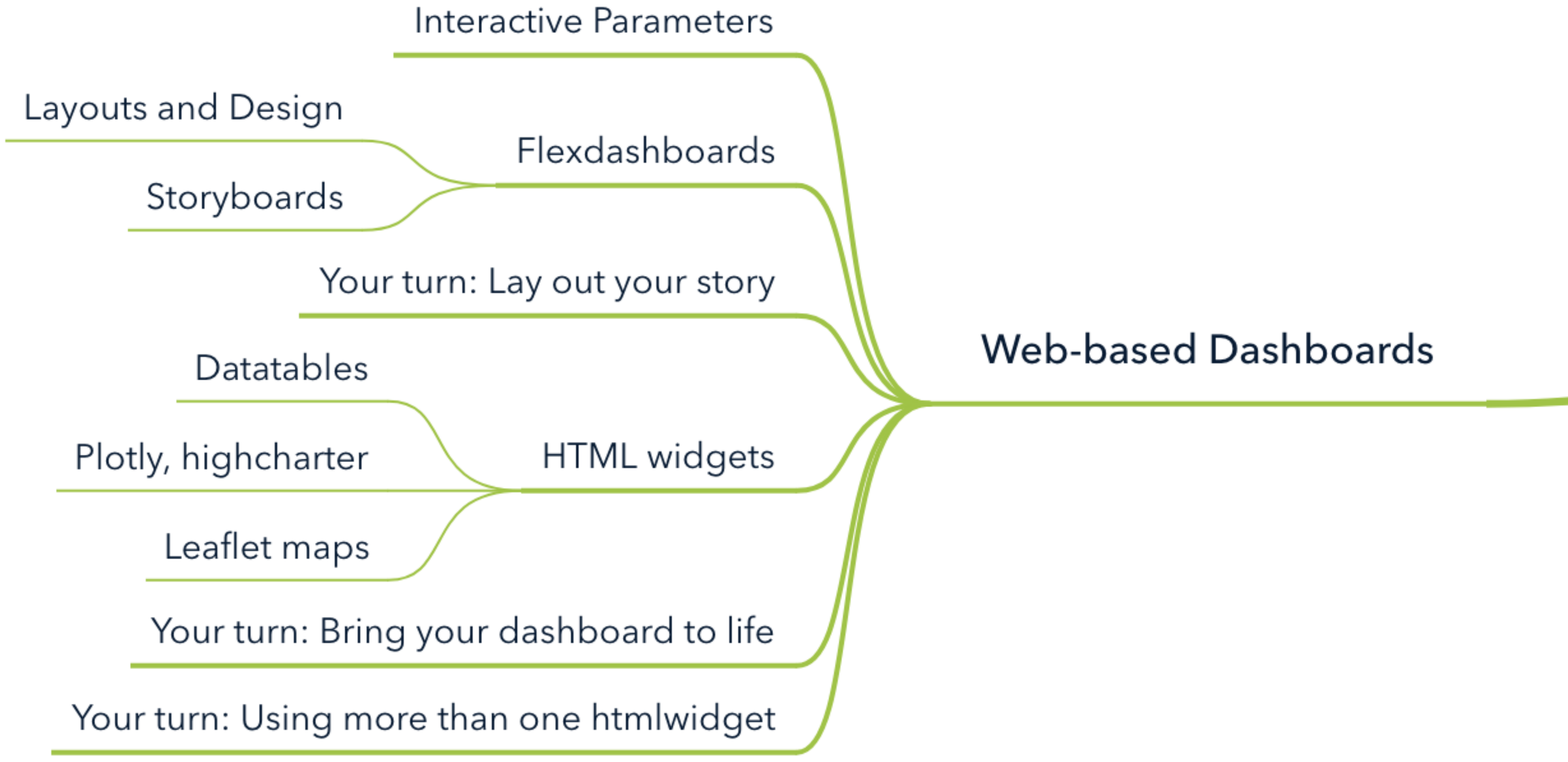

Web-based Dashboards
Interactive Parameters
Flexdashboards
Layouts and Design
Storyboards
Your turn: Lay out your story
HTML widgets
Datatables
Plotly, highcharter
Leaflet maps
Your turn: Bring your dashboard to life
Your turn: Using more than one htmlwidget


What is Shiny
How does Shiny work?
Shiny apps versus embedding Shiny in R Markdown
Input elements
App elements
Building out your app
Shiny Dashboards
Your turn: Your first Shiny dashboard
Next level Shiny: Reactivity
Your turn: Build a reactive dashboard

