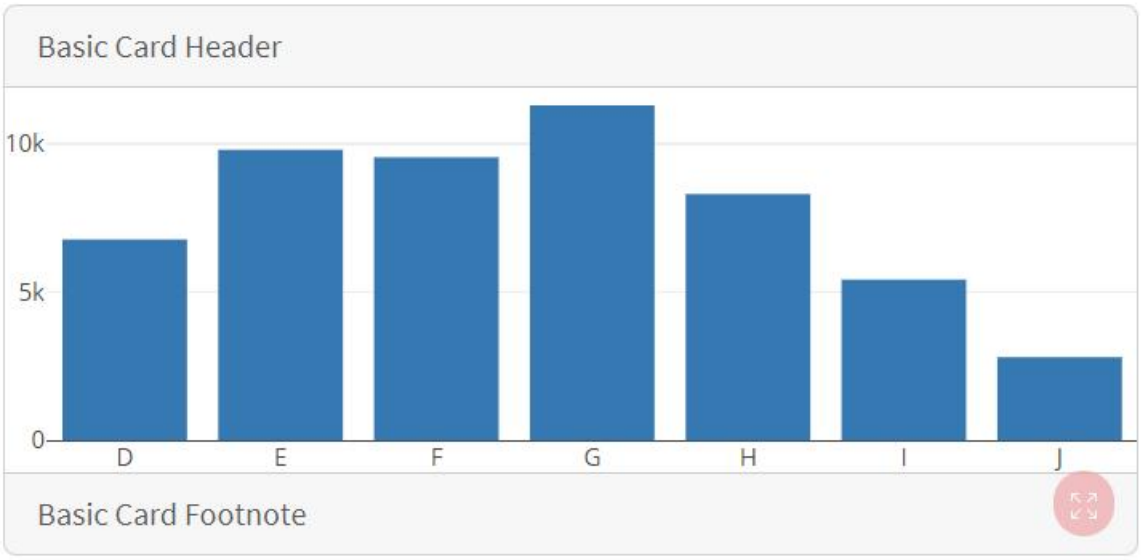

Figure 3. Basic Card

The basic card type includes the title of the card, the content it holds, the footnote of the card and some basic properties such as color and the size and color of the card text. Basic card can create from multiple packages.

CARD WITH MUTIPLE TABS

When we need to see data trends at a glance and then examine detailed data, we can choose this type of card that combines data and charts, a card with multiple tabs can let us explore a specific data from multiple angles. Below example created by **{bslib::navset_card_bar}.**

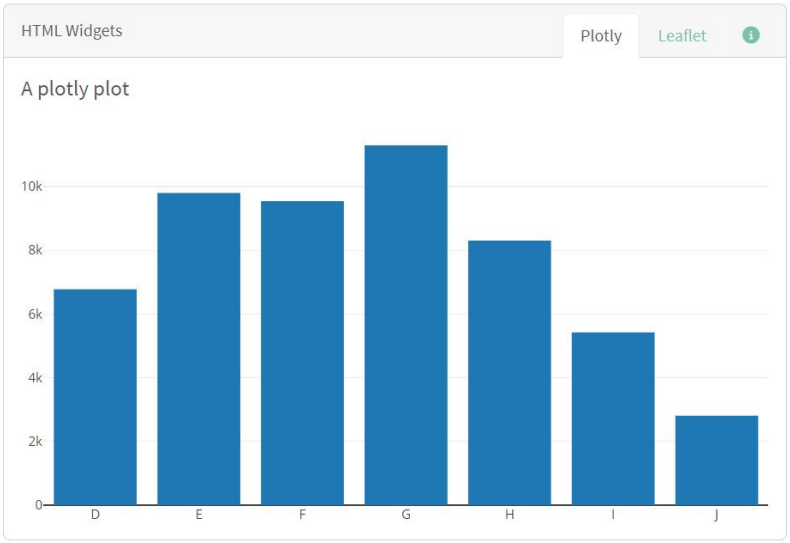

Figure 4. Card with multiple tabs

CARD WITH SIDEBAR

With the sidebar, we can do more things on card's contents, making it more interactive, Below sidebar card was created by **{bslib::layout_sidebar}.**

