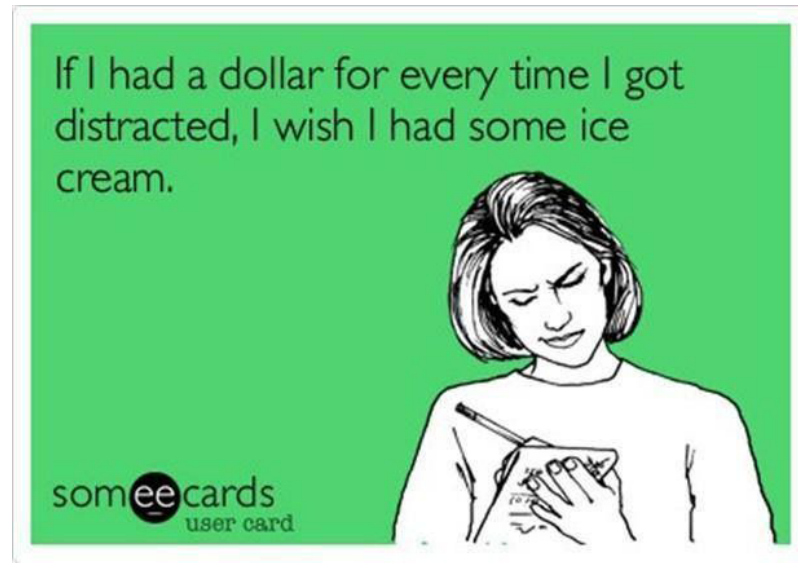


Oh Look! Shiny Objects

Get Prepared to Get Distracted by Shiny Things



This slideshow is adapted from the workshop *a gRadual intRoduction to Shiny*

What is Shiny?

- A web based framework for interactive visuals
- Developed by Joe Cheng and RStudio
- Server based: requires a basic installation of R and RStudio to work



Why Shiny?

- Interactive figures that help users explore data
- Dashboards for showing people summaries of data
- Widely Used (more than Tableau and Microsoft Power BI put together!)
- Leverages R and its visualization tools

But First: Functions

In order to use Shiny we need to have an understanding of R code syntax and how to use functions in R.

Functions take an argument - an *input* - and produces a result - an *output*

An Example

A simple function is the `mean()`. In R we can just write

```
mean(c(1,2,3,4,5))
```

to get 3.

Part 1: The Basic Shiny App Framework

Simple Shiny App Setup

app.R

```
library(shiny)
```

```
ui <- fluidPage(  
  )
```

→ Tells Shiny what to display on the webpage

```
server <- function(input,  
  output, session) {  
  }
```

→ Runs R code
Create outputs to be used in the UI

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

→ Spins up Shiny App

Credit: Vivian Peng

The (Bare) Minimal Shiny App

ui

```
ui <- fluidPage()
```

- note that `fluidPage` is a `function`
- uses `()`, so arguments need to be comma separated

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

server

```
server <- function(input, output) {}
```

- Note that `server` defines a new function
- Uses `{}` (curly brackets), so code is separated by line

input and output are how ui and server communicate

- `ui` and `server` are continuously running and listening to each other
- `ui`: listens to `output` and puts info into `input`
 - passes on information on state of controls into `input` (`input$my_slider`)
 - listens to `output` for generated plots and tables and changes
- `server`: listens to `input` and puts info into `output`
 - passes on plots and tables into `output` (`output$my_plot`)
 - listens to `input` for changes in controls

ggplot2movies dataset

```
## # A tibble: 58,788 x 24
##   title      year length budget rating votes   r1    r2
##   <chr>    <int> <int> <int> <dbl> <int> <dbl> <dbl>
## 1 $          1971   121   NA    6.4   348   4.5   4.5
## 2 $1000 a... 1939    71   NA    6     20    0    14.5
## 3 $21 a D... 1941     7   NA    8.2    5    0     0
## 4 $40,000    1996    70   NA    8.2    6   14.5    0
## 5 $50,000... 1975    71   NA    3.4   17   24.5   4.5
## 6 $pent     2000    91   NA    4.3   45   4.5   4.5
## 7 $windle   2002    93   NA    5.3  200   4.5    0
## 8 '15'      2002    25   NA    6.7   24   4.5   4.5
## 9 '38       1987    97   NA    6.6   18   4.5   4.5
## 10 '49-'17  1917    61   NA    6     51   4.5    0
## # ... with 58,778 more rows, and 16 more variables:
## #   r3 <dbl>, r4 <dbl>, r5 <dbl>, r6 <dbl>, r7 <dbl>,
## #   r8 <dbl>, r9 <dbl>, r10 <dbl>, mpaa <chr>,
## #   Action <int>, Animation <int>, Comedy <int>,
## #   Drama <int>, Documentary <int>, Romance <int>,
## #   Short <int>
```

`ggplot2movies` is essentially A tibble with 28819 rows and 24 variables with the following columns

- `title`. Title of the movie.
- `year`. Year of release.
- `budget`. Total budget (if known) in US dollars
- `length`. Length in minutes.
- `rating`. Average IMDB user rating.
- `votes`. Number of IMDB users who rated this movie.
- `r1-10`. Multiplying by ten gives percentile (to nearest 10%) of users who rated this movie a 1.
- `mpaa`. MPAA rating.
- `action, animation, comedy, drama, documentary, romance, short`. Binary variables representing if movie was classified as belonging to that genre.