RStudio Education

education.rstudio.com

R Markdown and Interactive Dashboards Shiny

rstd.io/RMAID

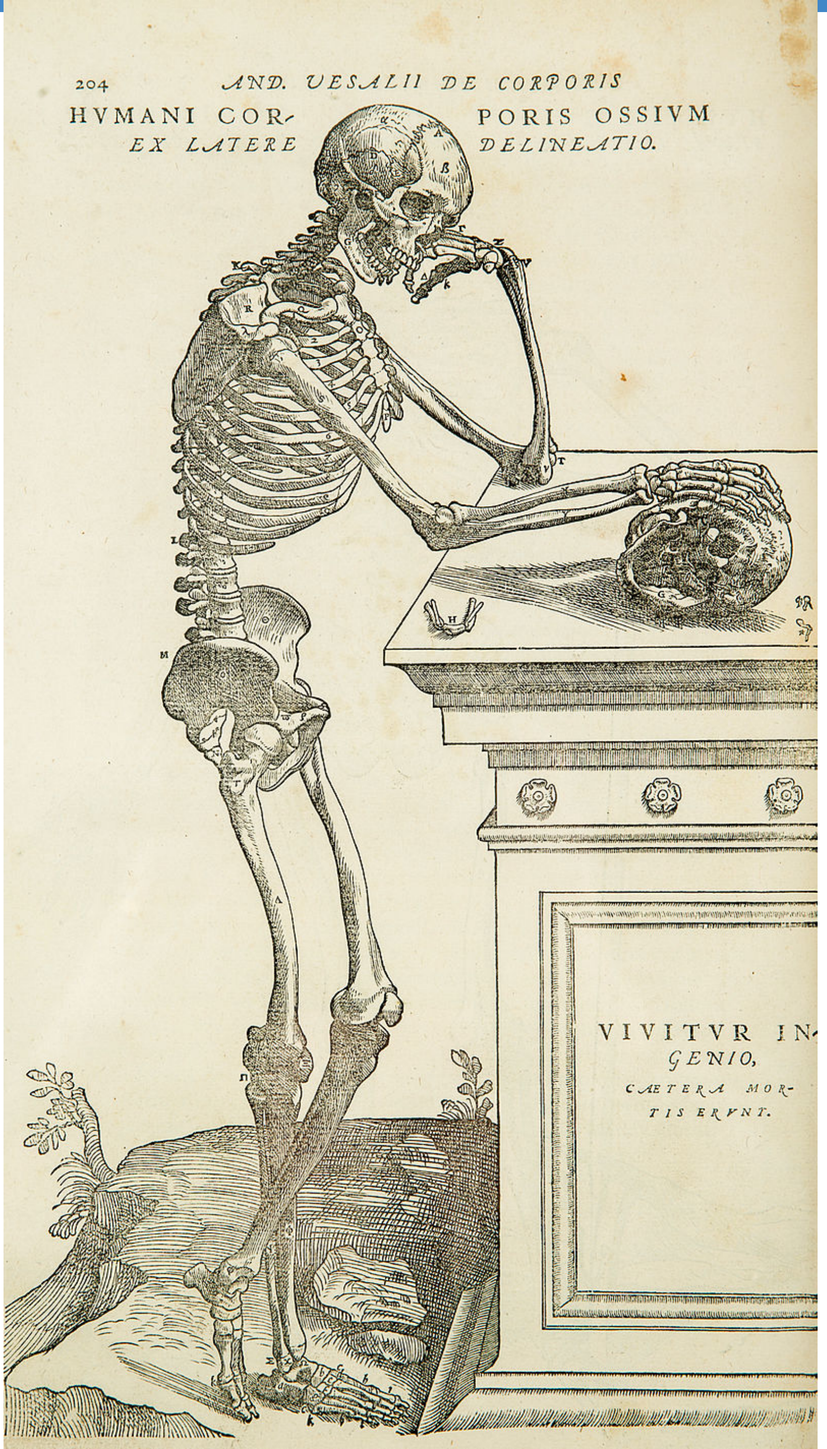
Shiny is an R package that makes it easy to build interactive web apps from R.

shiny.rstudio.com

In This Course We Teach A Subset of Shiny

- Shiny has all the mechanisms to create your own app with a fully custom UI
- In this course, we're teaching the essential inputs and outputs that will working within a flexdashboard layout
- There's much much more to Shiny if you wish to explore it
- Learn more at shiny.rstudio.com
- The Shiny cheat sheet lives at

  https://github.com/rstudio/cheatsheets/raw/master/shiny.pdf

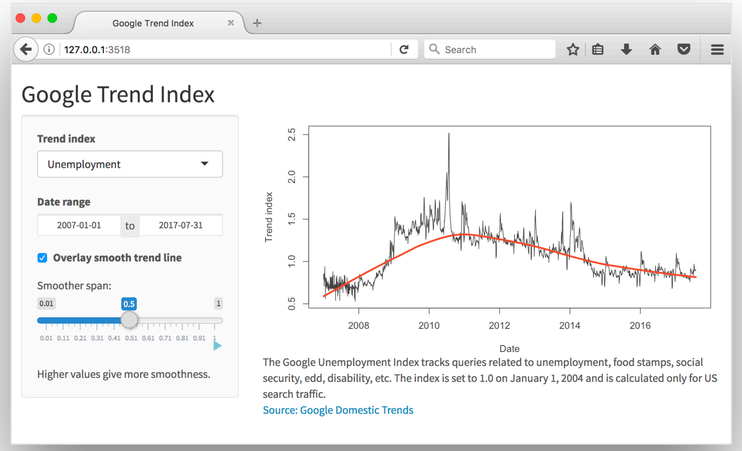


The Anatomy Of A Shiny App Made With Flexdashboard

You'll learn

- What in a Shiny flexdashboard app
- How to build Shiny inputs for your dashboard
- How to render differing types of output