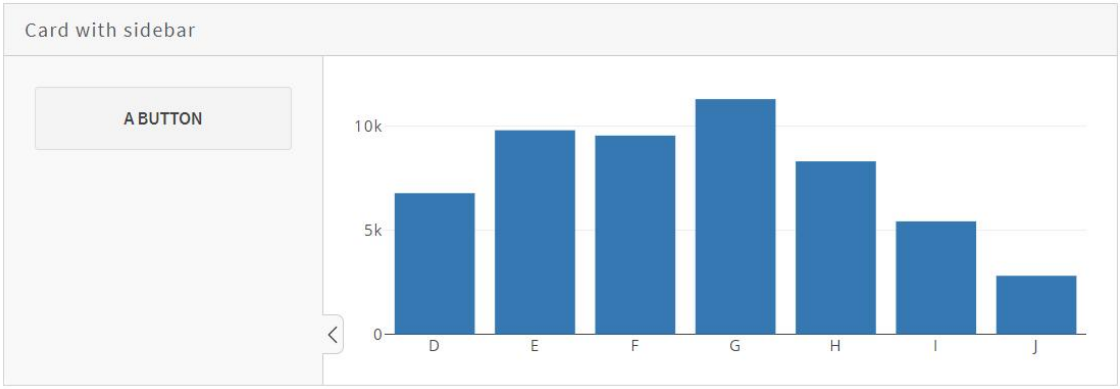

Figure 5. Card with Sidebar

USERINFO CARD

When our analysis object is a person, we often need to list the personal information,the user info box from **{shinydashboardplus::userbox}** do a great job. This card type is very flexible and we can infinitely add multiple pieces of information.

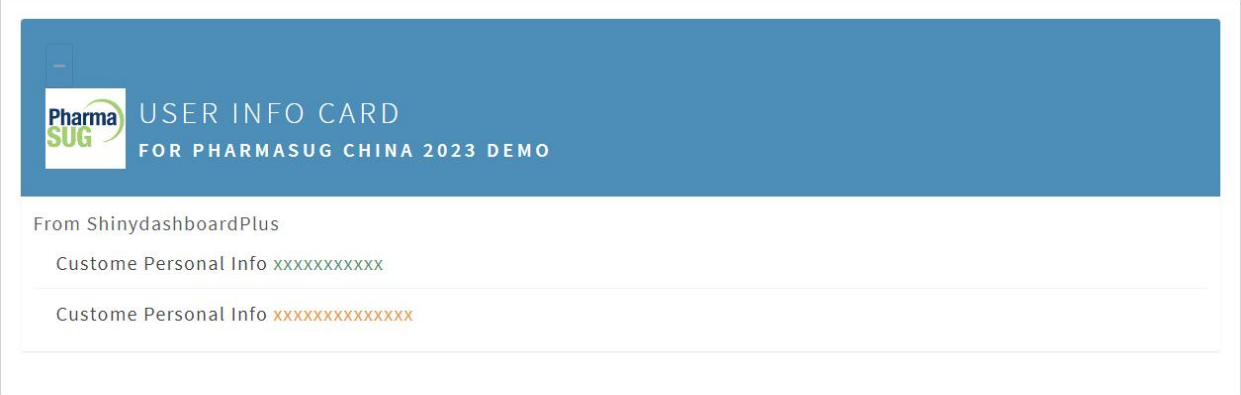

Figure 6. User Info Card

So above is a rough introduction to card types. When we think about what card types to use to hold our analysis content, we first consider the types provided by **{bslib},** If **{bslib}** cannot meet our needs, we will then look for them in the enhancement packages such as **{shinydashboardPlus}** and **{shinywidget}** and so on. These cards should be sufficient for our use.

