

labJournal4

Keiichiro Watanabe

2023-09-18

Part 0:

3 most interesting, different from other programming languages, or helpful) I learned

Comprehensive Libraries: R has one of the most comprehensive sets of packages for statistical modeling (like `glm`, `lmer`, `nls`, etc.), machine learning (like `caret`, `xgboost`, etc.), and data visualization (like `ggplot2`, `lattice`, etc.).

Tidy Data Principles: The tidyverse, a collection of R packages including `dplyr`, `tidyr`, `ggplot2`, and others, has brought about a paradigm shift in the way R users manipulate and visualize data. The principles are different from the traditional methods and can be incredibly efficient.

Data Frame as a First-Class Citizen: While many languages require external libraries to deal with data tables, R comes with built-in support for data frames. This is a perfect match for anyone working with datasets.

Part 1: R & Shiny

1

```
library(shiny)
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(tm)
```

```
## Loading required package: NLP
```

```
library(RColorBrewer)
```

```
text <- c("Data", "Science", "Machine", "R", "Python", "Data", "Machine", "Learning", "R", "Statistics")
```

```
# Define the UI
```

```
ui <- fluidPage(
```

```
# Application title
```

```
  titlePanel("Word Cloud"),
```

```

sidebarLayout(
  # Sidebar with a slider and selection inputs
  sidebarPanel(
    selectInput("selection", "Choose a topic:",
               choices = c("CS words" = "sample", "random words" = "another")),
    actionButton("update", "Change"),
    hr(),
    sliderInput("freq",
               "Minimum Frequency:",
               min = 1, max = 50, value = 15),
    sliderInput("max",
               "Maximum Number of Words:",
               min = 1, max = 300, value = 100)
  ),

  # Show Word Cloud
  mainPanel(
    plotOutput("plot")
  )
)

# Define the server
server <- function(input, output) {
  text_data <- reactive({
    if (input$selection == "sample") {
      return(text)
    } else if (input$selection == "another") {
      return(c("Different", "think", "Text", "Hello", "Hello", "think"))
    }
  })

  output$plot <- renderPlot({
    # Create a Corpus and Document-Term Matrix
    corpus <- Corpus(VectorSource(text_data()))
    dtm <- DocumentTermMatrix(corpus)

    # Convert the DTM to a matrix
    dtm_matrix <- as.matrix(dtm)

    # Calculate word frequencies
    word_freqs <- colSums(dtm_matrix)

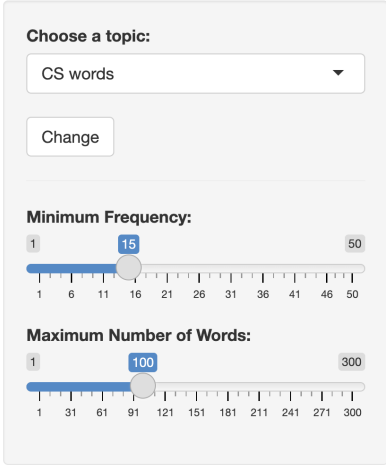
    wordcloud(
      words = names(word_freqs),
      freq = word_freqs,
      min.freq = input$freq,
      max.words = input$max,
      scale = c(3, 0.5),
      colors = brewer.pal(8, "Dark2")
    )
  })
}

```

```
# Run the Shiny app
shinyApp(ui, server)
```

I recreated similar Shiny apps using word cloud. My word cloud now changes colors and now has tool bars such as minimum frequency bar and minimum number of words bar. I can switch to different word clouds by