Wrangling

```
movies_wrangled <- movies %>%
  na.omit() %>%
  mutate(budget = budget/1000000) %>%
  gather(key = genre, value, -c(title:mpaa)) %>%
  filter(!mpaa == "") %>%
  select(-value)

movies_by_decade <- movies_wrangled %>%
  mutate(year = case_when(
    year %in% 1930:1939 ~ "1930s",
    year %in% 1940:1949 ~ "1940s",
    year %in% 1950:1959 ~ "1950s",
    year %in% 1960:1969 ~ "1960s",
    year %in% 1970:1979 ~ "1970s",
    year %in% 1980:1989 ~ "1980s",
    year %in% 1990:1999 ~ "1990s",
    year %in% 2000:2009 ~ "2000s"
  )
)
```

```
movies_by_decade %>%  
  head()
```

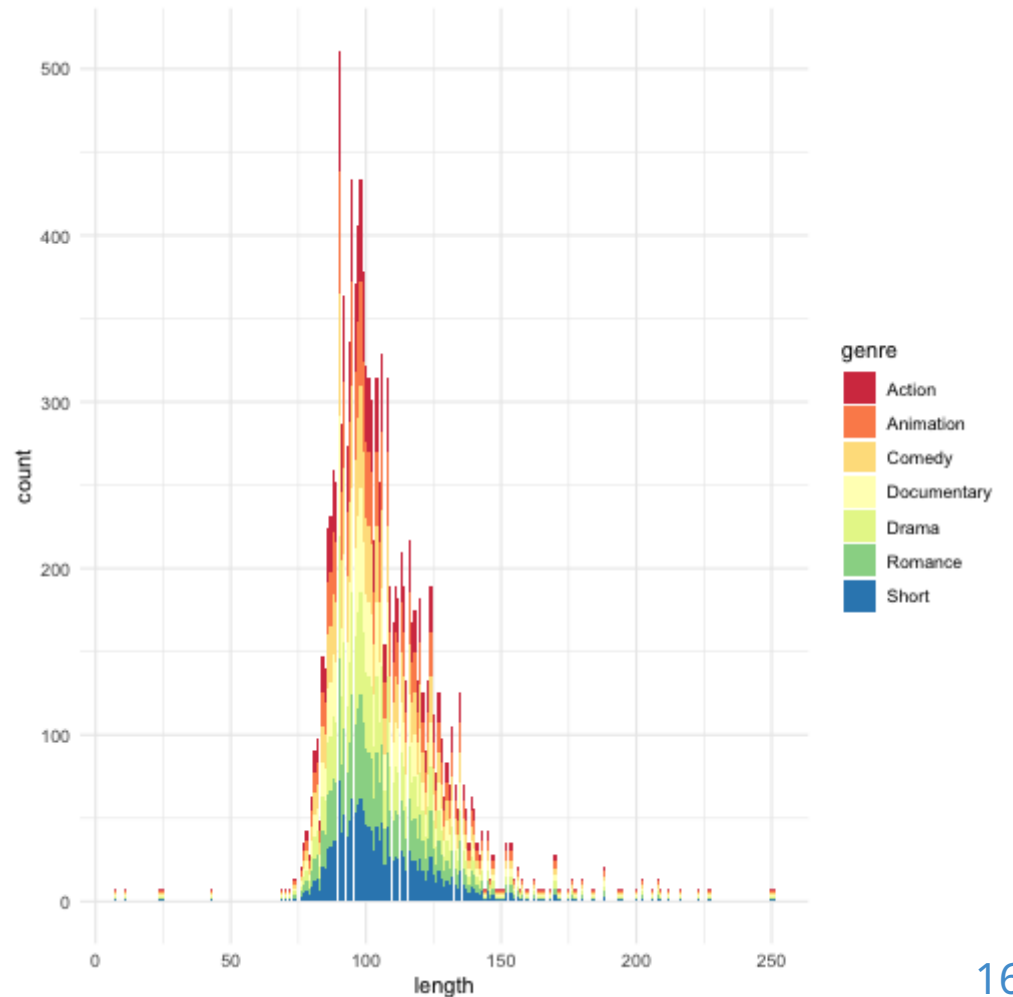
```
## # A tibble: 6 x 18  
##   title      year length budget rating votes   r1    r2  
##   <chr>      <chr> <int> <dbl> <dbl> <int> <dbl> <dbl>  
## 1 'Til The... 1990s   113    23     4.8   799   4.5   4.5  
## 2 10 Thing... 1990s    97    16     6.7 19095   4.5   4.5  
## 3 100 Mile... 2000s    98    1.1    5.6   181   4.5   4.5  
## 4 13 Going... 2000s    98    37     6.4  7859   4.5   4.5  
## 5 13th War... 1990s   102    85     6.1 14344   4.5   4.5  
## 6 15 Minut... 2000s   120    42     6.1 10866   4.5   4.5  
## # ... with 10 more variables: r3 <dbl>, r4 <dbl>,  
## #   r5 <dbl>, r6 <dbl>, r7 <dbl>, r8 <dbl>, r9 <dbl>,  
## #   r10 <dbl>, mpaa <chr>, genre <chr>
```

