PharmaSUG China 2022 - Paper DV-130

Use R Shiny to Develop a User-Friendly Interactive Tool

Chengcheng Ma, Legend

ABSTRACT

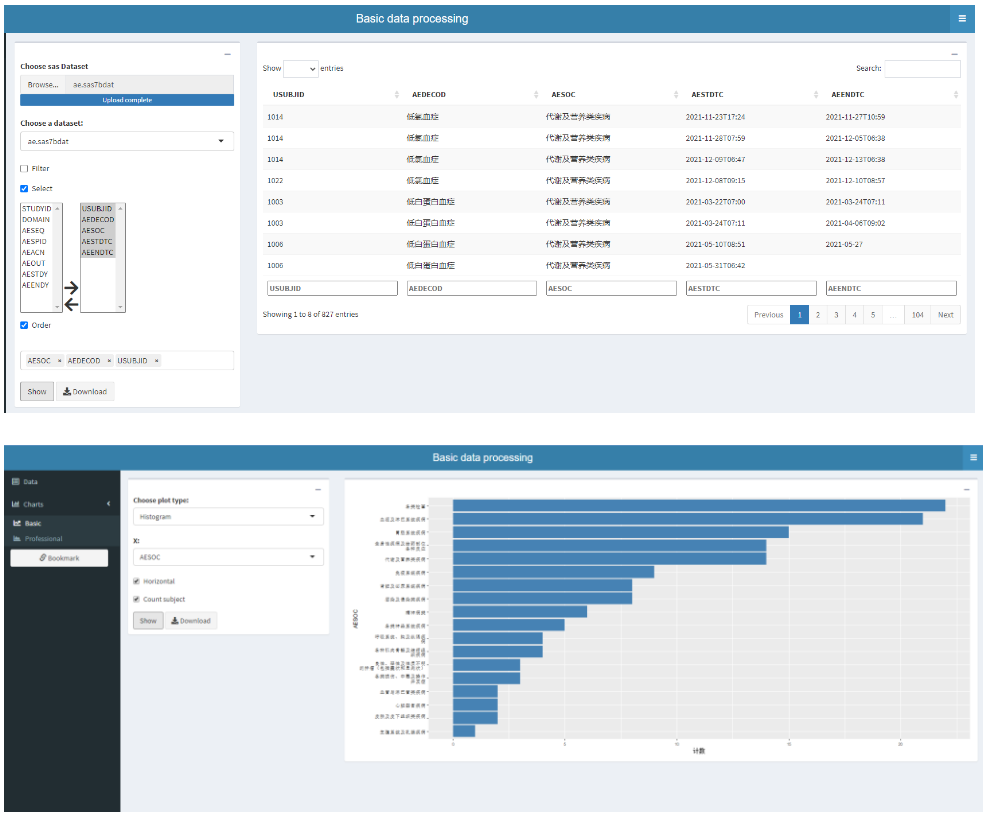
Over the last decade, R is becoming more and more popular in both academia and the clinical industry since it is free and open-source, which can also be an ideal platform for statistical analysis and data visualization. While SAS remains the main tool in pharma industries, it still pays a lot effort to learn the complex graph template language. R Shiny App provides a perfect solution for this obstacle. It could build a platform to directly connect programmers and users. Users who are not familiar with programming could do data clean or draw plots with simple clicks through the interactive web applications behind which are the complex programs developed by programmers.

In this paper, I would like to use a demo to show how to develop a user-friendly interactive tool using R Shiny. The demo tool would be constituted by two parts: data and charts with some basic functionalities that might be useful for daily data review in pharma industry.

INTRODUCTION

To design an R Shiny tool with multiple functions, what comes first is to write down our design thinking after we know the needs. Then we dive deeper to the design, what input and output should be added to realize our needs.

For the demo, we want to import the data from SAS datasets, display the data and do some basic tidy-up like sorting, filtering, and variable selection. After that, for the charts part, we also want to draw some plots using the dataset generated in the data part. We want to enable the user to choose the plot type and select the related parameters before they draw the plots.



Choose sas Dataset: ae.sas7bdat

Choose a dataset: ae.sas7bdat

☐ Filter

✓ Select

STUDYID, DOMAIN, AESEQ, AESPID, AEACN, AEOUT, AESTDY, AEENDY

USUBJID, AEDECOD, AESOC, AESTDTC, AEENDTC

✓ Order

AESOC × AEDECOD × USUBJID ×

Show entries Search:

USUBJID	AEDECOD	AESOC	AESTDTC	AEENDTC
1014	低氯血症	代谢及营养类疾病	2021-11-23T17:24	2021-11-27T10:59
1014	低氯血症	代谢及营养类疾病	2021-11-28T07:59	2021-12-05T06:38
1014	低氯血症	代谢及营养类疾病	2021-12-09T06:47	2021-12-13T06:38
1022	低氯血症	代谢及营养类疾病	2021-12-08T09:15	2021-12-10T08:57
1003	低白蛋白血症	代谢及营养类疾病	2021-03-22T07:00	2021-03-24T07:11
1003	低白蛋白血症	代谢及营养类疾病	2021-03-24T07:11	2021-04-06T09:02
1006	低白蛋白血症	代谢及营养类疾病	2021-05-10T08:51	2021-05-27
1006	低白蛋白血症	代谢及营养类疾病	2021-05-31T06:42	
USUBJID	AEDECOD	AESOC	AESTDTC	AEENDTC

Showing 1 to 8 of 827 entries

1 2 3 4 5 … 104

Choose plot type: Histogram

X: AESOC

✓ Horizontal

✓ Count subject

Figure 1 UI of Shiny Demo App

SHINY TOOL MIND MAP

A mind map or flowchart would definitely help a lot in design step.

I list all functions with the UI input and output methods in the below mind map.