Word Cloud



statistics
learning
visualization
python
machine
data
science

choosing a topic using the drop down menu.

```
library(shiny)
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:NLP':
##
##   annotate
```

```
set.seed(123)
num_samples <- 1000
your_data <- data.frame(shares = rpois(num_samples, lambda = 1000))

# UI
ui <- fluidPage(
  titlePanel("Article Shares Histogram"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("binwidth", "Bin Width:", min = 10, max = 200, value = 100)
    ),
    mainPanel(
      plotOutput("histogram")
    )
  )
)
```

```

)
)

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

  output$histogram <- renderPlot({
    # Create a histogram of the 'shares'
    gg <- ggplot(your_data, aes(x = shares)) +
      geom_histogram(binwidth = input$binwidth, fill = "skyblue", color = "black", alpha = 0.7) +
      labs(x = "Shares", y = "Frequency") +
      theme_minimal() +
      ggtitle("Distribution of Article Shares") +
      labs(subtitle = paste("Bin Width:", input$binwidth))

    # Calculate and add mean and median annotations
    mean_shares <- mean(your_data$shares)
    median_shares <- median(your_data$shares)

    gg <- gg +
      geom_vline(xintercept = mean_shares, color = "red", linetype = "dashed", size = 1) +
      annotate("text", x = mean_shares, y = 50, label = paste("Mean:", round(mean_shares, 2)), color = "red", size = 12) +
      geom_vline(xintercept = median_shares, color = "blue", linetype = "dashed", size = 1) +
      annotate("text", x = median_shares, y = 100, label = paste("Median:", round(median_shares, 2)), color = "blue", size = 12)

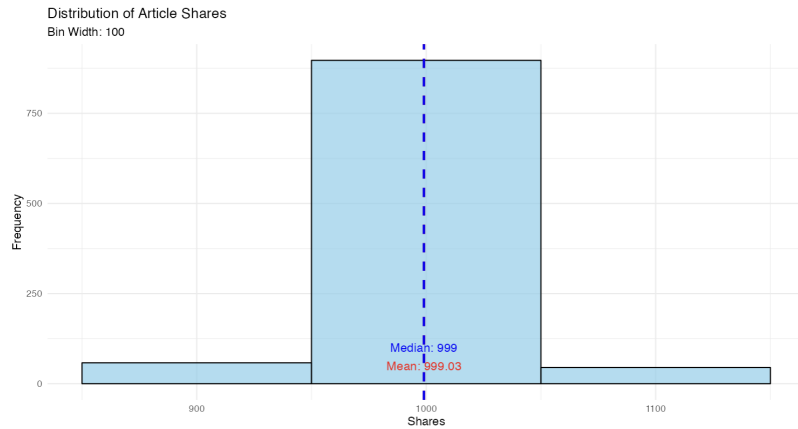
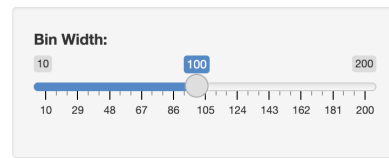
    print(gg)
  })
}

# Run the Shiny app
shinyApp(ui = ui, server = server)

```

I recreated similar Shiny apps using histogram. My histogram now have colors and can change bar width by

Article Shares Histogram



using the toggle.