rstd.io/RMAID



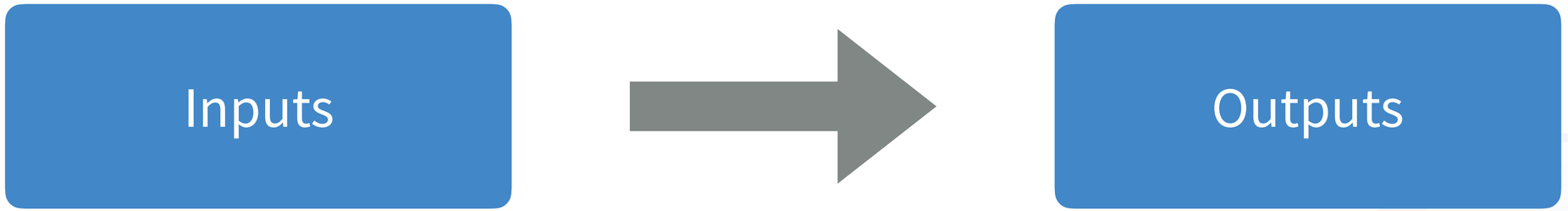
Server instructions

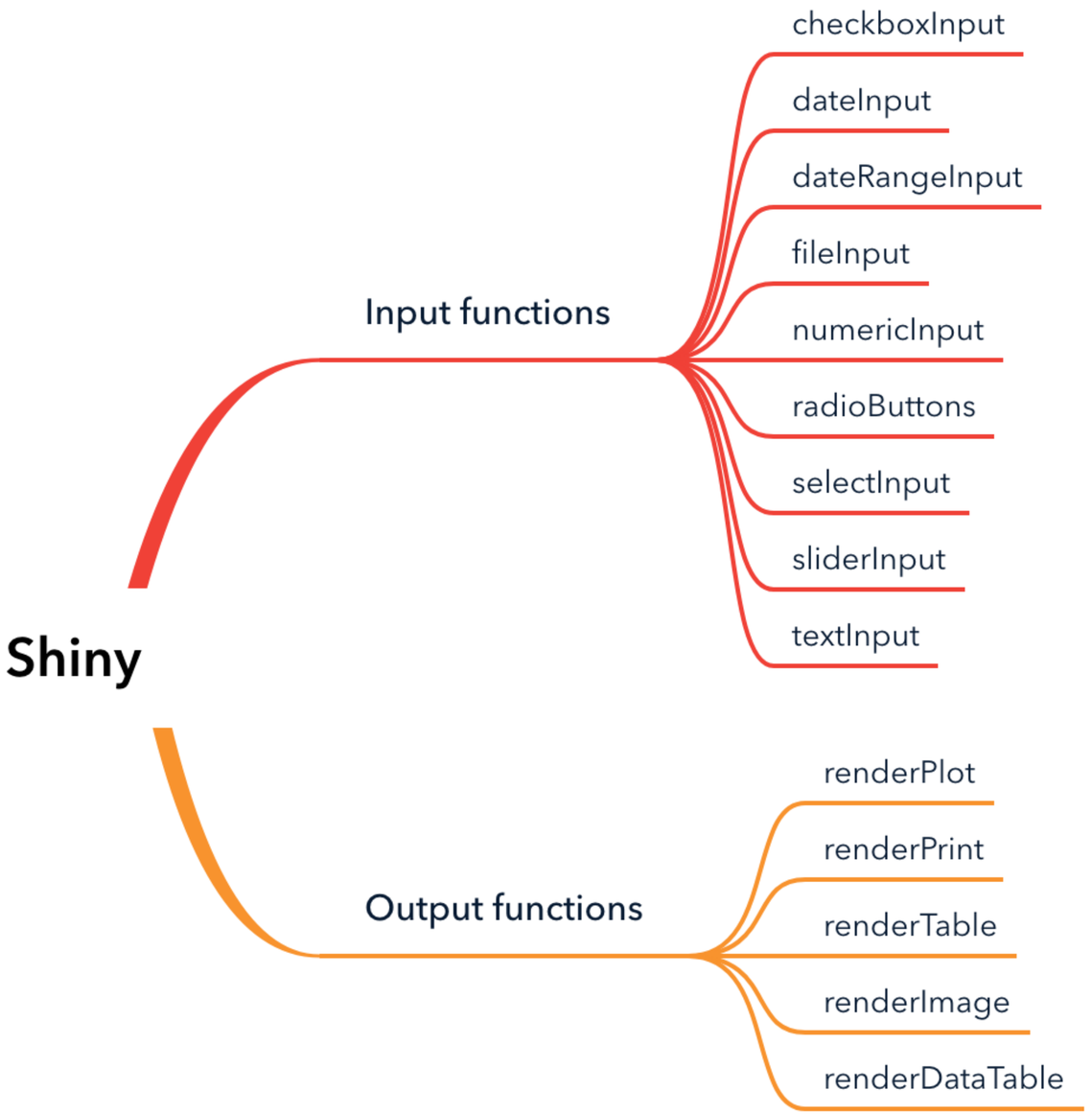
server.R

User interface

ui.R

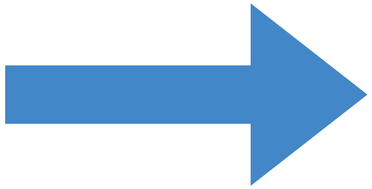
Shiny offers user interface and rendering functions for input and output controls.

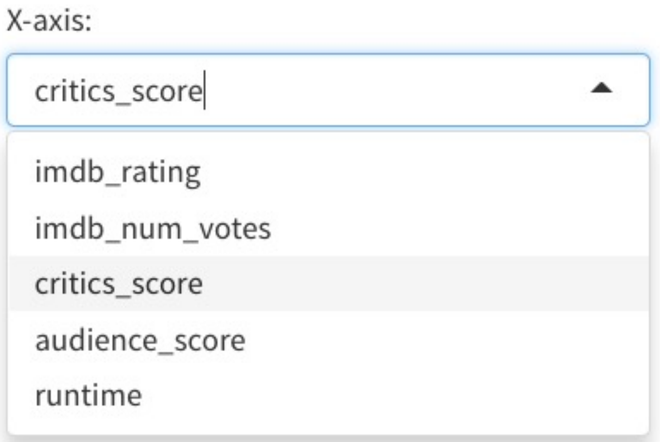

Inputs
Outputs


Shiny
Input functions
checkboxInput
dateInput
dateRangeInput
fileInput
numericInput
radioButtons
selectInput
sliderInput
textInput
Output functions
renderPlot
renderPrint
renderTable
renderImage
renderDataTable

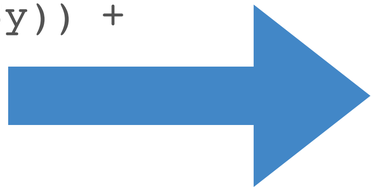
Let's Start With selectInput and renderPlot

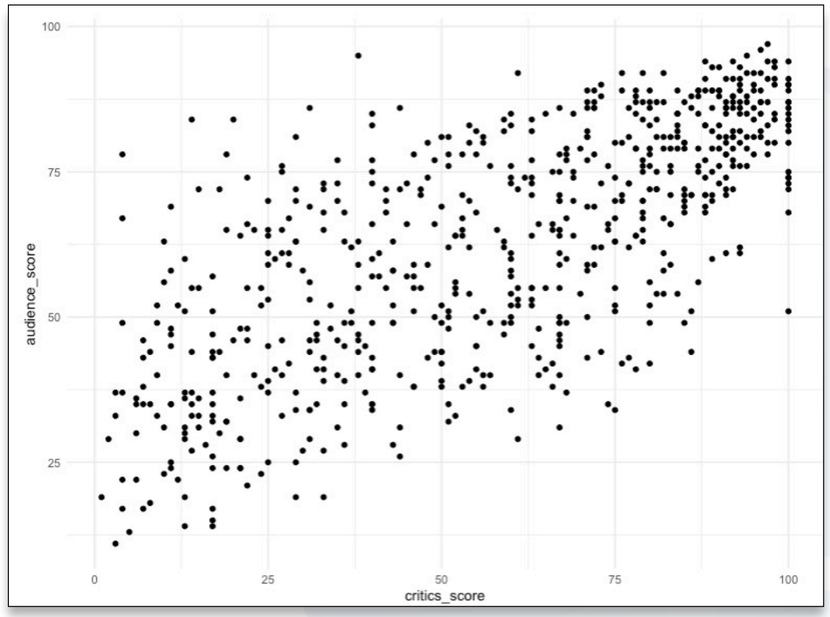
```
selectInput(inputId = "x",
            label = "X-axis:",
            choices = choices,
            selected = "critics_score")
```






```
renderPlot({
    ggplot(data = movies,
           aes_string(x = input$x, y = input$y)) +
      geom_point()
  })
```





rstd.io/RMAID

Basic Shiny Dashboard Structure

```
---
title: "Movies Dashboard with Shiny Inputs"
output:
  flexdashboard::flex_dashboard:
    orientation: columns
    vertical_layout: fill
  runtime: shiny
---
```

```
```{r setup, include=FALSE}
library(flexdashboard)
library(shiny)
library(ggplot2)
load("data/movies.Rdata")
theme_set(theme_minimal())
```
```

```
## Column {data-width=350}

### Inputs
```

```
```{r}
choices <- c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime")

