

Learning *R*

Carl James Schwarz

StatMathComp Consulting by Schwarz
cschwarz.stat.sfu.ca @ gmail.com

Introduction to *Shiny*

1. *Shiny* - interactivity to your analysis
 - 1.1 Introduction
 - 1.2 Selecting variables - dealing with non-standard evaluation

Shiny - Interactivity to R applications

Wonderful but design very carefully.

Suggested References:

- <https://https://shiny.rstudio.com/articles/>
- <https://shiny.rstudio.com/gallery/>
- <https://www.showmeshiny.com/>

Why use *Shiny* vs. other visualization packages such as *ggviz*?

- Able to use standard *R* packages such as *gplot2* without having to learn new graphing routines
- Not necessary to know HTML, CSS, or JavaScript
- Able to prototype very quickly

Why not use *Shiny*?

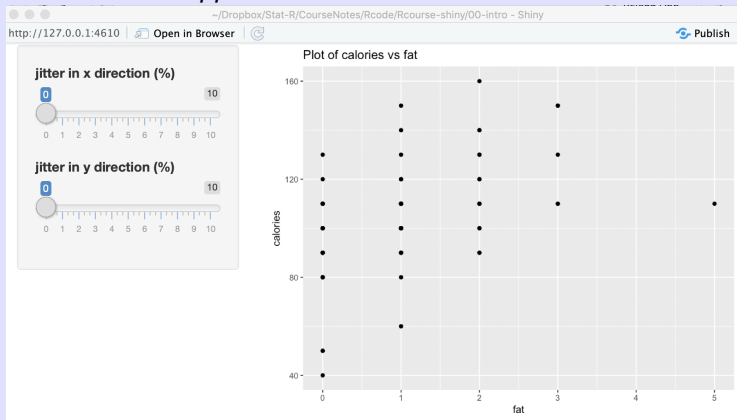
- Need *R* and *Rexperts* to develop complicated applications
- Security is an afterthought.
- May bog down with very large datasets.

How are *Shiny* applications structured?

- Separate directory for each application
- *app.R* is the file name containing the *Shiny* interface
 - User interface function (what to display; where to display; tools to display)
 - Server function - create the display items in reaction to changes to user changes
- Other functions, data sets, etc. as needed.

Shiny - Example

Open `app.r` in `00-Intro` in the `Rcourse-shiny` directory.
Don't forget to set the working directory.
Press the `Run app` button.



Play with the sliders.

Shiny - How structured I

How are *Shiny* applications structured?

Look at the code from the above *Shiny* app;

```
1 server <- function(input, output) {
2   output$scatterplot <- renderPlot({
3     ggplot(data=cereal, aes(x=fat, y=calories))+
4       ggtitle("Plot of calories vs fat")+
5       geom_point(position=
6         position_jitter(
7           h=input$jy*mean(cereal$calories)/100,
8           w=input$jx*mean(cereal$fat)/100))
9   })
10 }
```

The *server()* takes the *input* arguments and creates the *output* object (a list).

Notice that this is not a proper *R* function since results are passed BACK through the *output* argument.

The *renderPlot* is what is supposed to be displayed.

Shiny - How structured II

```
1 ui <- fluidPage(  
2   sidebarLayout(  
3     sidebarPanel(  
4       sliderInput("jx",  
5         "jitter in x direction (%) ",  
6         min = 0,  
7         max = 10,  
8         value = 0),  
9       sliderInput("jy",  
10        "jitter in y direction (%) ",  
11        min = 0,  
12        max = 10,  
13        value = 0)  
14     ), # end of sidebar Panel  
15     mainPanel(  
16       plotOutput("scatterplot")  
17     ) # end of mainPanel  
18   )
```