2

```
library(shiny)
library(shinydashboard)
```

```
##
## Attaching package: 'shinydashboard'
```

```
## The following object is masked from 'package:graphics':
##
## box
```

```
library(shinythemes)

# UI using the cerulean theme
ui_cerulean <- dashboardPage(
  dashboardHeader(title = "Cerulean Themed Dashboard"),
  dashboardSidebar(),
  dashboardBody(
    fluidRow(
      box(
        title = "Cerulean Themed Box",
        "This is a box with the cerulean theme.",
        width = 4,
        height = 200,
        style = "background-color: #5BC0DE; color: white;"
      )
    )
  )
)
```

```

)

# UI using the cosmo theme
ui_cosmo <- dashboardPage(
  dashboardHeader(title = "Cosmo Themed Dashboard"),
  dashboardSidebar(),
  dashboardBody(
    # Add content here
    fluidRow(
      box(
        title = "Cosmo Themed Box",
        "This is a box with the cosmo theme.",
        width = 4,
        height = 200,
        style = "background-color: #0275D8; color: white;"
      )
    )
  )
)

# UI using the flatly theme
ui_flatly <- dashboardPage(
  dashboardHeader(title = "Flatly Themed Dashboard"),
  dashboardSidebar(),
  dashboardBody(
    fluidRow(
      box(
        title = "Flatly Themed Box",
        "This is a box with the flatly theme.",
        width = 4,
        height = 200,
        style = "background-color: #18BC9C; color: white;"
      )
    )
  )
)

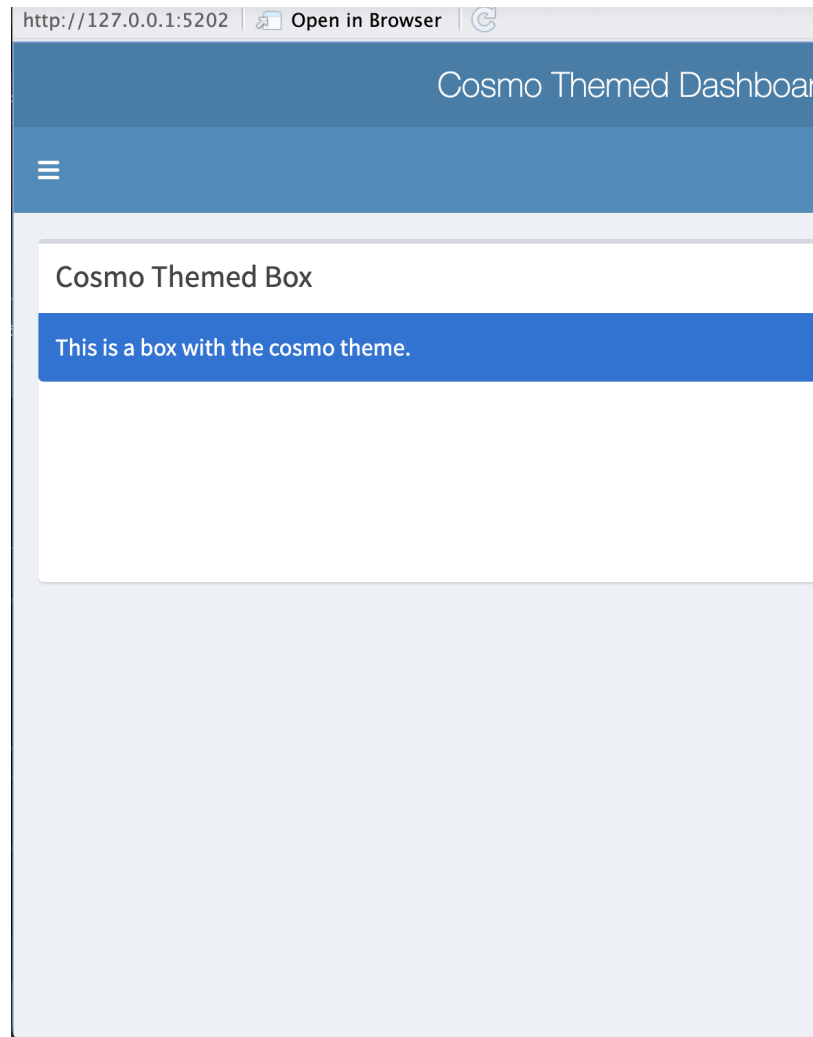
server <- function(input, output) {
}

# Run the Shiny apps with different themes
shinyApp(ui_cosmo, server)