1.2 Adding a Plot to our App

Let's Add This Plot

```
movies_plot <- ggplot(movies_by_decade) +  
  aes_string(  
    x="length",  
    fill= "genre"  
  ) +  
  geom_bar() +  
  theme_minimal() +  
  scale_fill_brewer(palette =  
"Spectral")
```

- We use `aes_string()` instead of `aes()` because we can specify variables as `character` - such as `"year"`
- Will be helpful later when we add a control



Adding a plot: `plotOutput` and `renderPlot`

```
ui <- fluidPage(  
  plotOutput("movies_plot")  
)
```

- for `ui`, need to add a `plotOutput()` to display the plot
- note the argument `"movies_plot"`

```
server <- function(input, output) {  
  output$movies_plot <- renderPlot({  
  
  })  
}
```

- for `server`, need to add a `renderPlot()` to generate the plot
- assign into `output$movies_plot` so `ui` can display it

Adding our ggplot code

```
ui <- fluidPage(  
  plotOutput("movies_plot")  
)
```

```
server <- function(input, output) {  
  output$movies_plot <- renderPlot({  
    ggplot(movies_by_decade) +  
      aes_string(  
        x="length",  
        fill= "genre"  
      ) +  
      geom_bar() +  
      theme_minimal() +  
      scale_fill_brewer(palette =  
"Spectral")  
  })  
}
```

- Now we add our `ggplot()` statement in

Let's Add a Control

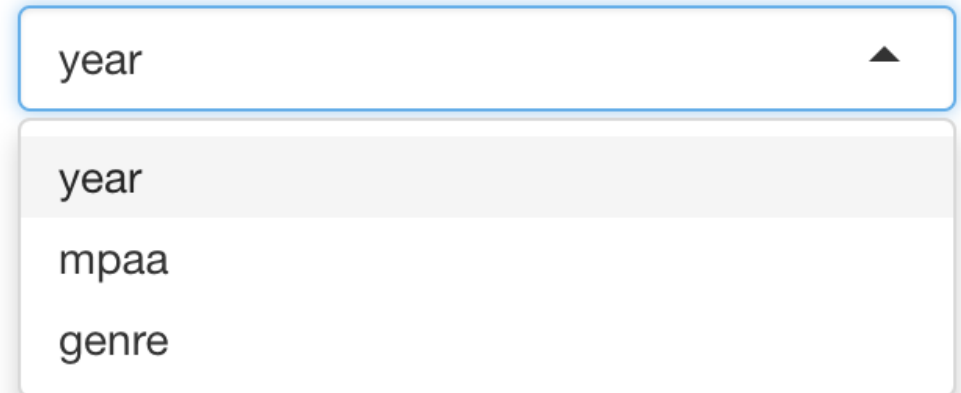
Define control options

```
categoricalVars <- c("year",  
                    "mpaa",  
                    "genre")
```

selectInput

```
selectInput(  
  inputId = "color_select",  
  label = "Select Categorical Variable",  
  choices = categoricalVars  
)
```

Select Categorical Variable



year

year

mpaa

genre

- Want to control the variable we **color** with the `selectInput()` control!