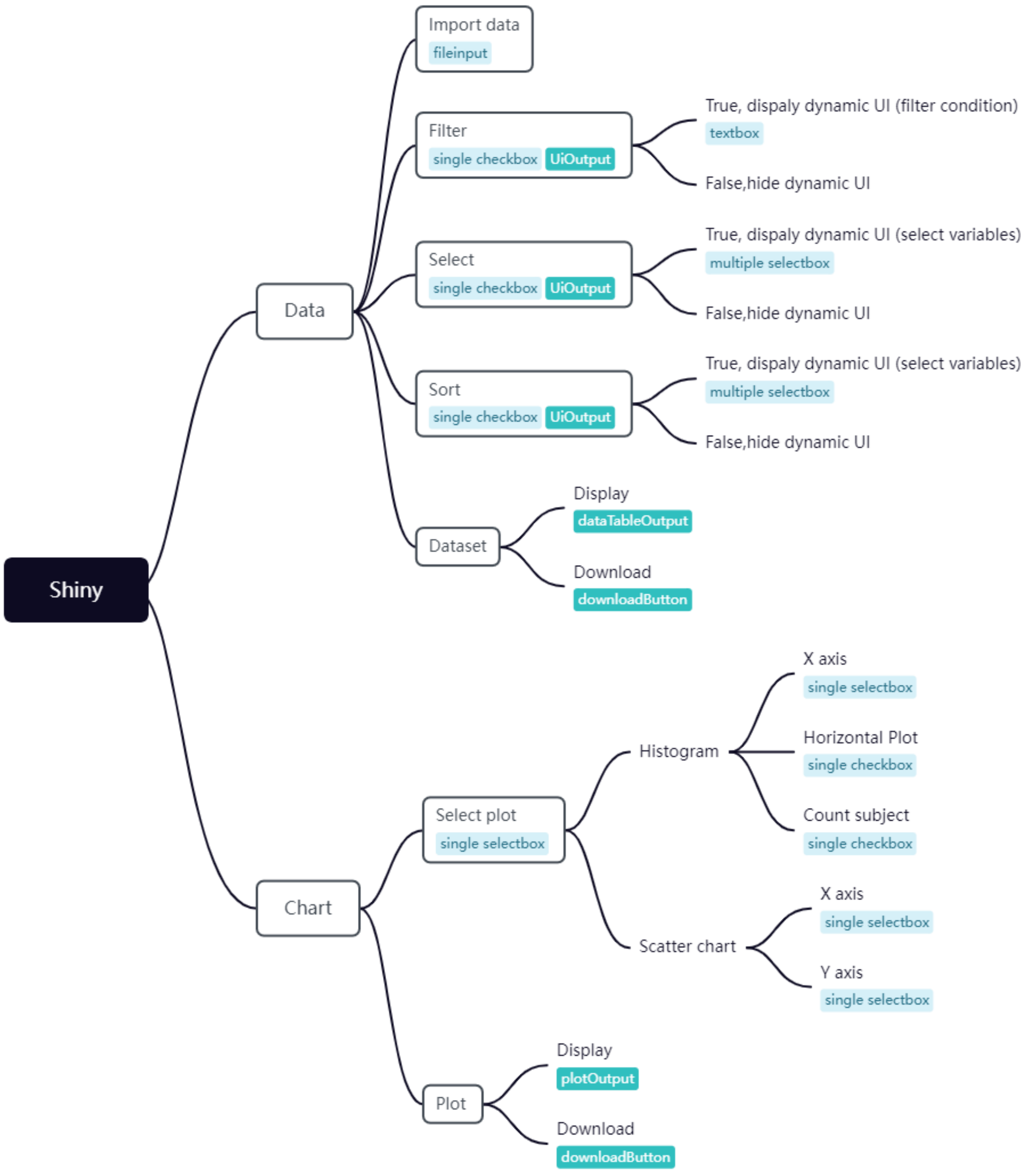


Figure 2.1 Mind Map

REACTIVE GRAPH

We might generate some intermediate step which we called reactive expression. We also choose draw a graph to explain the process of reactive execution.

In below reactive graph, blue is input, orange is reactive expression, green is output.

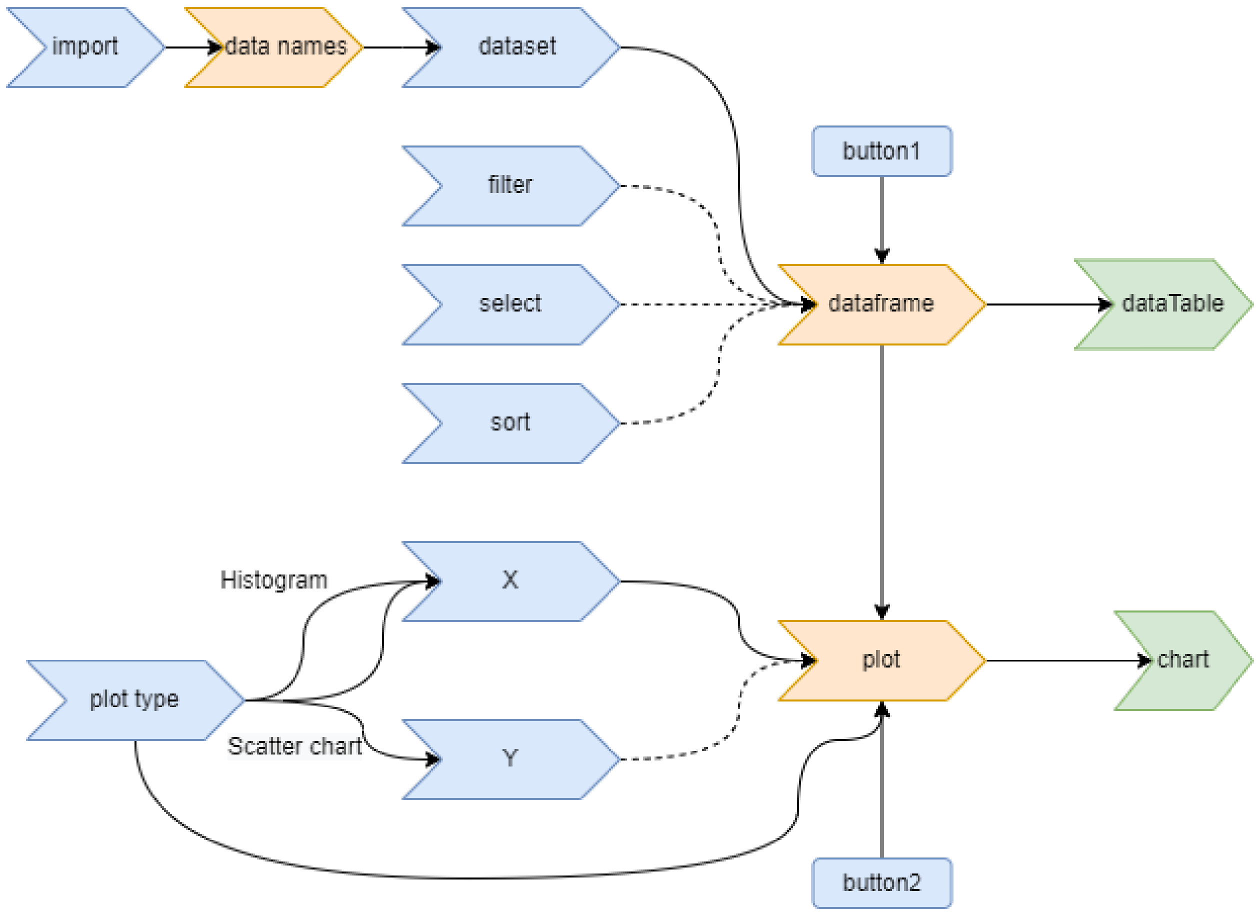


Figure 2.2 Reactive graph

SHINY EXTENSIONS

A lot of R packages could offer extended UI or server components for the R web framework Shiny.