```

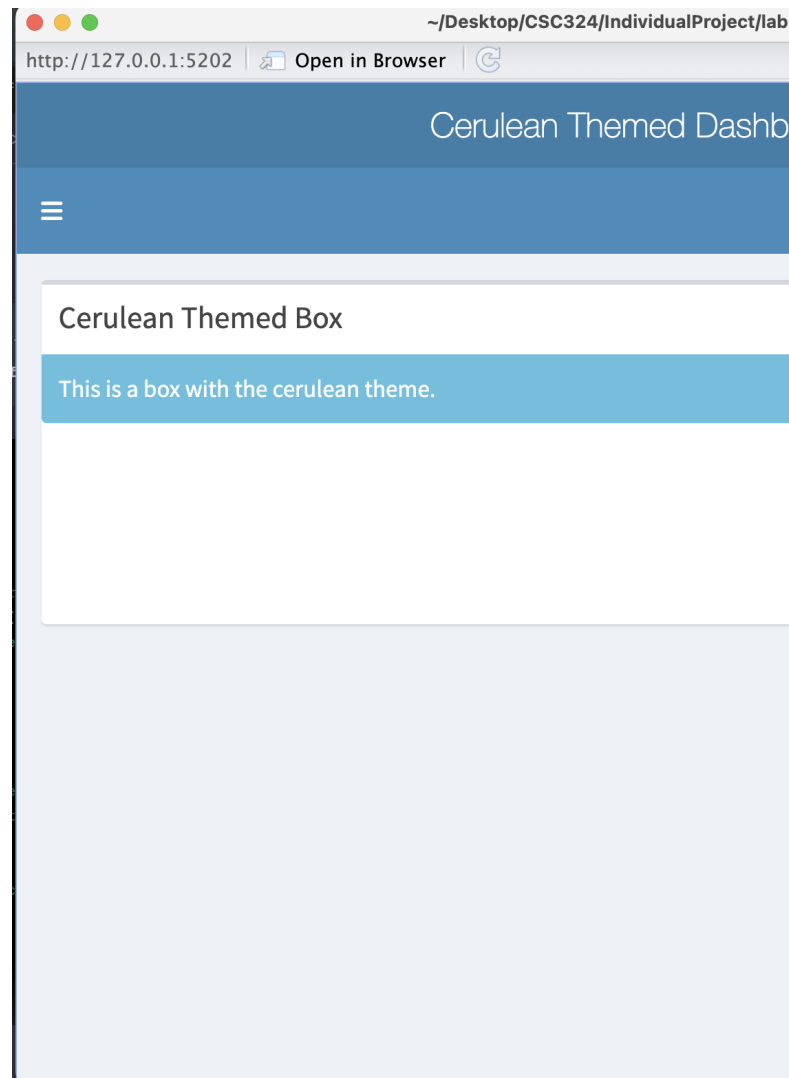
```
shinyApp(ui_cerulean, server)
```

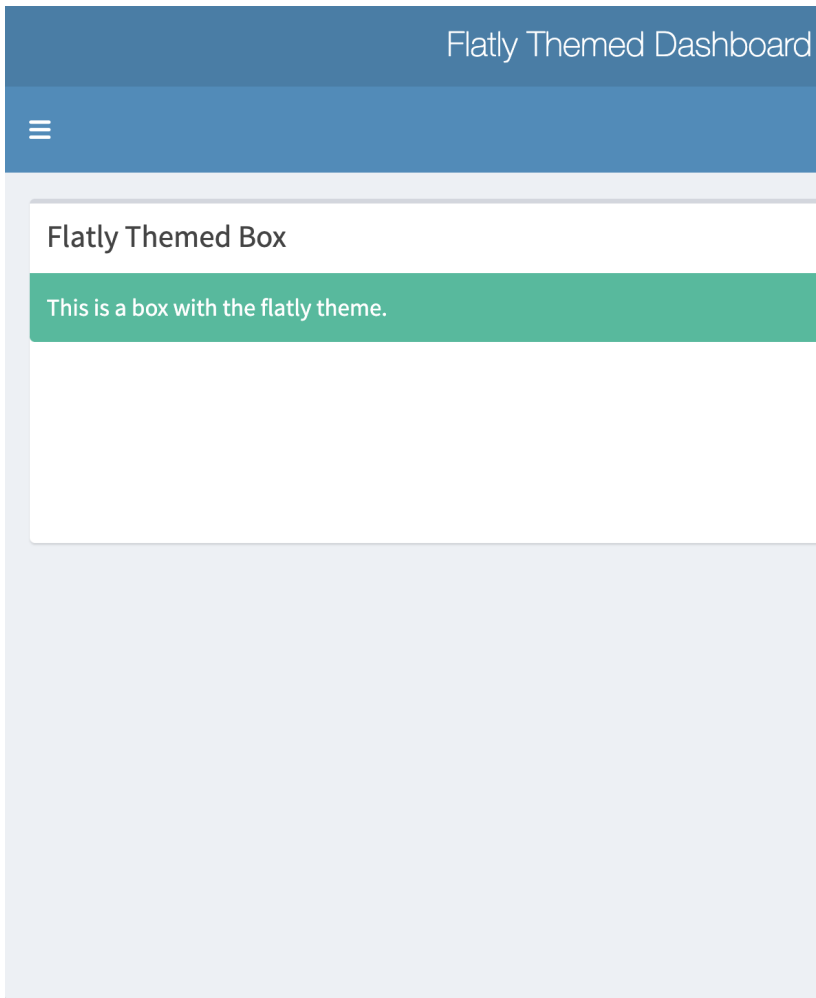
```
shinyApp(ui_flatly, server)
```



This is a screenshot of a dashboard using Cosmo theme.