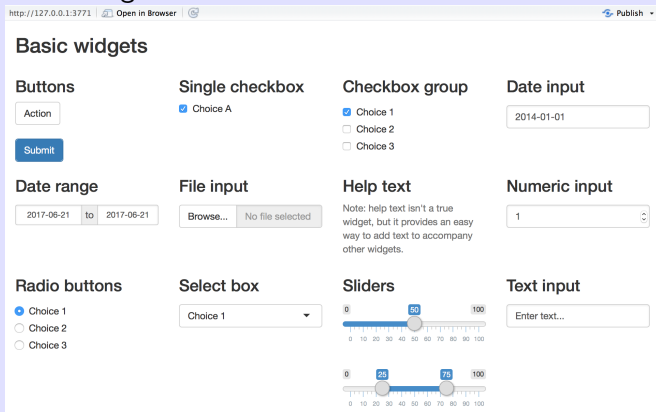

19)

This defines the user interface with types of pages (*fluidPage()*), types of panels etc.

```
1 # Run the application
2 shinyApp(ui = ui, server = server)
```

This is what meshes the two functions together.

Basic widgets:



The screenshot displays a Shiny web application interface with the following widgets:

- Buttons:** Two buttons labeled "Action" and "Submit".
- Single checkbox:** A checkbox labeled "Choice A" which is checked.
- Checkbox group:** A group of three checkboxes labeled "Choice 1", "Choice 2", and "Choice 3". "Choice 1" is checked.
- Date input:** A date input field containing "2014-01-01".
- Date range:** A date range input field showing "2017-06-21" to "2017-06-21".
- File input:** A file input field with a "Browse..." button and the text "No file selected".
- Help text:** A text area containing the note: "Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets."
- Numeric input:** A numeric input field containing the value "1".
- Radio buttons:** A group of three radio buttons labeled "Choice 1", "Choice 2", and "Choice 3". "Choice 1" is selected.
- Select box:** A dropdown menu labeled "Choice 1".
- Sliders:** Two horizontal sliders. The top slider has a value of 80, and the bottom slider has a value of 75.
- Text input:** A text input field with the placeholder text "Enter text...".

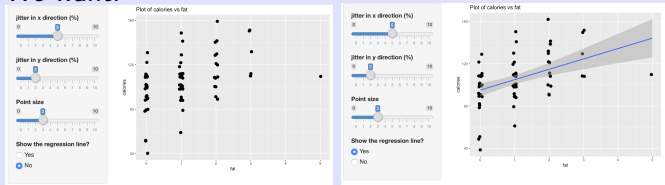
Also see

<http://shiny.rstudio.com/gallery/widget-gallery.html>

Exercise

- Add a slider for the the size of the points
- Add a radio button (*radioButtons*) to add a regression line (Yes/No)

We want:



Build incrementally,

- add the slider that does nothing;
- then react to the slide;
- then add the buttons that do nothing;
- then react to the buttons.

The new user controls:

```
1 ui <- fluidPage(  
2   sidebarLayout(  
3     ....  
4       sliderInput("psize",  
5                   "Point size",  
6                   min = 0,  
7                   max = 10,  
8                   value = 1),  
9       radioButtons("fitline",  
10                  "Show the regression line?",  
11                  c("Yes"="Yes",  
12                    "No" ="No")) )  
13   ), # end of sidebar Panel  
14   mainPanel(  
15     plotOutput("scatterplot")  
16   ) # end of mainPanel  ))
```

The new server function:

```
1 server <- function(input, output) {
2   output$scatterplot <- renderPlot({
3     plot1 <- ggplot(data=cereal, aes(x=fat, y=calories))+
4       ggtitle("Plot of calories vs fat")+
5       geom_point(position=position_jitter(
6         h=input$jy*mean(cereal$calories)/100,
7         w=input$jx*mean(cereal$fat)/100),
8         size=input$psize)
9     if(input$fitline == "Yes"){
10      plot1 <- plot1 + geom_smooth(method="lm")}
11     plot1 # be sure to return the plot at the end
12   })
13 }
```

We often want to select VARIABLES to plot. Run the following:

```
1 # Consider the following plot
2 ggplot(data=cereal, aes(y=calories, x=fat))+
3   geom_point()
4
5 # Suppose we wish to "program" the variables using something
6 xvar <- 'fat'
7 yvar <- 'calories'
8
9 # this fails because of non-standard evaluation
10 ggplot(data=cereal, aes(y=yvar, x=xvar))+
11   geom_point()
```

What happens is a problem of non-standard evaluation that occurs throughout R!