    # Select variable for x-axis ---------------------------------
    selectInput(inputId = "x",    # this is the name of a reactive input variable
                label = "X-axis:",
                choices = choices,
                selected = "critics_score")

     ### Add another selectInput function for the y axis.

```
```

```
## Column {data-width=550}

### Output
```

```
```{r}
renderPlot({
    ggplot(data = movies, aes_string(x = input$x, y = input$y)) +
      geom_point()
  })
```
```

Flexdashboard with columns orientation

rstd.io/RMAID

Basic Shiny Dashboard Structure

```
---
title: "Movies Dashboard with Shiny Inputs"
output:
  flexdashboard::flex_dashboard:
    orientation: columns
    vertical_layout: fill
  runtime: shiny
---

```{r setup, include=FALSE}
library(flexdashboard)
library(shiny)
library(ggplot2)
load("data/movies.Rdata")
theme_set(theme_minimal())
```

Column {data-width=350}

Inputs

```{r}
choices <- c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime")

      # Select variable for x-axis ---------------------------------
      selectInput(inputId = "x",     # this is the name of a reactive input variable
                  label = "X-axis:",
                  choices = choices,
                  selected = "critics_score")

       ### Add another selectInput function for the y axis.

```

Column {data-width=550}

Output

```{r}
renderPlot({
    ggplot(data = movies, aes_string(x = input$x, y = input$y)) +
      geom_point()
  })
```
```

Inputs get created by a Shiny function like `selectInput`.

They appear as variables like `input$x`

rstd.io/RMAID

Basic Shiny Dashboard Structure

```
---
title: "Movies Dashboard with Shiny Inputs"
output:
  flexdashboard::flex_dashboard:
    orientation: columns
    vertical_layout: fill
  runtime: shiny
---

```{r setup, include=FALSE}
library(flexdashboard)
library(shiny)
library(ggplot2)
load("data/movies.Rdata")
theme_set(theme_minimal())
```

Column {data-width=350}

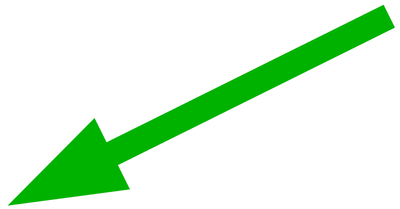
Inputs

```{r}
choices <- c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime")

      # Select variable for x-axis ---------------------------------
      selectInput(inputId = "x",     # this is the name of a reactive input variable
                  label = "X-axis:",
                  choices = choices,
                  selected = "critics_score")

       ### Add another selectInput function for the y axis.

```

Column {data-width=550}

Output

```{r}
renderPlot({
    ggplot(data = movies, aes_string(x = input$x, y = input$y)) +
      geom_point()
  })
```
```



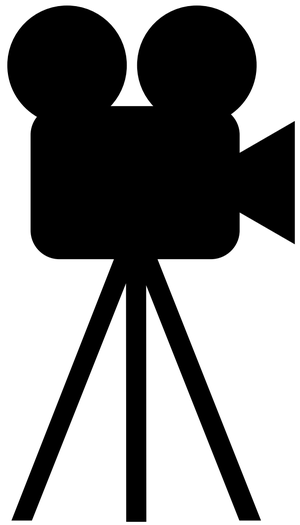
Output usually gets put in another flexdashboard pane using a function like `renderPlot`.

rstd.io/RMAID

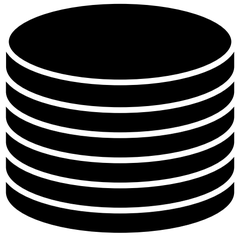
What Is This Syntax About?

```
renderPlot({
    ggplot(data = movies,
           aes_string(x = input$x,
                      y = input$y)) +
    geom_point()
})
```

- renderPlot takes an ggplot output object that generates a plot.
- The code inside the {} is code generates that output object, but it doesn't get evaluated until renderPlot asks it to.
- The funny syntax of `renderPlot({plotting_code})` is simply a way to say, "When you render the plot, use plotting_code to do so".



Let’s build a simple movie browser app!