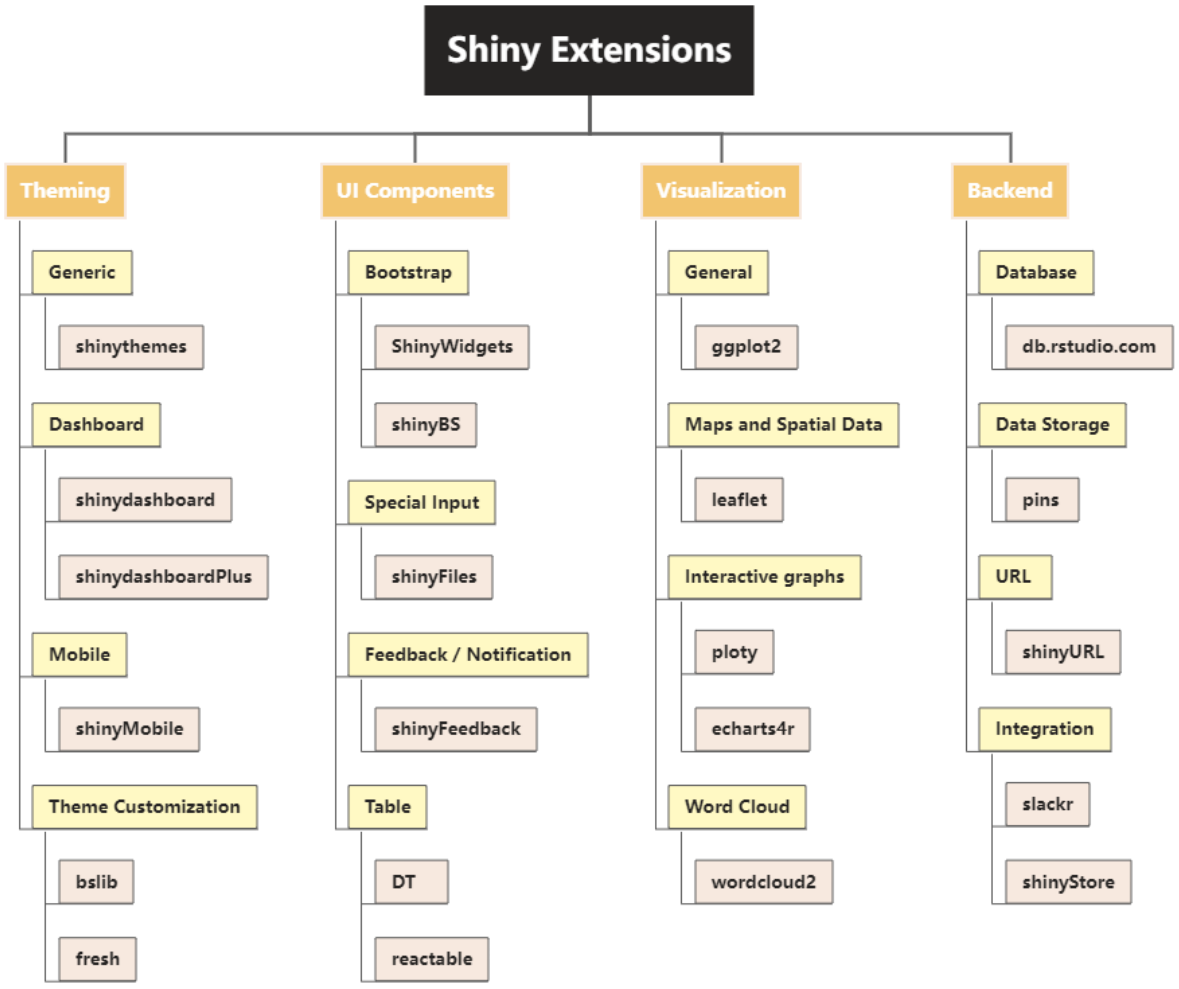


Figure 2.3 Shiny Extensions' packages

DETAILED USER-FRIENDLY DESIGN

1. DYNAMIC UI

As we know, shiny app is made of two main functions, UI and server.

UI, the frontend, what users see in the browser. We can use different types of input, such as textbox, checkbox, button, etc. These parts linked with app's input and output. if we want change partial UI by another input value, it's need to create dynamic user interfaces. For example, the variable selection is determined by the variables included in the dataset. our selection panel would change if we use different datasets.

Case 1 Updating Inputs

Imagine that you have a bunch of SAS datasets waiting for analysis and you want to save them as excel after your work is done which could be a common task in our daily work. We wish the tool could import multiple datasets at one time and select which one is in use now. Jump to the design, there should be two boxes, one for input datasets and one for dataset selection. No doubt, the options in single select box should be determined by those imported datasets. It's required to update the options in select box according to the import dataset names.

In shiny package, it has a series of update functions. For the case mentioned above, we can use updateSelectInput() function. Firstly, create a select input with choices equal to NULL in UI function, then use updateSelectInput() function to change the select box choice options according to our import datasets. At last, use freezeReactiveValue() to tell all downstream calculations that a new input value is stale and they should save their effort until it's useful.

UI Part:

```
selectInput(inputId = " dataset",
            label = " Choose a dataset:",
            choices = NULL),
```

Server part:

```
df<-reactive({
  fileNames_sas<-str_subset(input$import$name, ".sas7bdat")
})

observeEvent(df(), {
  freezeReactiveValue(input, " dataset")
  updateSelectInput(inputId = " dataset", choices = df())
})
```

Case 2 Show and Hide Partial UI

For filter, select and sort functions, we want them to show up only when we need to use the them. Use filter as example, we can add a single checkbox with ID "filter", when the input value is true, then we active the filter function code block, otherwise nothing will show up. This kind of design will also save the sidebar space.