This is a screenshot of a dashboard using Cerulean theme.





This is a screenshot of a dashboard using Flatly theme.

3

`%in%` operator is used to test whether elements from one vector are present in another vector. It returns a logical vector of the same length as the left-hand operand, indicating whether each element on the left-hand side is found in the right-hand operand.

Example 1: Checking for the presence of elements in a vector.

```
# Create two vectors
vector1 <- c(1, 2, 3, 4, 5)
vector2 <- c(3, 5, 7, 9)

# Use %in% to check if elements of vector1 are in vector2
result <- vector1 %in% vector2

# Display the result
print(result) # Output will be a logical vector: TRUE FALSE TRUE FALSE TRUE
```



```
## [1] FALSE FALSE  TRUE FALSE  TRUE
```

Example 2: Filtering a data frame based on a condition.

```

# Create a data frame
df <- data.frame(ID = c(101, 102, 103, 104, 105),
                  Name = c("Alice", "Bob", "Charlie", "David", "Eve"))

# Create a vector of IDs to filter
ids_to_filter <- c(102, 105)

# Use %in% to filter the data frame
filtered_df <- df[df$ID %in% ids_to_filter, ]

# Display the filtered data frame
print(filtered_df)

```

```

##      ID Name
## 2 102  Bob
## 5 105  Eve

```

Example 3: Checking for the presence of specific values in a character vector.

```

# Create a character vector
fruits <- c("apple", "banana", "cherry", "date", "fig")

# Create a vector of fruits to check for
fruits_to_check <- c("apple", "grape", "cherry")

# Use %in% to check if fruits_to_check are in the fruits vector
result <- fruits_to_check %in% fruits

# Display the result
print(result) # Output will be a logical vector: TRUE FALSE TRUE

```

```

## [1] TRUE FALSE TRUE

```

4

`paste()`: This function concatenates character vectors with a specified separator (default is a space).

`paste0()`: This function concatenates character vectors without any separator.

Example 1: Using `paste()` to concatenate strings with a separator.

```

# Create two character vectors
first_names <- c("John", "Alice", "Bob")
last_names <- c("Doe", "Smith", "Johnson")

# Use paste() to combine first names and last names with a space separator
full_names <- paste(first_names, last_names)

# Display the result
print(full_names) # Output will be "John Doe" "Alice Smith" "Bob Johnson"

```

```

## [1] "John Doe"      "Alice Smith"   "Bob Johnson"

```

Example 2: Using `paste0()` to concatenate strings without a separator.

```
# Create two character vectors
fruits <- c("apple", "banana", "cherry")
colors <- c("red", "yellow", "red")

# Use paste0() to combine fruits and colors without a separator
fruit_colors <- paste0(fruits, colors)

# Display the result
print(fruit_colors) # Output will be "applered" "bananayellow" "cherryred"
```



```
## [1] "applered"      "bananayellow" "cherryred"
```

5

1. Print statement debugging is the process of inserting print statements (or equivalent logging statements) into your code to output the values of variables, control flow information, or any other relevant information during program execution.
2. Interactive debugging involves using a debugger tool, which is a specialized program that allows you to interactively inspect the state of your program, set breakpoints, and step through code one line at a time while monitoring variable values.

6

Example 1: Using Inline CSS in the HTML Head Usage in My Project: I can use inline CSS to dynamically style elements of my Shiny app based on user interactions or data. For example, I could change the background color or font color of certain elements that are important like number of shares an article has.

Example 2: Using includeCSS with an External File Usage in My Project: Including an external CSS file can be very useful for maintaining clean and organized styling across my entire Shiny app. I can create a custom CSS file (style.css) and apply consistent styles to different visualizations of my app for a polished and professional look.

Example 3: Using a Style in an Individual HTML Tag Usage in My Project: I might use this approach to highlight certain critical information or create visual emphasis on specific content within my Shiny app like results of my data analysis.