data/movies.Rdata

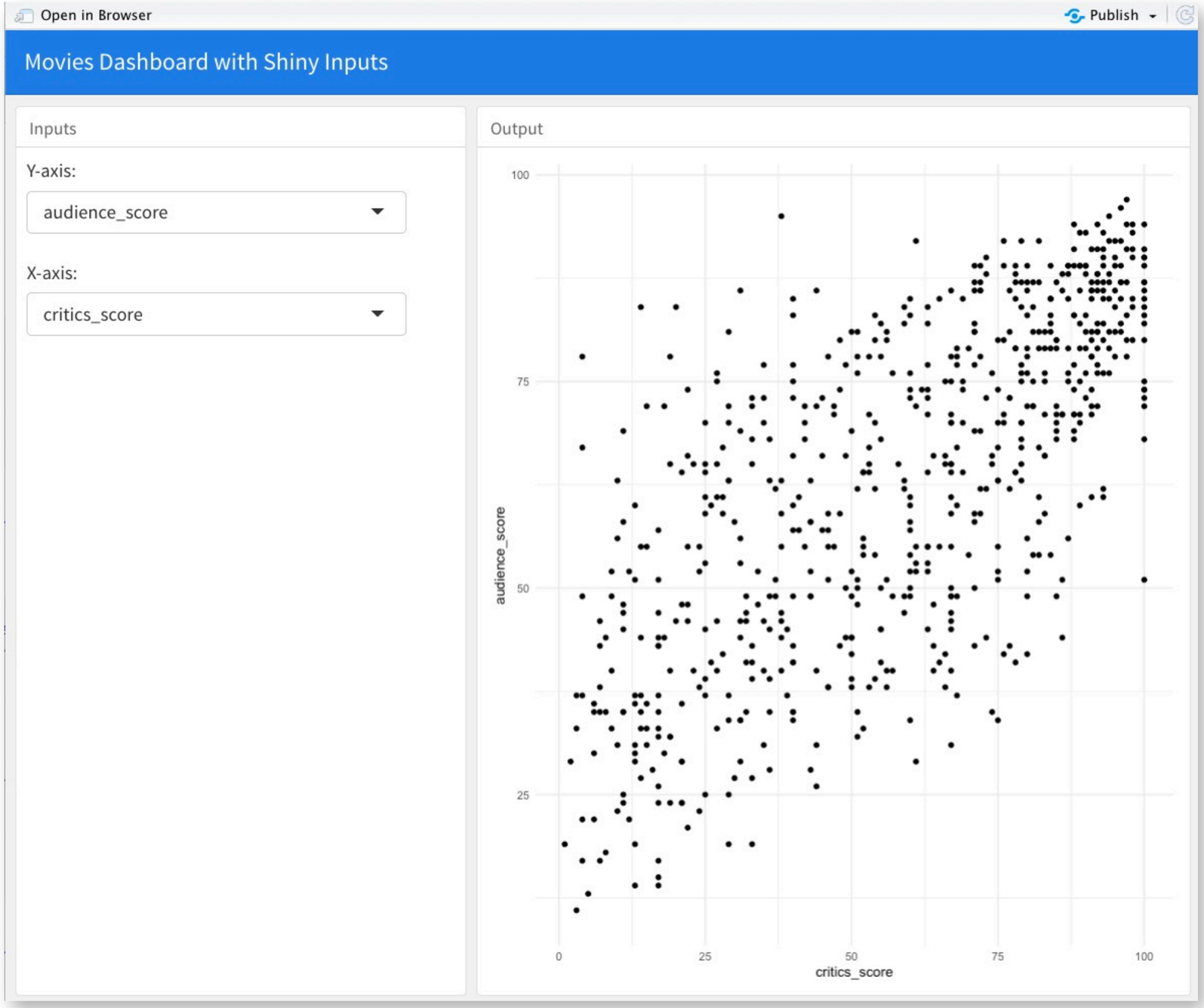
Data from IMDB and Rotten Tomatoes on random sample of 651 movies released in the US between 1970 and 2014

RStudio Cloud

Exercise 51

Build A Movie Brower

1. Open the project **05-RMAID-shiny**
2. Click on **`51-movies1.Rmd`** in the Files pane to open that file
3. Add in a second **`selectInput`** function to set the variable **`y`**
4. Click on *Run App*. Note that *Knit* isn't available.



rstd.io/RMAID

RStudio Cloud

Exercise 52

Add Color

1. Using the same file, now add a **`selectInput`** function to set the variable **`z`**.
2. Make the choices for z be **`c("title_type", "genre", "mpaa_rating", "critics_rating", "audience_rating")`**
3. Modify the ggplot routine to add an aesthetic value **`color=z`**.
4. Run your app.

5m 00s

Movies Dashboard with Shiny Inputs

Inputs

Y-axis:

audience_score

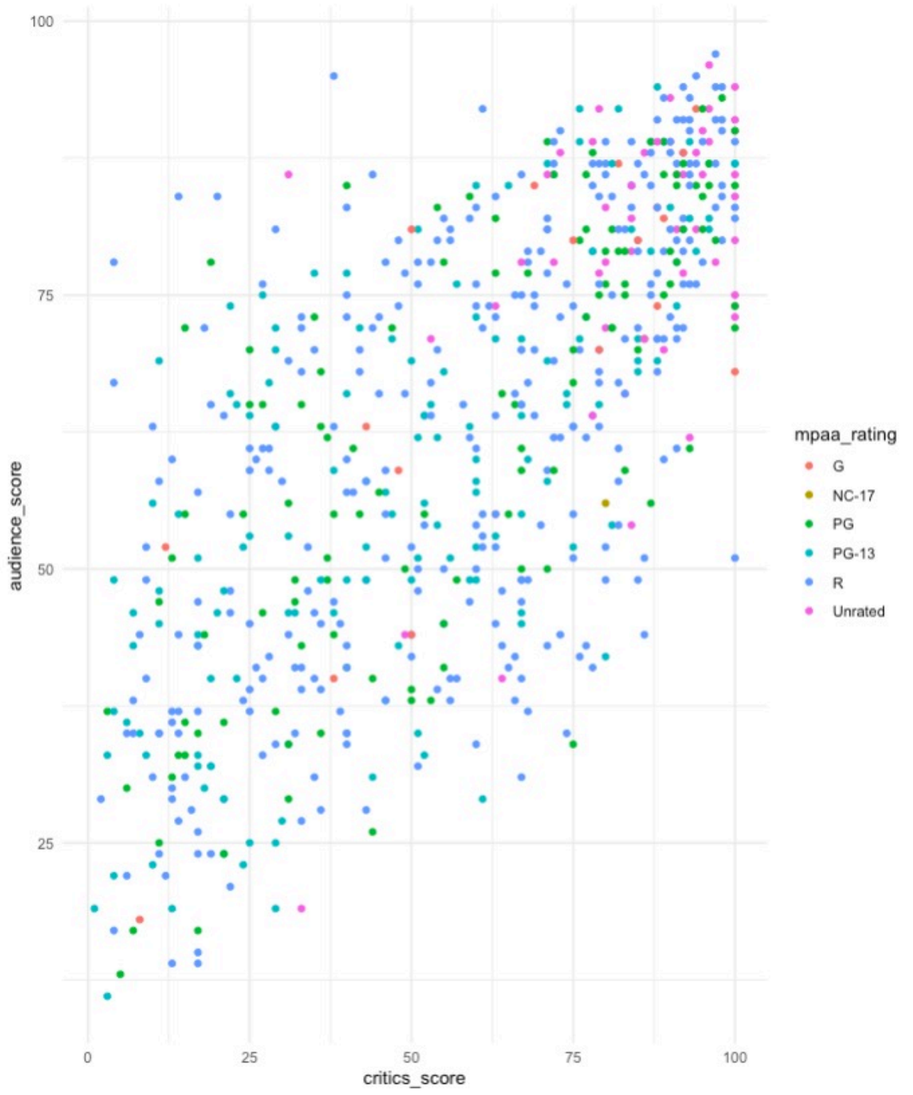
X-axis:

critics_score

Color by:

mpaa_rating

Output



rstd.io/RMAID

RStudio Cloud

Exercise 53

Add Transparency

1. Using the same file, now add a **`selectInput`** function to set the variable **`alpha`**.
2. 3. Modify the ggplot chunk to add an aesthetic value **`alpha=alpha.`**
4. Run your app.