The paired function uiOutput() and renderUI() would easily achieved this goal. Insert uiOutput into your UI function and it would leave a "hole" for your server code to fill in. Then renderUI() in your server function would fill the "hole" with dynamically generated UI. As the case below, uiOutput defines a dynamical UI called "add_filter", then in server we need to define the content of UI, so we add multiline text box "filter_text" and define the row as 3.

UI Part:

```
checkboxInput(inputId = "filter",
              label = "Filter",
              value = FALSE),
uiOutput("add_filter"),
```

Server part – add UI:

```
output$add_filter <- renderUI({
  if (input$filter == TRUE) {
      textAreaInput("filter_text", NULL, row=3)
     }
})
```

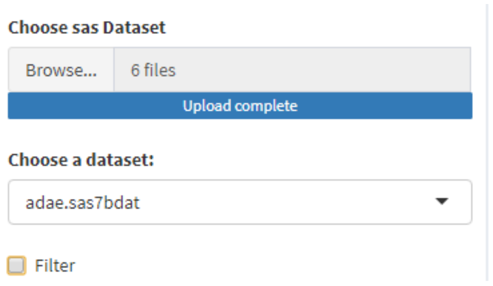


Figure 4.1.1 UI when Filter is inactive

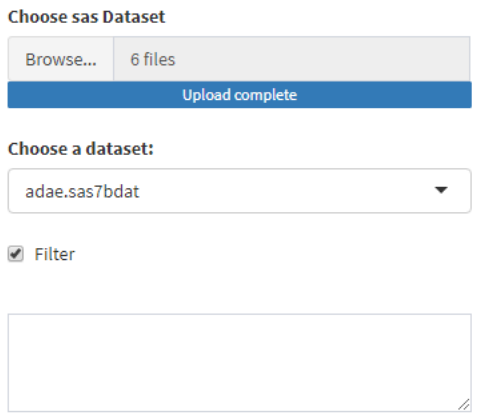


Figure 4.1.2 UI when Filter is active

Case 3 Conditional UI

For chart part, we want to enable users to choose the plot type. In the demo, we use histogram and scatter plot for example.

The histogram just needs the key parameter for X axis, but we want to add some functionalities to customize the plot. In the demo, we use two checkboxes to realize two simple functions: One is to change the plot from vertical bar(default) to horizontal; the other is to change the count from count records to subjects.

The scatter chart would need to state two key parameters for X and Y axis, so we use two single select boxes for it.

Since these two plots have different specialties, different UI should be used according to the value of plot type. There are many ways to achieve this goal, here we will show two ways.

Way 1: Put the unique user interface of each type into its own tabPanel. Then in UI, insert it after select the input “type”. Last, we would need updateTabsetPanel in server.

Function:

```
plot_tabs <- tabsetPanel(
  id = "plot",
  type = "hidden",
  tabPanel("Scatter chart",
           selectInput(inputId = "yaxis",
                       label = "Y:",
                       choices = NULL),
  ),
  tabPanel("Histogram",
           checkboxInput(inputId = "Horizontal",
                         label = "Horizontal",
                         value  = FALSE),
           checkboxInput(inputId = "Subject",
                         label = "Count subject",
                         value  = FALSE),
  ),
)
```

UI:

```
selectInput("type", "Choose plot type:",
             choices = c("Histogram","Scatter chart")),
selectInput(inputId = "xaxis",
            label = "X:",
            choices = NULL),
plot_tabs,
```

Server:

```
observeEvent(input$type, {
  updateTabsetPanel(inputId = "plot", selected = input$type)
})
```

Way2: Use conditionalPanel() function with uiOutput().

UI:

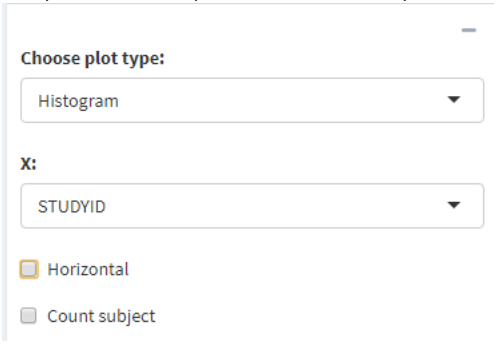
```
selectInput("type", "Choose plot type:",
             choices = c("Histogram","Scatter chart")),
   selectInput(inputId = "xaxis",
               label = "X:",
               choices = NULL),
   conditionalPanel("input.type == 'Histogram'",
                    uiOutput("ui_hist")),
   conditionalPanel("input.type != 'Histogram'",
                    uiOutput("ui_y"))
```

Server:

```
# add y axis UI panel
   output$ui_y <- renderUI({
     selectInput(inputId = "yaxis",
                 label = "Y:",
                 choices = names(df_1()))
   })

# add Histogram UI panel
   output$ui_hist <- renderUI({
     tagList(
       checkboxInput(inputId = "Horizontal",
                     label = "Horizontal",
                     value = FALSE),
       checkboxInput(inputId = "Subject",
                     label = "Count subject",
                     value = FALSE))
   })
```

These two ways will generate similar UI in this demo. However, if we need more complicated dynamical UI, you can compare different ways and choose the best one.

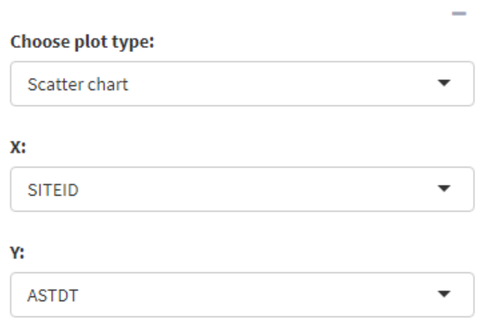

Choose plot type: Histogram

X: STUDYID

☐ Horizontal

☐ Count subject

Figure 4.1.3 UI when type is Histogram


Choose plot type:

Scatter chart

X:

SITEID

Y:

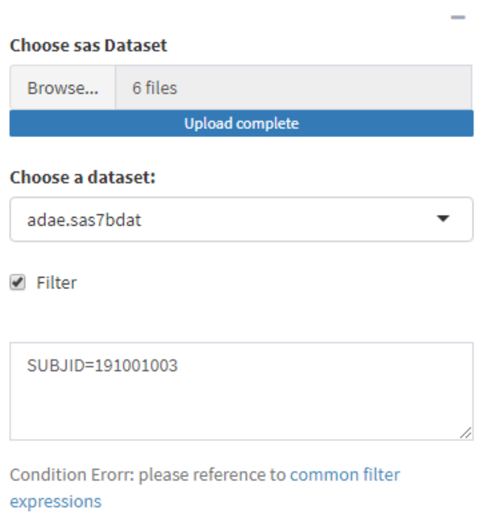
ASTDT

Figure 4.1.4 UI when type is Scatter Chart

2. SHOW NOTIFICATION WHEN ENCOUNTERED AN ERROR

When we run the shiny app, if we pass a wrong input to the app, an error will happen and cause the current session terminated. Just use the filter function for example, if the user type in a wrong filter condition, we want the app to show an error notification instead of terminating the whole session.
To avoid crashing the whole app, we can use tryCatch() to wrap the error-triggering code. And we also need to remind the users how to fill the filter condition in the correct way. For that, I write a help file about the filter operator details and store it in the google docs. When users fill in wrong conditions, the app will show a help text with hyperlink which links to the help doc.
Server:

```
   if (input$filter == TRUE) {
       req(input$filter_text)
       filter_cond <- isolate(input$filter_text)
       tryCatch({
         data1<-data()%>%filter(eval(parse(text=filter_cond)))
         output$ui_filter_error <- renderUI({
           return()
         })},
         error=function(err) {
           url <- a("common filter
           expressions",href="https://docs.google.com/document/d/1mtmM
           Bw94VYu3DqgUp-u63_11FEQ7X_iCCtmNgk6UL7A/edit?usp=sharing",
           target="_blank")
           output$ui_filter_error <- renderUI({
             helpText("Condition Erorr: please reference to ",url)
           })
           data1 <- data.frame()
           return(NULL)
         }
       )
   }else{
       data1<-data()
   }
```


Choose sas Dataset

Browse... 6 files

Upload complete

Choose a dataset:

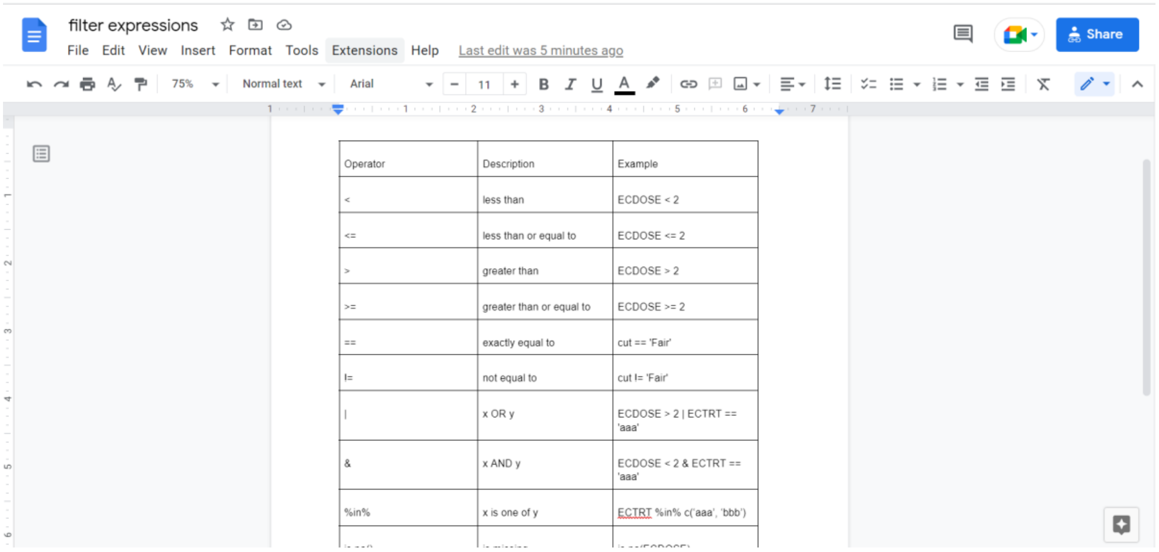
adae.sas7bdat

Filter

SUBJID=191001003

Condition Erorr: please reference to common filter expressions

Figure 4.2.1 UI when Show Notification


filter expressions
File Edit View Insert Format Tools Extensions Help Last edit was 5 minutes ago
Share
75% Normal text Arial 11

Operator	Description	Example
<	less than	ECDOSE < 2
<=	less than or equal to	ECDOSE <= 2
>	greater than	ECDOSE > 2
>=	greater than or equal to	ECDOSE >= 2
==	exactly equal to	cut == 'Fair'
!=	not equal to	cut != 'Fair'
|	x OR y	ECDOSE > 2 | ECTRT == 'aaa'
&	x AND y	ECDOSE < 2 & ECTRT == 'aaa'
%in%	x is one of y	ECTRT %in% c('aaa', 'bbb')

Figure 4.2.2 Google help doc

3. SAME TYPE INPUT FOR DIFFERENT NEEDS

For the select and sort function, we all need to use multiple select box, but we are using them for different purposes, so we need to make different designs.

In the select function, we add all variable names exist in the dataset to the list for user to manipulate. Besides, to make things easy, we enable users to select or delete multiple variables simply by using SHIFT and CTRL.

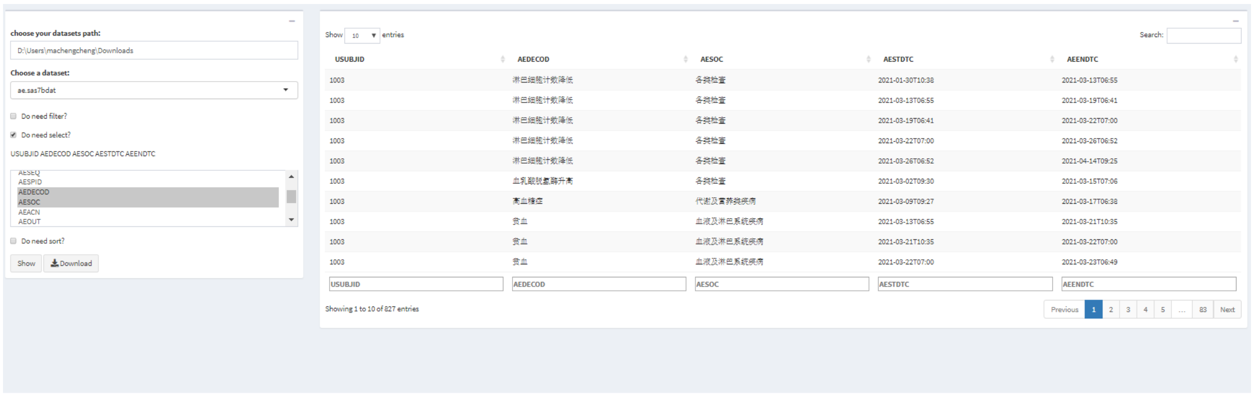
To use the standard HTML select input element instead of the JavaScript library selectize.js, we set selectInput() with selectize=F. In the meantime, set the size to get a drop-down list:

```
selectInput("vars", "", choices = names(data()),
        multiple = TRUE, size = min(5, length(names(data())), selectize = F)
```

Add a text UI to store the vars:

```
textOutput("vars_text")
```

```
    output$vars_text <- renderText({input$vars})
```



choose your datasets path:

D:\Users\machengcheng\Downloads

Choose a dataset:

ae.sas7bdat

☐ Do need filter?