As another important card design for card-style applications, the value box is an excellent way to display metrics. Value boxes come in various styles, depending on different packages. Due to space limitations, this paper will introduce 3 different styles of value boxes from different packages.

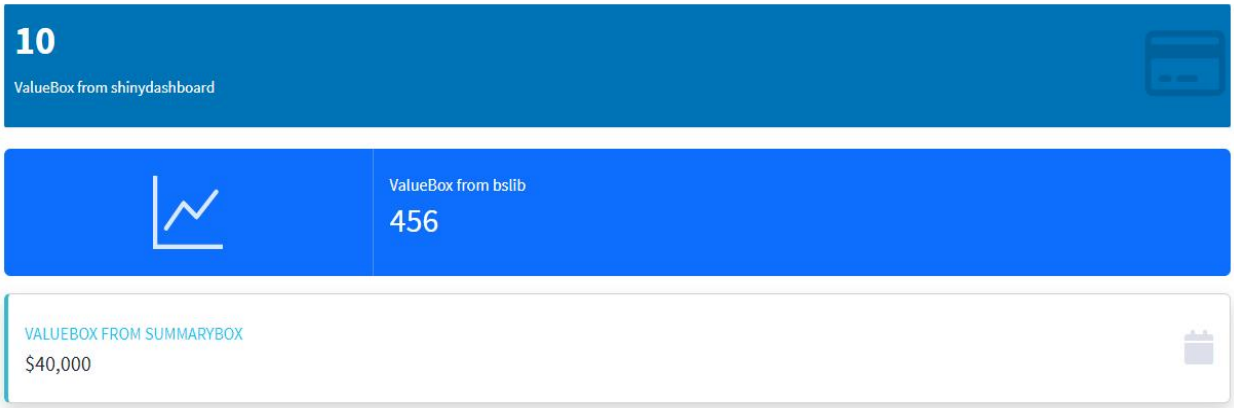

Figure 7. Value Box From Different Packages

Above 3 value boxes come from **{shinydashboard}, {bslib}, {summarybox}** separately, they also have different style inside package, If needed, you can delve deeper into them.

ORGANIZATION

When it comes to layout, I strongly recommend using Shiny's native layout functions. Although other packages also have corresponding page layout functions, Shiny's functions are more stable and work better with other packages' widgets. Layout functions from other packages may have some compatibility issues and sometimes fail with updates.

The layout for a card-style app is relatively simple, consisting of two parts: the overall layout of the entire page and the arrangement of individual cards. We will discuss these two aspects in detail.

PAGE LAYOUT

The core concept of card-based design is flat, simple and smooth, so a clean full-screen layout with tabs is a great choice. This combination will have below advantages:

- Clear and intuitive: Each card is independent, clearly displaying the content of each analysis unit, making it easier for users to understand.
- Easy to navigate: The full-screen layout with tabs design allows users to quickly switch between different analysis units, improving the user experience.
- Flexible and scalable: The card design can easily add or remove cards to adapt to constantly changing needs.
- Easy to maintain: The card design can separate each analysis unit, making it easier to maintain and update.
- Aesthetically pleasing: The card design can make the page look neater and more visually appealing, improving the user's visual experience.

In summary, the full-screen layout with tabs and card design is a very suitable design style for data analysis and visualization scenarios, allowing users to focus more on analyzing each analysis unit, making the entire app well-organized.

Therefore, this paper will focus on this layout. In shiny this layout created from **navbarPage** function.

Below are simple code example and output.

```
library(shiny)
# ui
ui=navbarPage(
  title = "PharmaSUG China 2023 Demo ",
           tabPanel(title = "Home",
                    "content 1"),
           tabPanel(title = "SDTM Explore",
                    "content 2"),
           tabPanel(title = "ADaM Explore",
                    "content 3")
         )

# server
server=function(input, output) {}

shinyApp(ui = ui, server = server)
```

Figure 8. NavbarPage Layout from native Shiny

CARDS LAYOUT

In terms of card-based design layout,the organization of cards is always from top to bottom, with multiple cards arranged in a row. Therefore, our focus point is on arranging each row. For the arrangement of rows, we can use function **fluidrow** from **{shiny}** and layout_column_wrap from **{bslib}** to handle it, and **layout_column_wrap** function works well with cards from **{bslib}, {shiny::fluidrow}** are much more compatible with cards from different packages.