Adding the selectInput

```
ui <- fluidPage(  
  plotOutput("movies_plot"),  
  selectInput(  
    inputId = "color_select",  
    label = "Select Categorical Variable",  
    choices = categoricalVars)  
)
```

- Here we add the `selectInput()` control
- Note the comma after `plotOutput("movies_plot")`

```
server <- function(input, output) {  
  output$movies_plot <- renderPlot({  
    ggplot(movies_by_decade) +  
      aes_string(  
        x="length",  
        fill= "genre"  
      ) +  
      geom_bar() +  
      theme_minimal() +  
      scale_fill_brewer(palette  
= "Spectral")  
  })  
}
```

Wiring in the Input

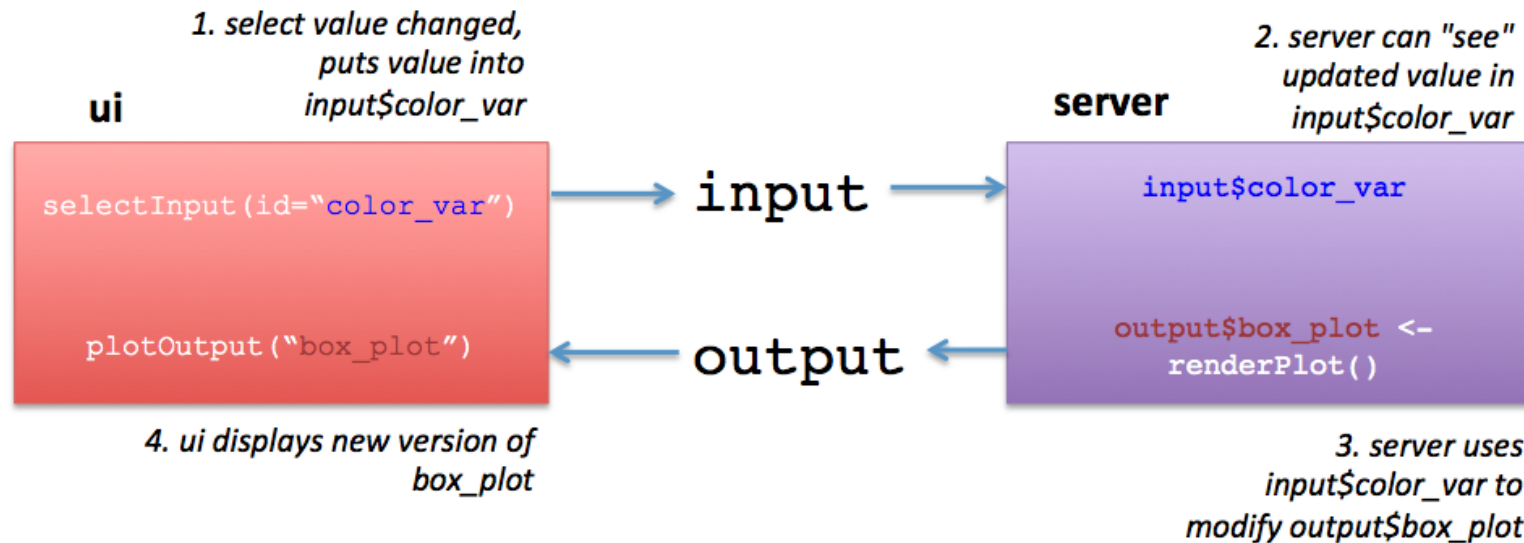
```
ui <- fluidPage(  
  plotOutput("movies_plot"),  
  selectInput(  
    inputId = "color_select",  
    label = "Select Categorical Variable",  
    choices = categoricalVars)  
)
```

```
server <- function(input, output) {  
  output$movie_plot <- renderPlot({  
    gplot(movies_by_decade) +  
      aes_string(  
        x="length",  
        fill="genre",  
        color=input$color_select  
      ) +  
      geom_bar() +  
      theme_minimal() +  
      scale_fill_brewer(palette  
= "Spectral")  
  })  
}
```

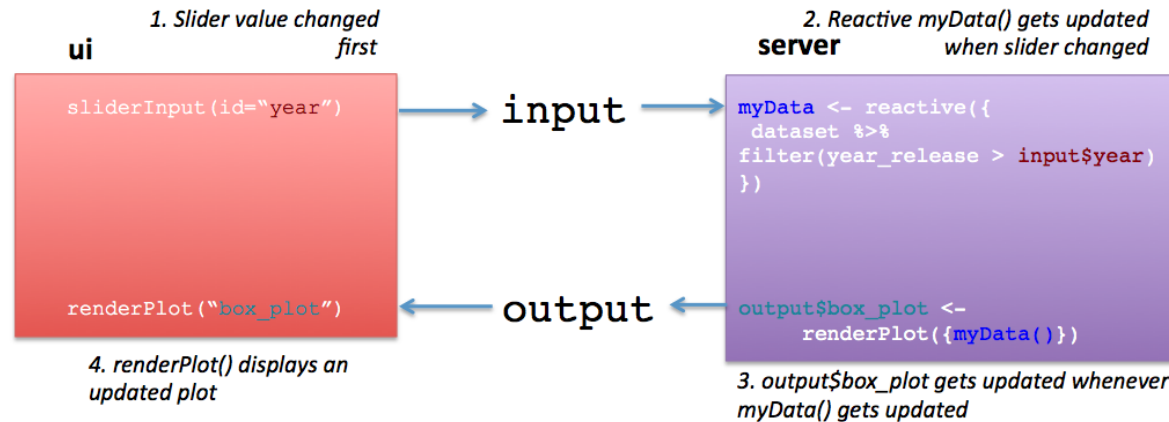
```
server <- function(input, output) {  
  output$movie_plot <- renderPlot({  
    gplot(movies_by_decade) +  
      aes_string(  
        x="length",  
        fill="genre",  
        color=input$color_select  
      ) +  
      geom_bar() +  
      theme_minimal() +  
      scale_fill_brewer(palette  
= "Spectral")  
  })  
}
```