☑ Do need select?

USUBJID AEDECOD AESOC AESTDTC AEENDTC

AESEQ
AESPID
AEDECOD
AESOC
AEACN
AEOUT

☐ Do need sort?

Show | Download

Show 10 entries Search:

USUBJID	AEDECOD	AESOC	AESTDTC	AEENDTC
1003	淋巴细胞计数降低	各类检查	2021-01-30T10:38	2021-03-13T06:55
1003	淋巴细胞计数降低	各类检查	2021-03-13T06:55	2021-03-19T06:41
1003	淋巴细胞计数降低	各类检查	2021-03-19T06:41	2021-03-22T07:00
1003	淋巴细胞计数降低	各类检查	2021-03-22T07:00	2021-03-26T06:52
1003	淋巴细胞计数降低	各类检查	2021-03-26T06:52	2021-04-14T09:25
1003	血乳酸脱氢酶升高	各类检查	2021-03-02T09:30	2021-03-15T07:06
1003	高血糖症	代谢及营养类疾病	2021-03-09T09:27	2021-03-17T06:38
1003	贫血	血液及淋巴系统疾病	2021-03-13T06:55	2021-03-21T10:35
1003	贫血	血液及淋巴系统疾病	2021-03-21T10:35	2021-03-22T07:00
1003	贫血	血液及淋巴系统疾病	2021-03-22T07:00	2021-03-23T06:49
USUBJID	AEDECOD	AESOC	AESTDTC	AEENDTC

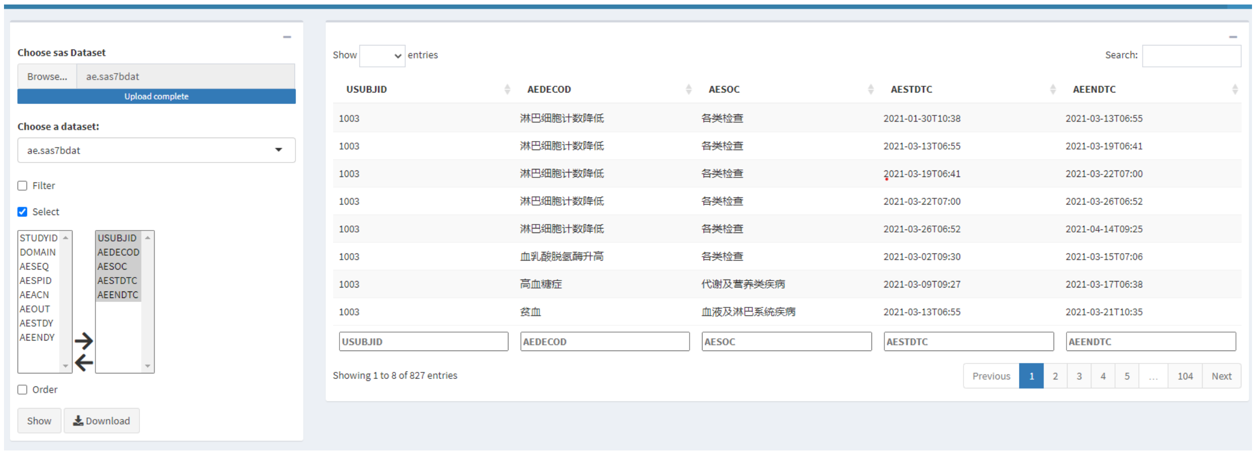
Showing 1 to 10 of 827 entries

Previous 1 2 3 4 5 … 83 Next

Figure 4.3.1 Select multiple select design 1

To make it more professional, if you have html, css, javascript skill, you can use them to customize a better UI input.

To make a more intuitive and easier way for the select function input, I also make an add-on by using css and javascript. I define two select boxes. The left one is the list of the variables available for choosing, the right one is the list of the variables already chosen. We can use the left and right arrows to modify.



Choose sas Dataset

Browse... ae.sas7bdat

Upload complete

Choose a dataset:

ae.sas7bdat

☐ Filter

☑ Select

STUDYID	USUBJID
DOMAIN	AEDECOD
AESEQ	AESOC
AESPID	AESTDTC
AEACN	AEENDTC
AEOUT	
AESTDY	
AEENDY	

☐ Order

Show | Download

Show entries Search:

USUBJID	AEDECOD	AESOC	AESTDTC	AEENDTC
1003	淋巴细胞计数降低	各类检查	2021-01-30T10:38	2021-03-13T06:55
1003	淋巴细胞计数降低	各类检查	2021-03-13T06:55	2021-03-19T06:41
1003	淋巴细胞计数降低	各类检查	2021-03-19T06:41	2021-03-22T07:00
1003	淋巴细胞计数降低	各类检查	2021-03-22T07:00	2021-03-26T06:52
1003	淋巴细胞计数降低	各类检查	2021-03-26T06:52	2021-04-14T09:25
1003	血乳酸脱氢酶升高	各类检查	2021-03-02T09:30	2021-03-15T07:06
1003	高血糖症	代谢及营养类疾病	2021-03-09T09:27	2021-03-17T06:38
1003	贫血	血液及淋巴系统疾病	2021-03-13T06:55	2021-03-21T10:35
USUBJID	AEDECOD	AESOC	AESTDTC	AEENDTC

