Sentiment Analysis and Shiny Integration

Stat 220

Bastola

February 23 2022

Shiny Reactivity

> An alert system that lets Shiny know exactly which expressions need to be re-run

- R usually works in a linear fashion
- When writing a Shiny app, we need to tell Shiny which chunks of your code should be reactive to events such as users changing the input values in the control widgets.
- Events are monitored and when they occur, the code reacts to those events.

Reactive Expressions

> A reactive expression is defined as one that transforms the reactive inputs to reactive outputs

Reactive expressions can be useful for caching the results of any procedure that happens in response to user input, including:

- accessing a database
- reading data from a file
- downloading data over the network
- performing an expensive computation

Covid Example Recap

```
# Example of a Shiny server function

server <- function(input, output) {
    filtered_data <- reactive({
        subset(MNdata,
               Counties %in% input$dv &
               month >= input$monthInput[1] &
                 month <= input$monthInput[2] &
                 year == input$yearInput)})

    output$plot <- renderPlot({
        ggplot(filtered_data(), aes(x=dates, y=cases, color="Counties")) +
        theme_economist_white() +
        geom_point(alpha=0.5, color = "blue") +
        theme(legend.position = "none") +
        ylab("Number of Cases") +
        xlab("Date")})

    output$table <- DT::renderDataTable({
        filtered_data()})

}
```

Covid Example Recap

```
# Example of a Shiny server function

server <- function(input, output) {
    filtered_data <- reactive({
        subset(MNdata,
               Counties %in% input$dv &
               month >= input$monthInput[1] &
                 month <= input$monthInput[2] &
                 year == input$yearInput)})

    output$plot <- renderPlot({
        ggplot(filtered_data(), aes(x=dates, y=cases, color="Counties")) +
        theme_economist_white() +
        geom_point(alpha=0.5, color = "blue") +
        theme(legend.position = "none") +
        ylab("Number of Cases") +
        xlab("Date")})

    output$table <- DT::renderDataTable({
        filtered_data()})

}
```

Covid Example Recap

```
# Example of a Shiny server function

server <- function(input, output) {
    filtered_data <- reactive({
        subset(MNdata,
               Counties %in% input$dv &
               month >= input$monthInput[1] &
                 month <= input$monthInput[2] &
                 year == input$yearInput)})

    output$plot <- renderPlot({
        ggplot(filtered_data(), aes(x=dates, y=cases, color="Counties")) +
        theme_economist_white() +
        geom_point(alpha=0.5, color = "blue") +
        theme(legend.position = "none") +
        ylab("Number of Cases") +
        xlab("Date")})

    output$table <- DT::renderDataTable({
        filtered_data()})

}
```

Reactive Values

> Reactive values contain values that can be read by other reactive objects.

- The input object is a ReactiveValues object, which looks something like a list, and it contains many individual reactive values.
- The values in input are set by input from the web browser.

```
# Example of a Shiny server function

server <- function(input, output) {
  output$distPlot <- renderPlot({
    hist(rnorm(input$obs))
  })
}
```