FLUIDROW

The FluidRow function enables you to create a row in a Shiny application and arrange multiple card elements within that row. It is a component of the Fluid page layout system, which utilizes the Bootstrap 12-column grid system. This means that when using FluidRow to divide a row into multiple columns, the sum of the column widths must not exceed 12. Let's take a simple example,which divide a row into 2 column.

```
ui <- fluidPage(

  fluidRow(
    column(6,
    card1
    ),
    column(6,
    card2
    )
  )
)
```

Figure 9. A sample example using fluidrow divide a row into 2 equal column

LAYOUT_COLUMN_WRAP

The function layout_column_wrap which comes from {bslib::layout_column_wrap}. The advantages of this function beyond {shiny::fluidrow} is that it can automatically divide a row evenly using simple parameters and do nested layout easily. Like the code below, simply assign "1/3" to the parameter width, we can arrange 3 cards in a row evenly. Additionally, we can also arrange the cards with fixed width.

```
#Divede a row into 3 column evenly

page_fluid(layout_column_wrap(
  width = 1 / 3,
  card(full_screen = F, card_header("Card 1"), card1),
  card(full_screen = F,card_header("Card 2"), card2),
  card(full_screen = F, card_header("Card 3"),card3)
```

```
))
```

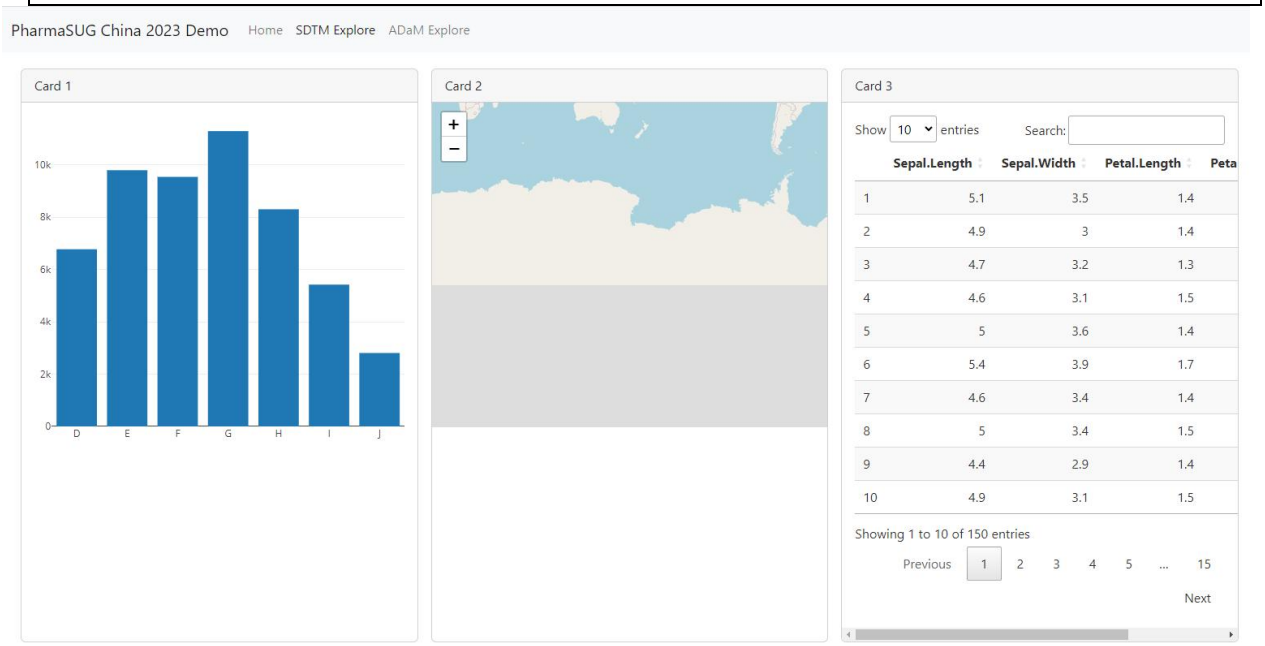

Figure 10. Dividing a row evenly using layout_column_wrap

Nested layout means that we can use the layout_column_wrap function inside the layout_column_wrap function, making the layout more flexible.

```
#Nested Layouut

layout_column_wrap(
  width = 1/2,
  height = 300,
  card1,layout_column_wrap(
    width = 1,
    heights_equal = "row",
    card2, card3
  )
)
```

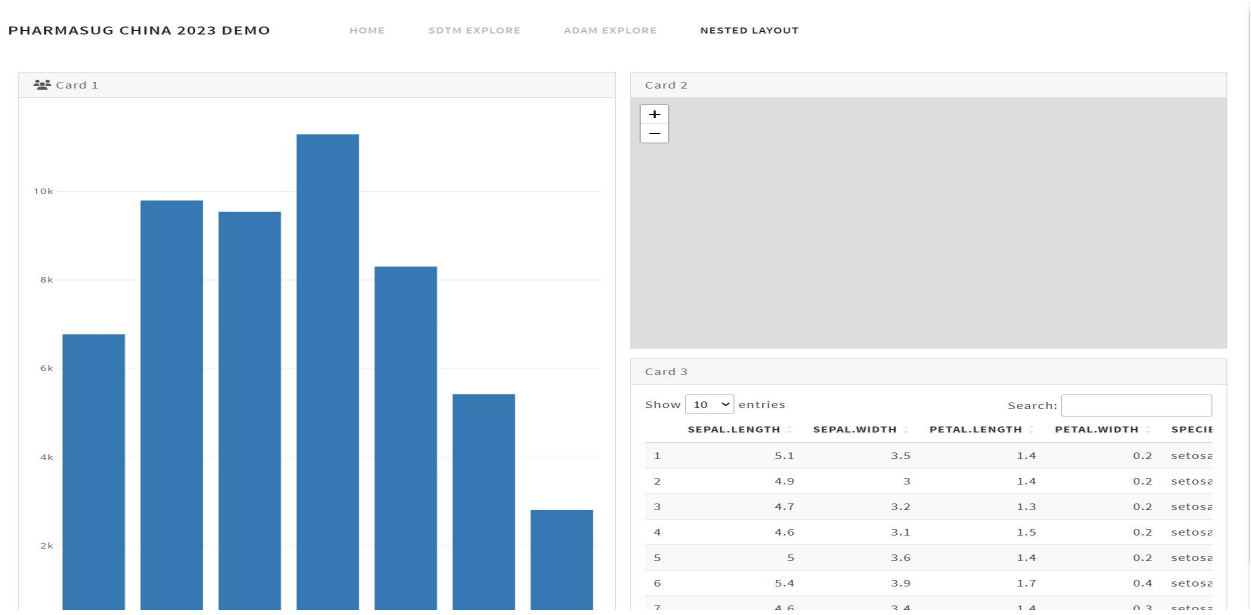

Figure 11. Nested layout using layout_column_wrap

DECORATE

Up until now, most of the work on our app has been completed, and all that remains is to decorate and beautify the app. The paper introduce two parts of content that can beautify the app.