Showing 1 to 8 of 827 entries

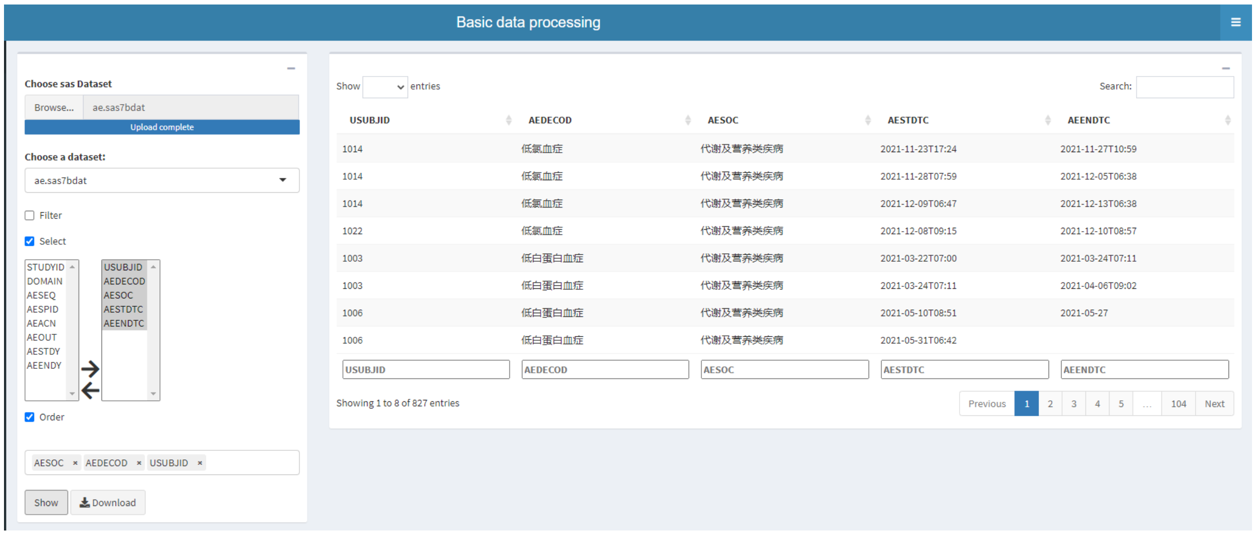
Previous 1 2 3 4 5 … 104 Next

Figure 4.3.2 Select multiple select design 2

For the sort function, we need to delete or change the sequence of the selected variables and show all sorting variables in the correct order.We could add the plugins option "remove_button" and "drag_drop" to realize it.

One step further, if the select function is active, the sort variable list should come from the selected variables, otherwise, it use all the variable names from the imported SAS datasets:

```
    if (input$order == TRUE) {
        if(input$select == TRUE){
           selectizeInput("order_vars", "", choices =input$select_vars$right,
                               multiple = TRUE, options = list(plugins=
                               list("remove_button", "drag_drop")))
        }else{
            selectizeInput("order_vars", "", choices =names(data()),
                                  multiple = TRUE, options = list(plugins =
            list("remove_button", "drag_drop")))
        }
    }
```



Basic data processing

Choose sas Dataset

Browse... ae.sas7bdat

Upload complete

Choose a dataset:

ae.sas7bdat

☐ Filter

☑ Select

STUDYID
DOMAIN
AESEQ
AESPID
AEACN
AEOUT
AESTDY
AEENDY

USUBJID
AEDECOD
AESOC
AESTDTC
AEENDTC

☑ Order

AESOC × AEDECOD × USUBJID ×

Show entries Search:

USUBJID	AEDECOD	AESOC	AESTDTC	AEENDTC
1014	低氯血症	代谢及营养类疾病	2021-11-23T17:24	2021-11-27T10:59
1014	低氯血症	代谢及营养类疾病	2021-11-28T07:59	2021-12-05T06:38
1014	低氯血症	代谢及营养类疾病	2021-12-09T06:47	2021-12-13T06:38
1022	低氯血症	代谢及营养类疾病	2021-12-08T09:15	2021-12-10T08:57
1003	低白蛋白血症	代谢及营养类疾病	2021-03-22T07:00	2021-03-24T07:11
1003	低白蛋白血症	代谢及营养类疾病	2021-03-24T07:11	2021-04-06T09:02
1006	低白蛋白血症	代谢及营养类疾病	2021-05-10T08:51	2021-05-27
1006	低白蛋白血症	代谢及营养类疾病	2021-05-31T06:42	
USUBJID	AEDECOD	AESOC	AESTDTC	AEENDTC

Showing 1 to 8 of 827 entries

Figure 4.3.3 Sort multiple select design

4. SHOW HISTOGRAM WITH LOTS OF X CATEGORIES

When we draw a histogram with variable which has multiple categories, especially with restricted plot area, it's hard to balance explicit details with the plot size. So why don't we split the huge plot into multiple sub-plots to make it easy to read.

First, calculate how many sub-plots we need to split, then create a list p to store all the plots, last we use do.call() and grid.arrange to combine these plots together:

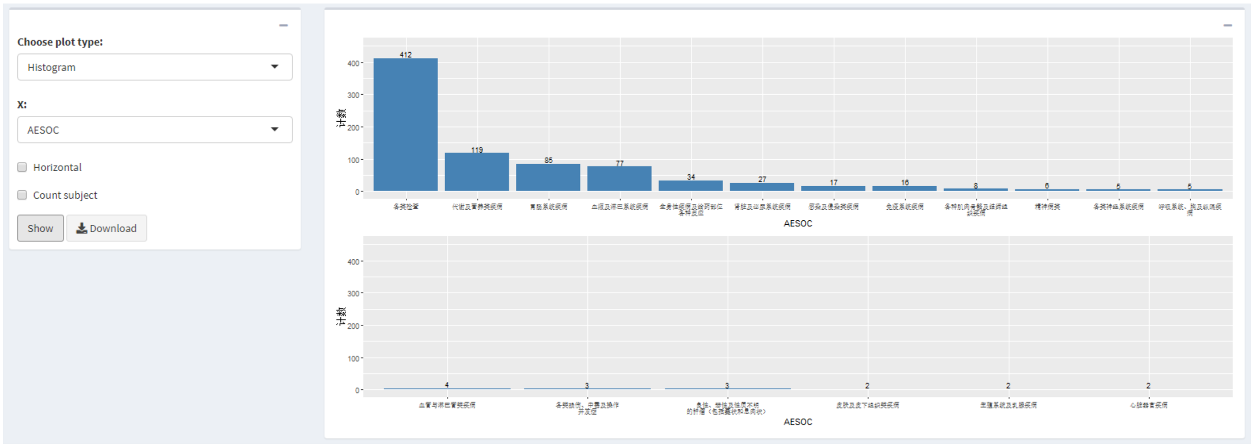
```
df0 <-data3 %>% count(.data[[input$xaxis]]) %>% arrange(desc(n))
var<-df0[[input$xaxis]]
length_var <- length(var)
nplot<- ceiling(length_var/12)
if(input$type=="Histogram"&input$Horizontal==F){
   p<-list()
   for (i in 1:nplot) {
      if((12*i)>length_var){
        list_var<-var[((i-1)*12+1):length_var]
      }else{
        list_var<-var[((i-1)*12+1):(12*i)]
      }
```

```
         data4<-data3%>%filter(.data[[input$xaxis]] %in% list_var)
         p[[i]]<-ggplot(data4,mapping= aes(x =
                 reorder_size(.data[[input$xaxis]],T), y = ..count..)) +
                 geom_bar(stat = 'count',fill="steelblue")+
                 geom_text(aes(label=..count..),stat = 'count',vjust=-
                 0.2,size = 3) +
                 ylim(0, 1.1*max(table(data3[[input$xaxis]])))+
                 theme(axis.text.x = element_text(size=8))+
                 scale_x_discrete(labels=function(x) str_replace(x,
                 "(^.{10})","\\1\n"))+
                 labs(x=isolate(input$xaxis),y='计数')
        }
        return(do.call(grid.arrange,c(p,ncol=1)))
   }
   else if(input$type=="Histogram"&input$Horizontal==T){
          plot1 <- ggplot(data3,mapping= aes(x =
                   reorder_size(.data[[input$xaxis]],F), y = ..count..)) +
                   geom_bar(stat = 'count',fill="steelblue") +
                   geom_text(aes(label=..count..),stat = 'count',hjust = -
                   0.2,size = 3) +
                   theme(axis.text.x = element_text(size=8))+
                   scale_x_discrete(labels=function(x) str_replace(x,
                   "(^.{10})","\\1\n"))+
                   coord_flip()+
                   labs(x=isolate(input$xaxis),y='计数')
         return(plot1)
   }
```