What does $y=yvar$ mean and how is it distinguished from $y=cereal$?

ggplot2 includes the *aes_string* to deal with this problem. *dplyr* also allows for non-standard evaluation.

```
1 xvar <- 'fat'  
2 yvar <- 'calories'  
3 ggplot(data=cereal, aes_string(y=yvar, x=xvar))+  
4   geom_point()
```

This does what you want.

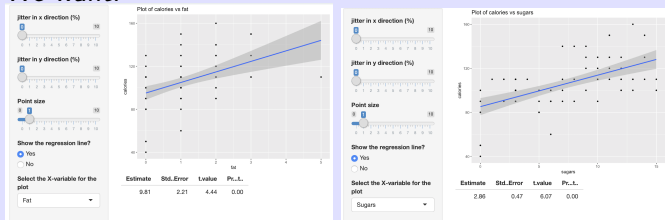
Similarly

```
1 # use [] in regular data frames  
2 yvar <- 'calories'  
3 cereal$yvar # returns null  
4 cereal$"yvar" # also returns null  
5 cereal[, yvar, drop=FALSE]
```

Exercise

- Add a radio drop down list to select among the x variables from *fat*, *protein*, *carbo*, and *sugars*.
- Also print the estimated slope and se under the plot (`tableOutput()` etc.)

We want:



The new user controls:

```
1 ui <- fluidPage(  
2   sidebarLayout(  
3     sidebarPanel(...,  
4       selectInput("xvar",  
5         "Select the X-variable for the plot",  
6         c("Fat" ='fat', "Carbohydrates"='carbo'  
7     ), # end of sidebar Panel  
8     mainPanel(  
9       plotOutput("scatterplot"),  
10      tableOutput("slope")  
11    ) # end of mainPanel  
12  )  
13 )
```

Notice how two output panels can be stacked using *mainPanel()*

The new server function:

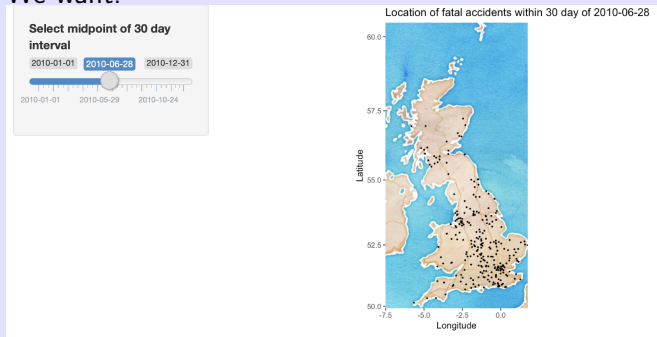
```
1 server <- function(input, output) {
2   output$scatterplot <- renderPlot({
3     plot1 <- ggplot(data=cereal, aes_string(x=input$xvar, y=
4       ggtitle(paste("Plot of calories vs ", input$xvar, sep=
5       geom_point(position=position_jitter(
6         h=input$jy*mean(cereal$calories)/100,
7         w=input$jx*mean(cereal[, xvar,drop=TRUE])/100),
8         size=input$psize)
9     if(input$fitline == "Yes"){ plot1 <- plot1 + geom_smooth
10    plot1 # be sure to return plot a end
11  })
12  output$slope <- renderTable({
13    fit <- lm( calories ~ cereal[, input$xvar], data=cereal
14    data.frame(summary(fit)$coefficients[2,, drop=FALSE])
15  })
16 }
```

Notice how we dealt with non-standard evaluation.
Notice how we created `data.frame` to hold results.