THEME

For an app, the theme is equivalent to clothes for a person. Native Shiny currently uses a Bootstrap 3-style theme, but Bootstrap has now evolved to its fifth generation, and bootswatch is a free collection of bootstrap theme. With the help of **{bslib},** we can easily switch bootstrap version and the bootswatch theme, making it remarkably simple to alter the appearance of your app.

```
# Bootswatch theme flatly

theme <- bs_theme(bootswatch="lux")

# UI
ui <- fluidPage(

  theme = theme,


# Server
server <- function(input, output, session) {


}

# Run App
shinyApp(ui = ui, server = server)
```

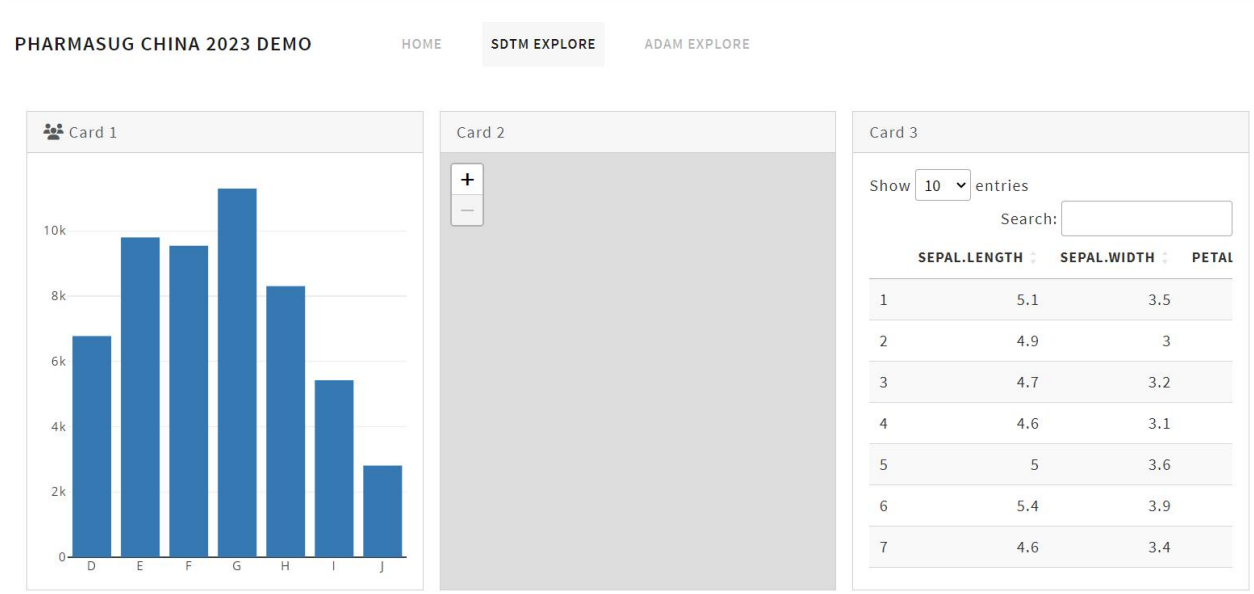

Figure 12. Shiny App with bootswatch theme lux

In the code above, we can see that we only need to select a theme from Bootswatch and then pass the theme name to the bootswatch parameter of the bs_theme() function. This way, a beautiful theme will be applied to your app. However, **{bslib}** can do more than just that. The **{bslib::bs_theme}** function has multiple parameters that allow you to customize the appearance of your app. You can delve deeper into it by reading the **{bslib}** documentation.

```
bs_theme(
  version = version_default(),
  preset = NULL,
  ...,
  bg = NULL,
  fg = NULL,
  primary = NULL,
  secondary = NULL,
  success = NULL,
  info = NULL,
  warning = NULL,
  danger = NULL,
  base_font = NULL,
  code_font = NULL,
  heading_font = NULL,
  font_scale = NULL,
  bootswatch = NULL
)
```

ICON

In the title of a card, a suitable ICON can make the card more lively. We only need to use the icon parameter to add the name of the icon. The icon name we can find from the website Fontawesome.

```
card( full_screen = F,
      card_header(span(icon("users"),"Card 1")),
      plotly_widget)
```