- now we connect our `selectInput` to our `ggplot`
- use `input$color_select` as argument to `color` in `aes_string()`

The Flow: from `selectInput()` to `plotOutput()`



Part 2: Making Data Reactive



Making a Dataset Filterable

```
movies %>%  
  filter(year > 1979) %>%  
  head(n=3)
```

```
## # A tibble: 3 x 24  
##   title    year length budget rating votes   r1    r2  
##   <chr>  <int> <int> <int> <dbl> <int> <dbl> <dbl>  
## 1 $40,000  1996    70    NA    8.2     6  14.5    0  
## 2 $pent    2000    91    NA    4.3    45   4.5   4.5  
## 3 $windle 2002    93    NA    5.3   200   4.5    0  
## # ... with 16 more variables: r3 <dbl>, r4 <dbl>,  
## #   r5 <dbl>, r6 <dbl>, r7 <dbl>, r8 <dbl>, r9 <dbl>,  
## #   r10 <dbl>, mpaa <chr>, Action <int>,  
## #   Animation <int>, Comedy <int>, Drama <int>,  
## #   Documentary <int>, Romance <int>, Short <int>
```

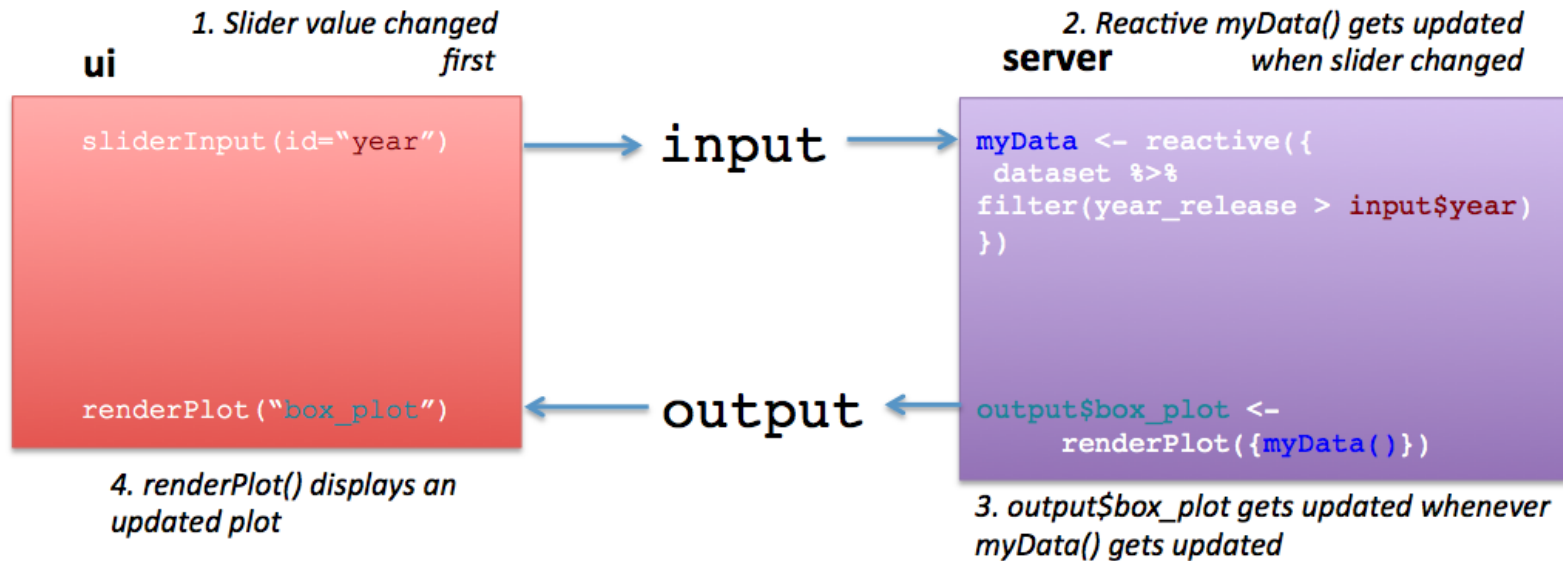
- We want to make this `filter()` statement dynamic
- Move a slider, and change the year
- We'll need to put it in a `reactive` expression