Refer to accident data base.

- Use a slider to select a date between.
- Find all fatal accidents within 30 days of the date
- Plot them on a map using *ggmap* as shown previously

We want:



How to select a map using *ggmap*. Do this once (at top of code) and do not update.

```
1 mean.lat <- mean(accidents$Latitude)
2 mean.long<- mean(accidents$Longitude)
3
4 my.map.dl <- ggmap::get_map(c(left  =min(accidents$Longitude)
5                               right =max(accidents$Longitude)
6                               matype="watercolor",  source="")
7
8 my.map <- ggmap(my.map.dl)
9 ....
10 # This code fragment plots the location of fatal accidents
11   plot1 <- my.map +
12     ggtitle(paste("Location of fatal accidents within :"))
13     geom_point(data=myfatal, aes(x=Longitude, y=Latitude))
14     ylab("Latitude")+xlab("Longitude")
15   plot1 # be sure to return plot a end
```

The new user controls:

```
1 ui <- fluidPage(  
2   sidebarLayout(  
3     sidebarPanel(  
4       sliderInput("Date",  
5                   "Select midpoint of 30 day interval",  
6                   min=as.Date('2010-01-01'), max=as.Date(  
7                     round=TRUE)  
8       ), # end of sidebar Panel  
9     mainPanel(  
10      plotOutput("fatalplot")  
11    ) # end of mainPanel  
12  )  
13 )
```

Unfortunately, it is NOT easy to drop the colored bar in the slider (???? seems should be a easy task)?

The new server function:

```
1 server <- function(input, output) {
2   output$fatalplot <- renderPlot({
3     myfatal <- fatal[ abs(fatal$Date-input$Date)<30,]
4
5     plot1 <- my.map +
6       ggtitle(paste("Location of fatal accidents within 30 days of", input$Date)) +
7       geom_point(data=myfatal, aes(x=Longitude, y=Latitude)) +
8       ylab("Latitude")+xlab("Longitude")
9     plot1 # be sure to return plot a end
10  })
11 }
```

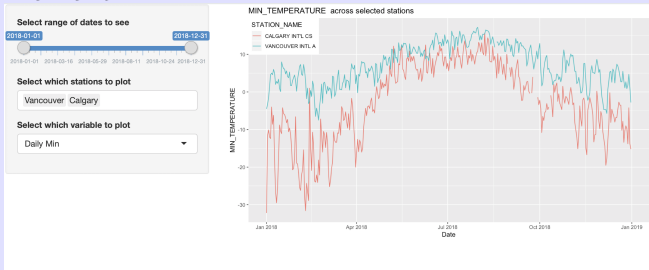
Notice how we selected the data based on the date from the input slider

Refer to weather records for 2018 at weather stations across Canada.

Design a *Shiny* app to:

- Use a slider to select range of dates in 2018.
- Select one of number of variables, e.g. minimum daily temperature, maximum daily temperature, total precipitation, etc
- Select one or more of stations (e.g. from a small selected list.)

We want:



The new user controls:

```
1 ui <- fluidPage(  
2   sidebarLayout(  
3     sidebarPanel(  
4       sliderInput("Date",  
5         "Select range of dates to see",  
6         min=as.Date('2018-01-01'), max=as.Date(  
7         round=TRUE),  
8       selectInput("stations",  
9         "Select which stations to plot",  
10        c("Vancouver"="VANCOUVER INTL A",  
11          "Victoria" = "VICTORIA INTL A",  
12          "Edmonton" = "EDMONTON INTERNATIONAL C",  
13          "Calgary" = "CALGARY INT'L CS",  
14          "Winnipeg" = "WINNIPEG A CS",  
15          "Toronto" = "TORONTO INTL A"),  
16        multiple=TRUE),  
17       selectInput("variable",
```

```
18         "Select which variable to plot",
19         c("Daily Min"="MIN_TEMPERATURE",
20           "Daily Max"="MAX_TEMPERATURE",
21           "Total Precip"="TOTAL_PRECIPITATION"))
22     ), # end of sidebar Panel
23     mainPanel(
24       plotOutput("climateplot")
25     ) # end of mainPanel
26   )
27 )
```