5m 00s

Movies Dashboard with Shiny Inputs

Inputs

Y-axis:

audience_score

X-axis:

critics_score

Color by:

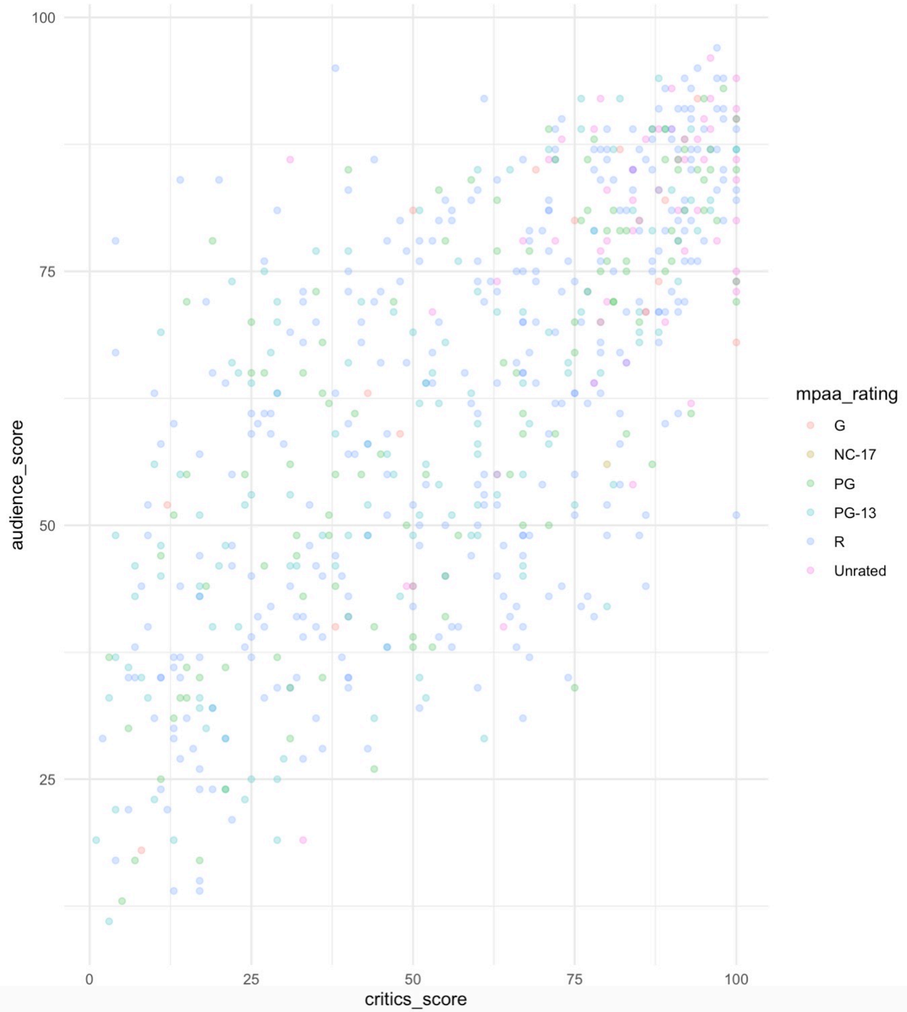
mpaa_rating

Alpha:

0 0.25 1

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Output



rstd.io/RMAID

RStudio Cloud

Exercise 54

Add DataTable Output

1. Open file **`54-movies-4.Rmd`**
2. Add another RMarkdown pane below the plot using a level 3 header (### something)
3. Within that pane, add a renderDataTable for movies using the following code:

```
DT::renderDataTable({
    DT::datatable(data = movies %>% select(selected_columns),
                  options = list(pageLength = 10,
                                 rownames = FALSE))
})
```

5. Run your app.

5m 00s

Movies Dashboard with Shiny Inputs

Inputs

Y-axis:

audience_score

X-axis:

critics_score

Color by:

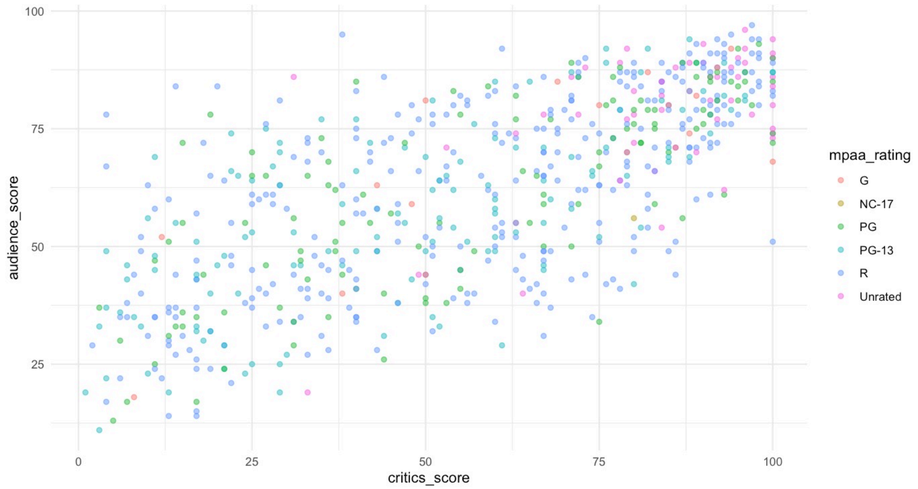
mpaa_rating

Alpha:

0 0.5 1

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Output Plot



Output Data Table

Show 10 entries Search:

	imdb_rating	critics_score	audience_score	runtime	title_type	genre
1	5.5	45	73	80	Feature Film	Drama
2	7.3	96	81	101	Feature Film	Drama
3	7.6	91	91	84	Feature Film	Comedy
4	7.2	80	76	139	Feature Film	Drama
5	5.1	33	27	90	Feature Film	Horror
6	7.8	91	86	78	Documentary	Documentar
7	7.2	57	76	142	Feature Film	Drama
8	5.5	17	47	93	Feature Film	Drama

RStudio Cloud

Exercise 55

Filtering Results

1. Open file **`55-movies5-doesnt-work.Rmd`**
2. Examine the code we added below ### $$$.
3. Run the app and see what happens.