Observers

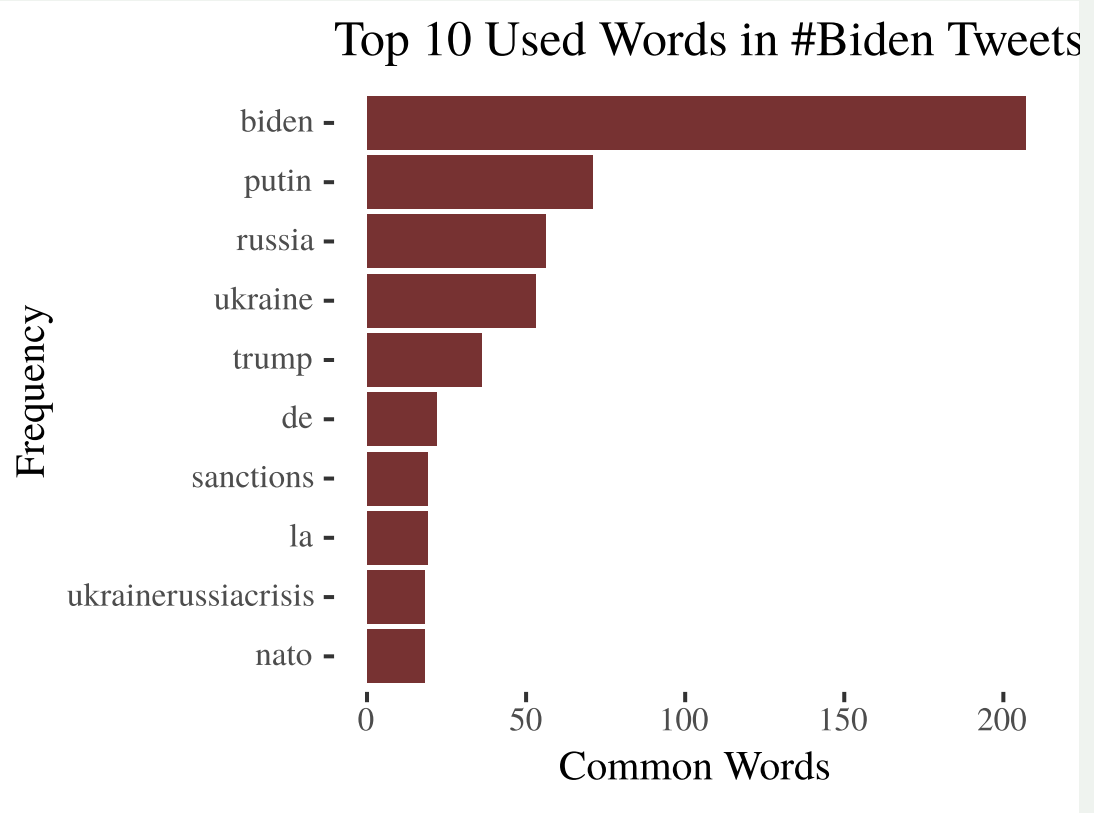
> Observers can access reactive values and reactive expressions.

- Observers do not return any values, and therefore do not cache their return values.
- Instead of returning values, they have side effects, which typically involves sending data to the web browser.

```
ui <- fluidPage(
  mainPanel(
    actionButton("button1","Button 1"),
    actionButton("button2","Button 2")
  )
)
server <- function(input, output) {
  # observe button 1 press.
  observe({
    input$button1
    input$button2
    showModal(modalDialog(
      title = "Button pressed", "You pressed one of the buttons!"
    ))
  })
}
```

Plot the top words

```
# Plot of top 10 words
words %>% count(word, sort=TRUE) %>%
  top_n(10) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(x = word, y = n)) +
  geom_col(fill ="#773232") +
  xlab(NULL) +
  coord_flip() +
  theme_tufte() +
  labs(x = "Frequency",
       y = "Common Words",
       title = "Top 10 Used Words in #Biden Twe
```



Sentiment Bing

```
# Function to take in tweet and return clean words with sentiment scores
sentiment_bing <- function(tweet){
  tweet_tbl <- tibble(text = tweet) %>%
  mutate(text = str_replace_all(text, replace_reg, "")) %>%
  unnest_tokens(word, text, token = "words") %>%
  anti_join(stop_words, by = "word") %>%
  filter(str_detect(word, "[a-z]")) %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  mutate(score = case_when(
        sentiment == 'negative'~n*(-1),
        sentiment == 'positive'~n*1))

  sentiment_score = case_when(
    nrow(tweet_tbl)==0~0,
    nrow(tweet_tbl)>0~sum(tweet_tbl$score)
  )

  zero_type = case_when(
    nrow(tweet_tbl)==0~"Zero",
    nrow(tweet_tbl)>0~"NoZero"
  )
list(score = sentiment_score, type = zero_type, tweet_tbl = tweet_tbl)
}
```

Sentiment Bing

```
# Function to take in tweet and return clean words with sentiment scores
sentiment_bing <- function(tweet){
  tweet_tbl <- tibble(text = tweet) %>%
  mutate(text = str_replace_all(text, replace_reg, "")) %>%
  unnest_tokens(word, text, token = "words") %>%
  anti_join(stop_words, by = "word") %>%
  filter(str_detect(word, "[a-z]")) %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  mutate(score = case_when(
        sentiment == 'negative'~n*(-1),
        sentiment == 'positive'~n*1))

  sentiment_score = case_when(
    nrow(tweet_tbl)==0~0,
    nrow(tweet_tbl)>0~sum(tweet_tbl$score)
  )

  zero_type = case_when(
    nrow(tweet_tbl)==0~"Zero",
    nrow(tweet_tbl)>0~"NoZero"
  )
list(score = sentiment_score, type = zero_type, tweet_tbl = tweet_tbl)
}
```