The new server function:

```

1 server <- function(input, output) {
2   output$climateplot <- renderPlot({
3
4     myclimate <- climate[ climate$Date ≥ input$Date[1] & cl
5     myclimate <- myclimate[ myclimate$STATION_NAME %in% input
6     plot1 <- ggplot(data=myclimate, aes_string(x="Date", y=
7       ggtitle(paste(input$variable, " across selected sta
8   #     geom_point()+
9     geom_line()+
10    ylab(input$variable)+xlab("Date")+
11    theme(legend.position=c(0,1),legend.justification=c
12    plot1 # be sure to return plot a end
13  })
14 }
```

Notice how we selected the data based on the date from the input slider

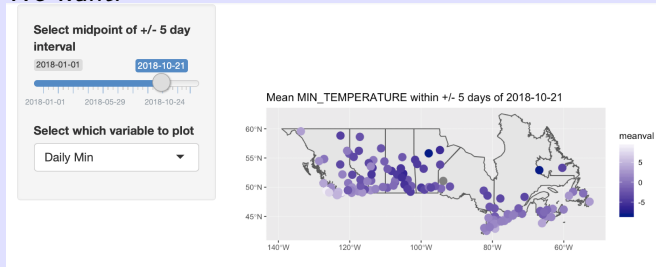
Notice how we dealt with non-standard evaluation in `ggplot()`

Refer to weather records for 2018 at weather stations across Canada.

Design a *Shiny* app to:

- Use a slider to select a date in 2018.
- Select one of number of variables, e.g. minimum daily temperature, maximum daily temperature
- Plot the mean of that variable at each station in a suitable interval (e.g. ± 5 days) from the selected date.

We want:



The controls are similar to previous exercises and so nothing new here.

```
1 ui <- fluidPage(  
2   sidebarLayout(  
3     sidebarPanel(  
4       sliderInput("Date",  
5                   "Select midpoint of +/- 5 day interval",  
6                   min=as.Date('2018-01-01'), max=as.Date('2018-01-01'),  
7                   round=TRUE),  
8       selectInput("variable",  
9                   "Select which variable to plot",  
10                  c("Daily Min"="MIN_TEMPERATURE",  
11                    "Daily Max"="MAX_TEMPERATURE"))  
12     ), # end of sidebar Panel  
13     mainPanel(  
14       plotOutput("climateplot")  
15     ) # end of mainPanel  
16   )
```

17)

The new server function:

```
1 server <- function(input, output) {
2   output$climateplot <- renderPlot({
3     #browser()
4     myclimate <- climate[ abs(climate$Date-input$Date)<5,]
5     myclimate$var.to.analyze <- myclimate[, input$variable]
6     # get the mean for each station
7     mean.climate <-
8       myclimate %>%
9         group_by(STATION_NAME,x, y) %>%
10          summarize(
11            meanval=mean(var.to.analyze, na.rm=TRUE)
12          )
13     mean.climate.sf <- st_as_sf(x = mean.climate, coords = c
14     plot1 <- ggplot() +
15       ggtitle(paste("Mean ", input$variable," within +/-
16       geom_sf(data=s.canada.sf, aes(fill=NULL))+
17       geom_sf(data=mean.climate.sf, aes(color=meanval), s
```

```
18         scale_color_gradient2(midpoint=10, low="darkblue",
19         plot1 # be sure to return plot a end
20     })
21 }
22 }
```

Notice how we selected the data based on the date from the input slider

Notice how we dealt with non-standard evaluation in `dplyr()`

Check the app for introductory code in preparing the map.