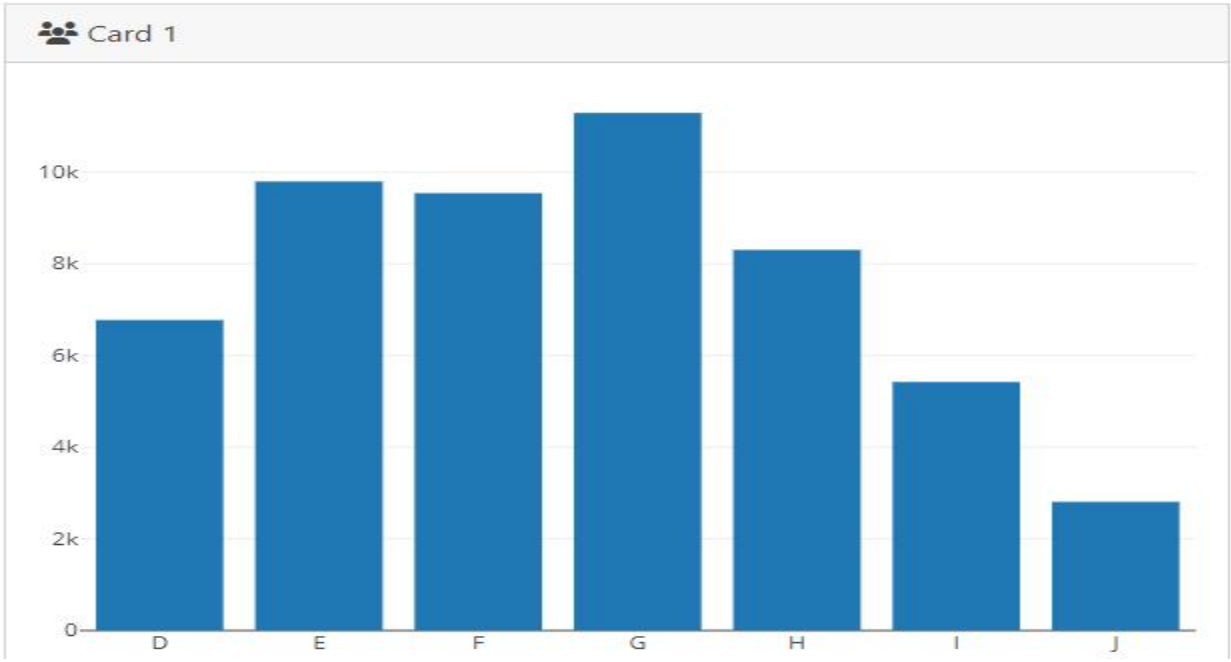

Figure 13. A card title with Icon

CONCLUSION

This paper provides a simple workflow for creating card-based apps, covering topics from selecting card types to layout to decoration with suggested solution.

At the page level, we should consider using **{Shiny}** native layout functions, but when it comes to card layouts, we can use the layout functions from **{bslib}.** When it comes to card types, we should first see if the cards provided by **{bslib}** meet our needs. If not, we can search from other packages like **{shinydashboardPlus}** and **{shinyWidgets},** etc. Finally, when beautifying the dashboard, we should first use **{bslib}** to choose the corresponding theme from bootswatch. If we are not satisfied, we can customize the theme by define parameters in **{bslib::bs_theme}.**

So far, with the help of the aforementioned packages, we can create beautiful and professional dashboards without any CSS or JavaScript knowledge. However, by mastering CSS and JavaScript, we can take it a step further, this is an advanced topic.

RECOMMENDED READING

- *Outstanding User Interfaces with Shiny https://debruine.github.io/shinyintro/index.html*
- *Building Web Apps with R Shiny https://unleash-shiny.rinterface.com/index.html*

CONTACT INFORMATION

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

Zhiwei Luo
Pfizer (China) Research and Development Co., Ltd. China
17607086677
luozhiwei@live.cn