Making your data listen

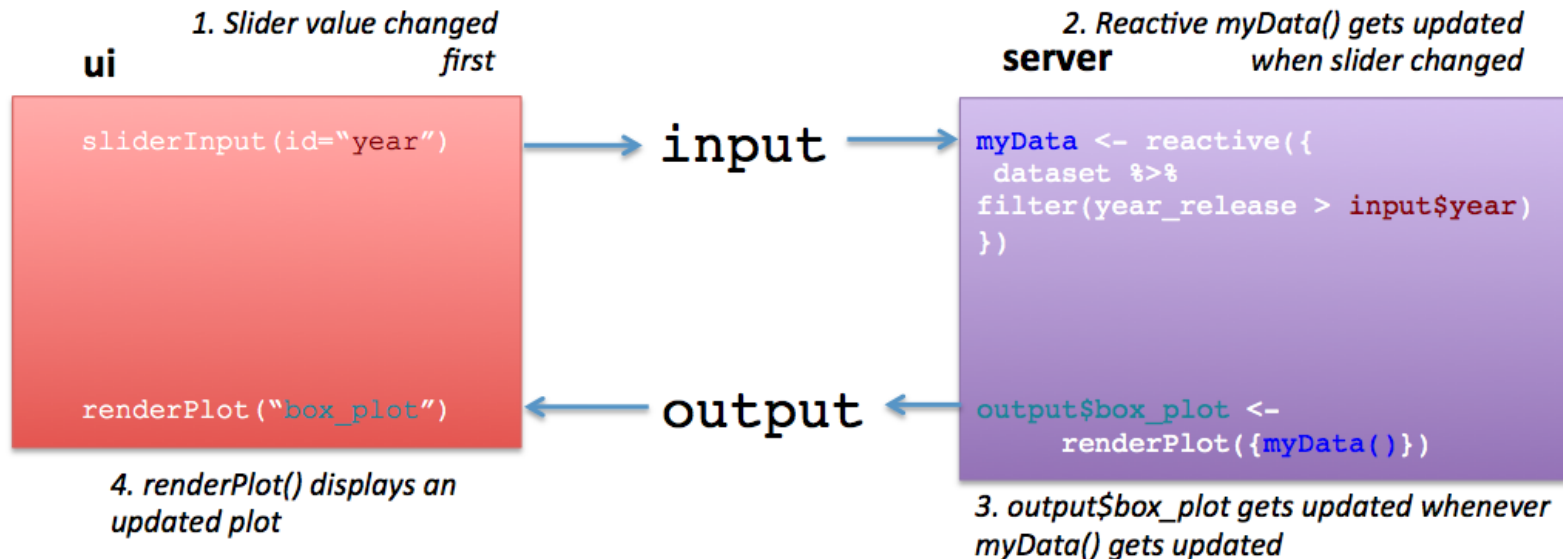
```
movies_filtered <- reactive({  
  movies %>%  
    filter(year > input$year)  
})
```

- **reactive** expressions listen to changes in `input`
- started with a `reactive({})`
- The curly brackets in `reactive({})` let us use more than one line of code `{}`

Reactive Flow: from slider to data to plot



Reactive Flow: from slider to data to plot



Adding our control: sliderInput()

```
sliderInput(inputId = "year",  
            "Select a Year",  
            min = min(movies),  
            max = max(movies),  
            value = 1970)
```

Adding sliderInput (in ui)

```
ui <- fluidPage(  
  plotOutput("movies_plot"),  
  sliderInput("year_filter",  
             "Select a Year",  
             min = 1893,  
             max = 2005,  
             value = 1970)  
)
```

- Don't forget the comma after `plotOutput("movies_plot")!`

Using our Reactive (in server)

```
movies_filtered <- reactive({  
  movies_wrangled %>%  
    filter(year >  
           input$year_filter)  
})
```

```
renderPlot({  
  
  output$movies_plot <-  
    ggplot(movies_filtered()) +  
      aes_string(x="year",  
                y="length") +  
      geom_bar(stat="identity")  
  
})
```

Plot

Take a look at `classthing.R` to see an output.