5m 00s

What is

```
Error in .getReactiveEnvironment()
$currentContext: Operation not allowed
without an active reactive context. (You
tried to do something that can only be
done from inside a reactive expression or
observer.)?
```

Shiny inputs drive outputs through ***reactivity***

This Code Doesn't Work Because....

```
# Select genre for filtering
      selectInput(inputId = "genre_selection",
                  label = "Filter by genre:",
                  choices = movies %>% select(genre) %>% unique(),
                  selected = "mpaa_rating")

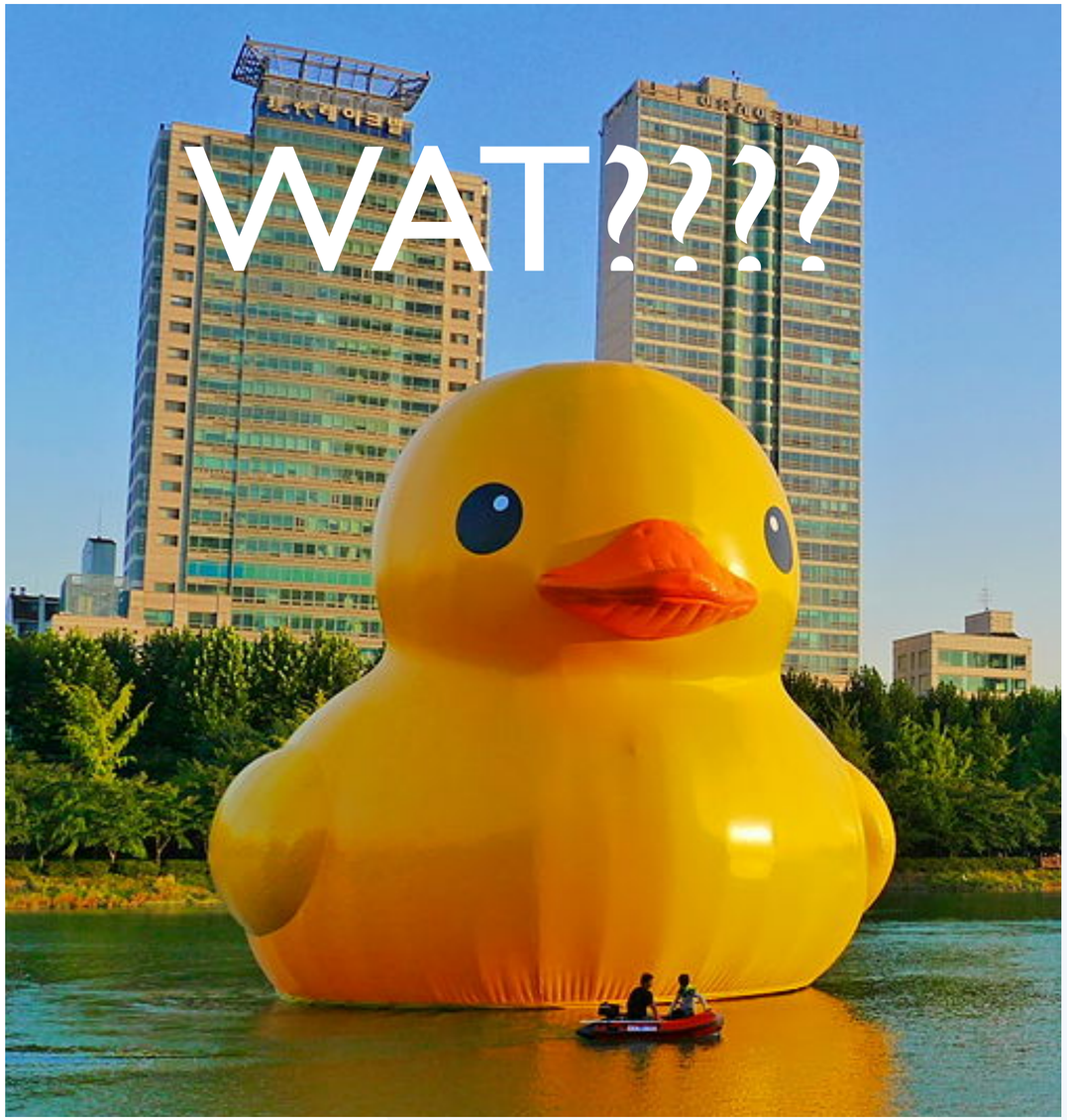

filtered_movies <- movies %>%
  filter(genre == input$genre_selection)
```