Sentiment Bing

```
# Function to take in tweet and return clean words with sentiment scores
sentiment_bing <- function(tweet){
  tweet_tbl <- tibble(text = tweet) %>%
  mutate(text = str_replace_all(text, replace_reg, "")) %>%
  unnest_tokens(word, text, token = "words") %>%
  anti_join(stop_words, by = "word") %>%
  filter(str_detect(word, "[a-z]")) %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  mutate(score = case_when(
        sentiment == 'negative'~n*(-1),
        sentiment == 'positive'~n*1))

  sentiment_score = case_when(
    nrow(tweet_tbl)==0~0,
    nrow(tweet_tbl)>0~sum(tweet_tbl$score)
  )

  zero_type = case_when(
    nrow(tweet_tbl)==0~"Zero",
    nrow(tweet_tbl)>0~"NoZero"
  )
list(score = sentiment_score, type = zero_type, tweet_tbl = tweet_tbl)
}
```

Sentiment Bing

```
# Function to take in tweet and return clean words with sentiment scores
sentiment_bing <- function(tweet){
  tweet_tbl <- tibble(text = tweet) %>%
  mutate(text = str_replace_all(text, replace_reg, "")) %>%
  unnest_tokens(word, text, token = "words") %>%
  anti_join(stop_words, by = "word") %>%
  filter(str_detect(word, "[a-z]")) %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  mutate(score = case_when(
        sentiment == 'negative'~n*(-1),
        sentiment == 'positive'~n*1))

  sentiment_score = case_when(
    nrow(tweet_tbl)==0~0,
    nrow(tweet_tbl)>0~sum(tweet_tbl$score)
  )

  zero_type = case_when(
    nrow(tweet_tbl)==0~"Zero",
    nrow(tweet_tbl)>0~"NoZero"
  )
list(score = sentiment_score, type = zero_type, tweet_tbl = tweet_tbl)
}
```

Sentiment Bing

```
# Function to take in tweet and return clean words with sentiment scores
sentiment_bing <- function(tweet){
  tweet_tbl <- tibble(text = tweet) %>%
  mutate(text = str_replace_all(text, replace_reg, "")) %>%
  unnest_tokens(word, text, token = "words") %>%
  anti_join(stop_words, by = "word") %>%
  filter(str_detect(word, "[a-z]")) %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  mutate(score = case_when(
        sentiment == 'negative'~n*(-1),
        sentiment == 'positive'~n*1))

  sentiment_score = case_when(
    nrow(tweet_tbl)==0~0,
    nrow(tweet_tbl)>0~sum(tweet_tbl$score)
  )

  zero_type = case_when(
    nrow(tweet_tbl)==0~"Zero",
    nrow(tweet_tbl)>0~"NoZero"
  )
list(score = sentiment_score, type = zero_type, tweet_tbl = tweet_tbl)
}
```

Run the function

```
sentiment_bing(tweets_twitterdata$text)
$score
[1] -78

$type
[1] "NoZero"

$tweet_tbl
# A tibble: 159 × 4
   word           sentiment     n score
   <chr>          <chr>     <int> <dbl>
 1 trump          positive     36    36
 2 die            negative      8    -8
 3 aggression     negative      6    -6
 4 traitor        negative      6    -6
 5 breaking       negative      5    -5
 6 authoritarian negative      3    -3
 7 destroy        negative      3    -3
 8 threat         negative      3    -3
 9 weak           negative      3    -3
10 attack         negative      2    -2
# … with 149 more rows
```

Data preparation using `lapply` amd `map`

```
twitterdata_sent = lapply(tweets_twitterdata$text[1:10], function(x) sentiment_bing(x))

twitter_sentiment = bind_rows(
  tibble(
    name = hashtag_to_search,
    score = unlist(purrr::map(twitterdata_sent, 'score')),
    type = unlist(purrr::map(twitterdata_sent, 'type'))
  )
)

twitter_sentiment
# A tibble: 10 × 3
   name   score type
   <chr>  <dbl> <chr>
 1 #Biden     0 Zero
 2 #Biden     1 NoZero
 3 #Biden     0 Zero
 4 #Biden    -4 NoZero
 5 #Biden     0 Zero
 6 #Biden    -2 NoZero
 7 #Biden     0 Zero
 8 #Biden     0 NoZero
 9 #Biden    -1 NoZero
10 #Biden     0 Zero
```

Group Work 1

Please clone the repository on twitter sentiments and Shiny to your local folder. For the remainder of the class, let's work through the example in the .Rmd file to build a Shiny app that

1. Does a sentiment of analysis of limited tweets for a hashtag of user's choice and plots the distribution of the sentiments
2. Does a word frequency analysis and produces an informative plot
3. Has substantial reactive components that runs the code only after users action.