Some Tips

- Always call reactives with the ()
- Example: `movies_filtered()`

Part 3: Adding Tooltips with `plotly`

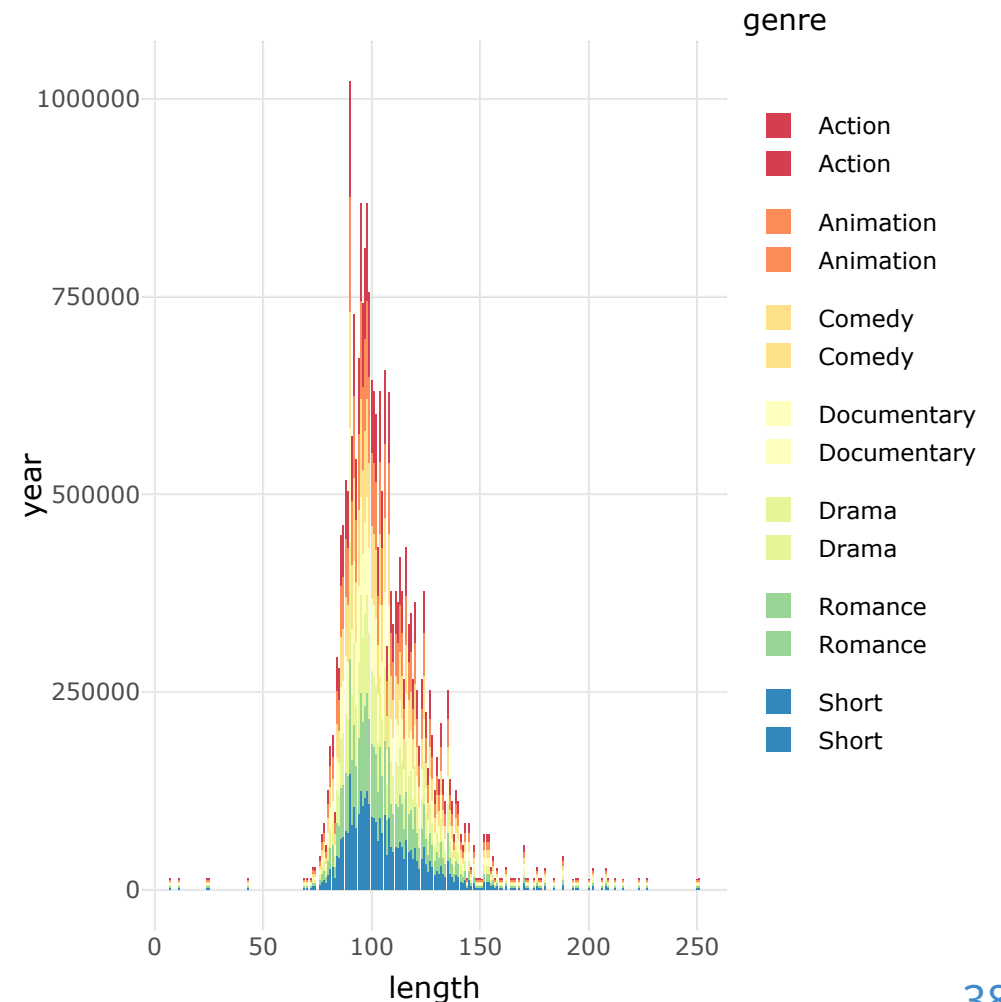
What is plotly?

- A JavaScript library that makes your interactive plots more interactive.
- accessed with the `plotly` package in R

Making a ggplot into a plotly plot

```
my_plot <- ggplot(movies_wrangled) +  
  aes_string(x = "length",  
            y = "year",  
            fill = "genre") +  
  geom_bar(stat = "identity") +  
  theme(legend.position="none") +  
  theme_minimal() +  
  scale_fill_brewer(palette = "Spectral")  
  
#ggplotly(my_plot)
```

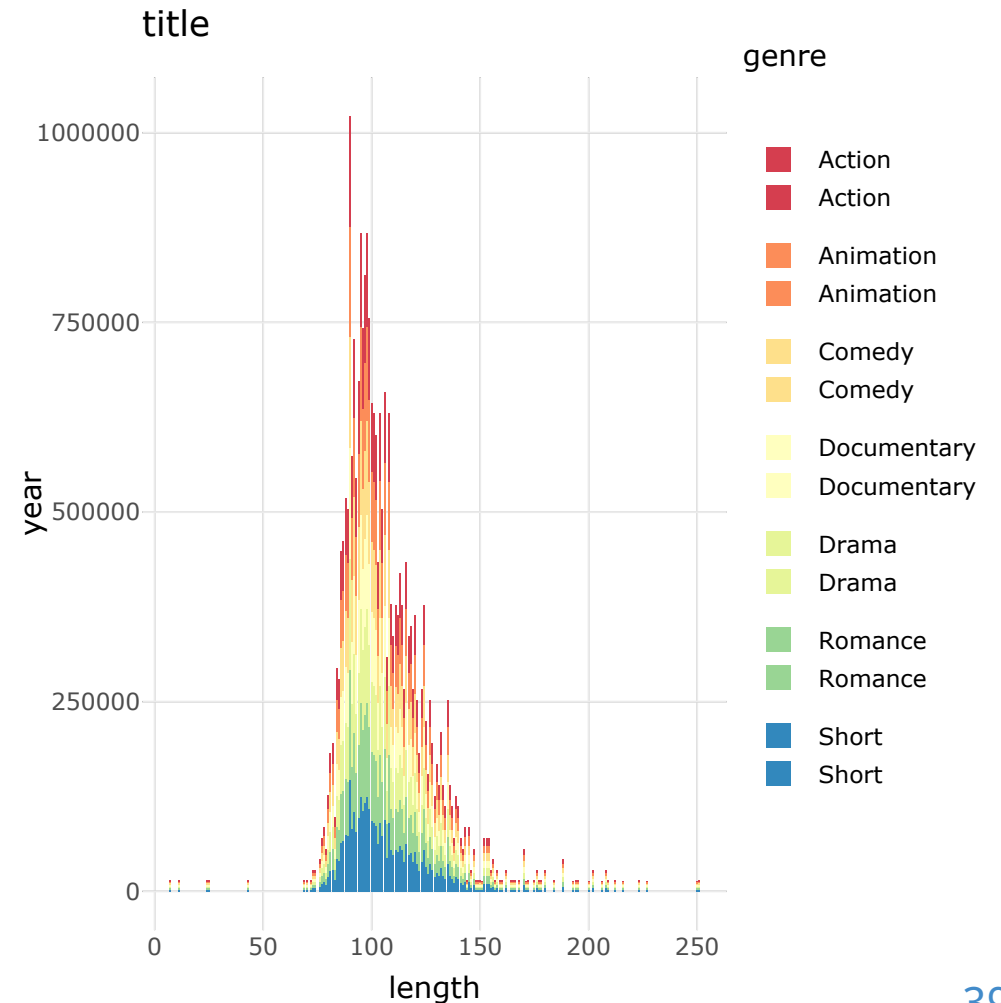
- assign our plot to `my_plot`
- run `ggplotly()` on `my_plot`