7

1. I learned that “shiny.react” provides integration between Shiny and React, allowing R developers to use React components within their Shiny applications. This combination could offer advantages like a more dynamic and interactive user interface, improved performance, and access to a wider range of modern web development tools.
2. Further by using React, you can create reusable components for your Shiny app’s user interface. Each interactive element, such as drop downs, input fields, and plots, can be encapsulated within React components.
3. Also, You can leverage Fluent UI components for creating a consistent and visually appealing user interface for your Shiny app. shiny.fluent gives you app a professional look and seamless integration with R & Shiny

Part 2: Reflection and questions on readings

1

In Chapter 10 of Tamara Munzner’s book on visualization, the emphasis is on the effective use of color and other visual channels to convey information accurately and intuitively. One of the key takeaways is the importance of considering human perception while selecting color schemes. Not all color palettes are equally effective for all types of data; for instance, sequential palettes work well for ordered data, while qualitative palettes are suitable for categorical data. Munzner also stresses on the significance of considering color blindness, cultural variations, and other accessibility issues.

Another critical point is the use of additional visual channels like size, shape, or texture to augment the information portrayed. Each of these channels has its own strengths and limitations, and one needs to consider the task at hand (e.g., comparison, identification) while making the selection. Munzner advises that these channels should complement, not confuse, thus supporting a cohesive narrative.

For my project on predicting the number of shares for Mashable articles, I will consider using a heatmap with a diverging color scheme to display correlations among features. Red could indicate positive correlation and blue for negative. I can use size as a visual channel in scatter plots, where larger markers signify more shares. I can employ shape to differentiate article types like tech, lifestyle, or politics. A sequential color scheme in time-series plots can indicate article age, transitioning from light to dark green. These visual channels can provide an intuitive and multifaceted way to understand the factors affecting article shares.

2

What does the article explain?

The article outlines various frameworks used in design thinking, emphasizing that the core principles are more important than the specific steps or terminology. The most interesting for me was the three of these frameworks:

Double Diamond Model: Introduced by the British Design Council, this model focuses on four phases—Discover, Define, Develop, and Deliver. It places an emphasis on both divergent and convergent thinking, making room for creativity and refinement.

Collective Action Toolkit (CAT) by Frog Design: This framework aims for social impact and is more community-centric. It consists of six non-linear stages—Clarify, Build, Seek, Imagine, Make, and Plan—emphasizing iterative processes and group collaboration.

Designing for Growth by Jeanne Liedtka & Tim Ogilvie: This approach reframes design thinking questions into four inquisitive “What” questions: What is? What if? What wows? What works? It aims for a more intuitive and inquisitive methodology.

What did you learn about it?

What I learned from this is that design thinking is not a one-size-fits-all approach; it’s a flexible, adaptable methodology with various frameworks that cater to different needs, tasks, and objectives. Each framework brings something unique to the table but adheres to the central tenets of design thinking—human-centered problem-solving, iteration, and collaboration.

3

What does the article explain?

The article by Simon, a software engineer at Datawrapper, discusses the issues with the way election polls are reported in the media, particularly as Germany prepares for an important election. He identifies three major problems: the lack of attention to margins of error, the need to account for deviating polls from different sources, and the absence of historical context to understand how polls change over time. To fix these problems, he suggests emphasizing the margin of error, aggregating data from multiple sources, and using annotations in charts to explain how public opinion changes due to events.

What did you learn about it?

I learned that election poll reporting can often be misleading and overlook important aspects like margins of error and bias. The article argues for a more nuanced approach to presenting this data, both to improve accuracy and provide a richer context for understanding what the numbers actually signify. It advocates for more responsible and detailed reporting of election polls.

How does this relate to your project? Can you apply it?

Since I am working on predicting the popularity of articles based on various factors, the principles in the article can certainly be applied. First, I can clearly indicate the confidence intervals or margins of error around my predictions. Next, I will consider data from multiple models or sources to provide a more rounded view. Also I will use annotations or a timeline to show how the popularity of certain topics or keywords has changed over time. The article's emphasis on transparency, multiple viewpoints, and historical context can enrich my project's data presentation and interpretation.

Reference

Name(s) of all authors: Keiichiro Watanabe, Megan Bernacchi

Lab: Lab Journal 2

Due Monday, 11 September 2023

Written/online sources used: (enter "none" if no sources other than the textbook and course website used)
: none

Help obtained (Acknowledgments): none

"I/we confirm that the above list of sources is complete AND that I/we have not talked to anyone else about the solution to this problem."