In addition, I sort the categories in descending order by count, it would be easier for user to find out which categories have more records. Users can also transfer the plot to the horizontal bar version through a simple click.

Meanwhile, we need to modify the plot height correspondingly, so we need to change the height parameter of plotOutput():

```
   if(input$Horizontal==F){
      plotOutput("Plot",
              height=(ceiling(length(unique(df_1()[[input$xaxis]]))/15)*250))
   }
   else{
      plotOutput("Plot",
              height=(150+length(unique(df_1()[[input$xaxis]]))*20))
   }
```



Choose plot type: Histogram

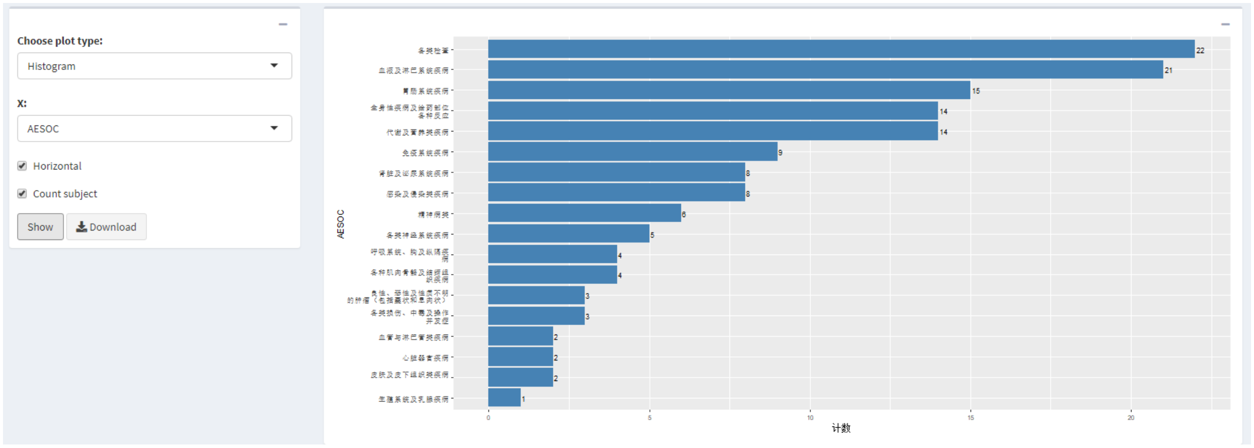
X: AESOC

☐ Horizontal

☐ Count subject

Show | Download

Figure 4.3.4 Histogram plot



Choose plot type: Histogram

X: AESOC

☑ Horizontal

☑ Count subject

Show | Download

Figure 4.3.5 Histogram plot (horizontal bar and count subject)

5. SHOW CHINESE CHARACTER IN SHINYAPP SERVER

We should pay more attention to publish our shiny app to server, it might have more issues but they not exist in local shiny app, such as it doesn't display Chinese character in shinyapp io server. Because the server is a Linux environment, but we usually use window environment. Add below code to the global file, and submit font file together with your main program.

Code to fix the Chinese character:

```
options(shiny.usecairo = FALSE)
font_home <- function(path = '') file.path('~', '.fonts', path)
if (Sys.info()[['sysname']] == 'Linux') {
    dir.create(font_home())
    file.copy('Chinese_font.ttc', font_home())
    system2('fc-cache', paste('-f', font_home()))
}
rm(font_home)
if (.Platform$OS.type == "windows") {
    if (!grepl("Chinese", Sys.getlocale())) {
        Sys.setlocale(, 'Chinese')
  }
}
```

CONCLUSION

This paper is to share an example of the Shiny app design. Programmer can use Shiny to develop interactive tools which meet pharmaceutical and biotech industries' needs. While building the tools, stand in users' point of view, and take users' behaviors, thoughts into your consideration will make the tools better.

REFERENCES

Hadley, Wickham. 2020. Mastering Shiny. 1005 Gravenstein Highway North, CA 95472. O'Reilly Inc.
Yihui, Xie. "Unicode characters". 2018. https://github.com/rstudio/shiny-examples/tree/main/022-unicode-chinese
Joe, Cheng. "Custom input control". 2016. https://github.com/rstudio/shiny-examples/tree/main/036-custom-input-control
Vincent, Nijs. Travis, Cl. Carson, Sievert. Jim, Hester "Data Menu for Radiant: Business Analytics using R and Shiny". 2020. https://github.com/radiant-rstats/radiant.data
Nan, Xiao. etc. "Awesome Shiny Extensions". https://github.com/nanxstats/awesome-shiny-extensions#pdf

ACKNOWLEDGMENTS

Thanks for the inspiring and supporting from Gang Liu, Zhenghang Dai and statistical programming team of Legend.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Name: Chengcheng Ma
Enterprise: Legend
E-mail: Chengcheng.ma@legendbiotech.com