Adding more tooltip information

```
my_plot <- ggplot(movies_wrangled) +  
  aes_string(x = "length",  
            y = "year",  
            fill = "genre",  
            title = "title") +  
  geom_bar(stat = "identity") +  
  theme(legend.position="none") +  
  theme_minimal() +  
  scale_fill_brewer(palette = "Spectral")  
  
#ggplotly(my_plot)
```

- add to `aes_string()`



Adding to our app - make these changes

in ui:

Change

`plotOutput()`

to

`plotlyOutput()`

in server:

Change

`renderPlot()`

to

`renderPlotly()`

Modified App for plotly tooltips

```
ui <- fluidPage(  
  plotlyOutput("movies_plot"),  
  selectInput(inputId = "color_select",  
              label = "Select Categorical  
Variable",  
              choices = categoricalVars)  
)
```

```
server <- function(input, output) {  
  output$movies_plot <- renderPlotly({  
    my_plot <- ggplot(movies_wrangled) +  
      aes_string(x = "length",  
                y = "year",  
                fill = "genre") +  
      geom_bar(stat = "identity") +  
      theme(legend.position="none") +  
      theme_minimal() +  
      scale_fill_brewer(palette = "Spectral")  
  
    ggplotly(my_plot)  
  })  
}
```

app.R

We've been running Shiny apps as code blocks so far.

Apps are usually set up in a folder with `app.R`

Making a new app as a project

In a project, use

File > New Project > New Directory > Shiny Web Application

And then name your app.

Wrap up and Tips

More about inputs and outputs

Further reading on the different control inputs, and data output types here:

https://laderast.github.io/gradual_shiny/app-1-connecting-ui-and-server.html#more-about-inputs-and-outputs

Shiny Widget Gallery: <https://shiny.rstudio.com/gallery/widget-gallery.html>

Layouts

Ways to lay out elements of your application:

- `fluidPage` - <https://shiny.rstudio.com/articles/layout-guide.html>
- `flexdashboard` - <https://rmarkdown.rstudio.com/flexdashboard/>

Extensions

More info here: https://laderast.github.io/gradual_shiny/where-next.html

Shiny at WVU

- I am negotiating with RStudio to get an RStudio Connect server that is accessible to everyone
- Typically requires a developer's fee and user. This applies to all app rendering systems like Tableau and Microsoft BI

Shiny in the Real World

- shinyapps.io lets you host Shiny apps externally
 - sign up for an account
 - Be very careful about PHI

Deploying Apps on shinyapps.io

- Requires installing `{rsconnect}` package
- When you first try to deploy, it will ask you for your account info
- When you run the app, there is a "Publish" button

Shiny Gallery

You should now know enough to start learning from the examples:

- <https://shiny.rstudio.com/gallery/>
- Look at demos

Going Further

- Try to compute statistics ahead of time prior to any Shiny code
- Learn more about how to dynamically update the `ui`
- Look at [htmlwidgets](#) for possible JavaScript visualizations you can leverage

Suggested Reading

- [Mastering Shiny](#) by Hadley Wickham
- [Interactive web-based data visualization with R, plotly, and shiny](#) by Carson Sievert

Questions?