`input$genre_selection` isn't an ordinary variable

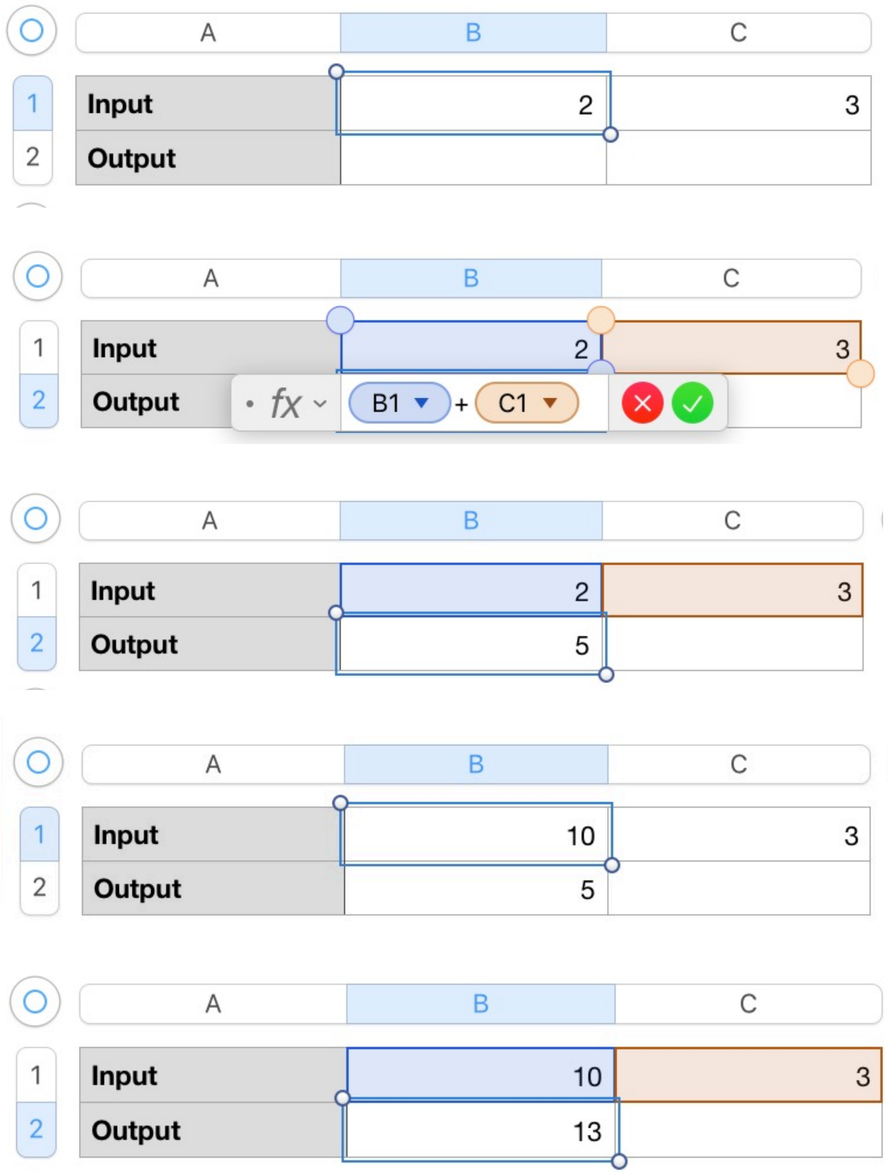
it's a reactive input that can change value with time

Reactivity

- Shiny input results of the form `input$variable` are reactive variables
- `input` variables can be used directly inside `render` expressions
- Any other uses of `input` variables must be inside a `reactive({})` expression
- Reactive expressions yield functions that are reactive when called as functions.



Think Of A Spreadsheet



- The value 13 was computed by **`reacting`** to B1 changing from 2 to 10
- All cells in spreadsheets are reactive by default
- In Shiny, only input variables created by Shiny input functions are reactive by default

We Tell Shiny About Reactivity With The `reactive` Function

We have to change this:

```
filtered_movies <- movies %>%
  filter(genre == input$genre_selection)
```

To this:

```
reactive_movies <- reactive({ movies %>%
  filter(genre == input$genre_selection)
    })
```

And when we want the value of reactive_movies, we have to invoke a function instead of just referencing it:

```
    ggplot(data = reactive_movies(), ...
```

RStudio Cloud

Exercise 55B

Filtering Results

1. Open file **`55-movies5.Rmd`**
2. Change **`filtered_movies`** to a new reactive expression, **`reactive_movies`** where $$$ appears
3. Also change references to **`filtered_movies`** to **`reactive_movies()`**
4. Run the app and see what happens.

Movies Dashboard with Shiny Inputs

Inputs

Y-axis:

audience_score

X-axis:

critics_score

Color by:

title_type

Alpha:

0 0.71 1

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Size:

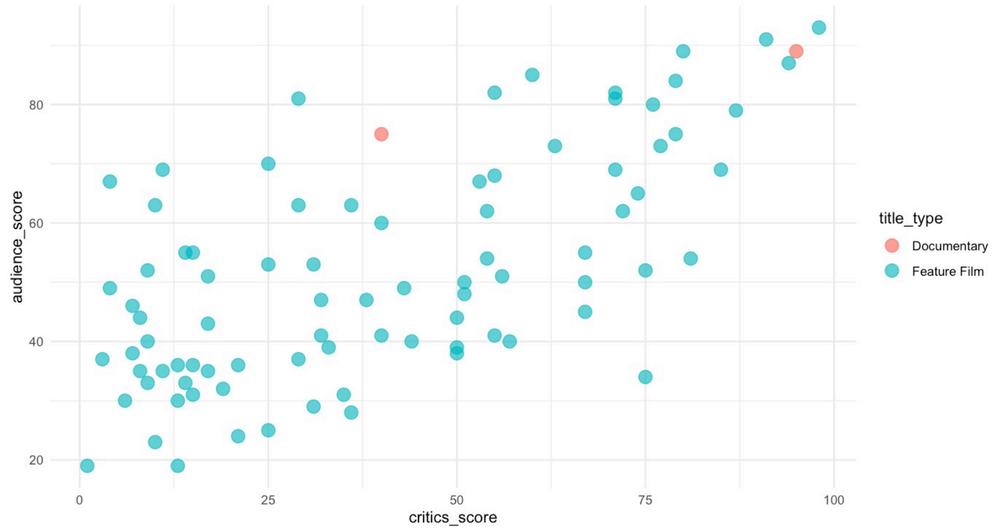
1 4 10

1 2 3 4 5 6 7 8 9 10

Filter by genre:

Comedy

Output Plot



Output Data Table

Show 10 entries Search:

	title	title_type	genre	runtime	mpaa_rating	studio	thtr_rel_
1	Waiting for Guffman	Feature Film	Comedy	84	R	Sony Pictures Classics	
2	The Royal Tenenbaums	Feature Film	Comedy	110	R	Buena Vista Distribution Compa	
3	School for Scoundrels	Feature Film	Comedy	100	PG-13	MGM	
4	Rhinestone	Feature Film	Comedy	111	PG	20th Century Fox	

rstd.io/RMAID

Publishing Shiny apps requires a server that runs R. shinyapps.io and RStudio Connect are two easy options

Publishing Shiny Apps

- Unlike HTML widgets that just rely on HTML and Javascript, Shiny apps must be published to a server that can run R.
- RStudio runs a public server called shinyapps.io that you can publish Shiny apps to.
- To publish your app so others can use it, hit the *Publish* button in the IDE.
- Create an account on shinyapps.io when asked and publish your app there.
- Note that shinyapps.io will only allow you to run 5 apps without paying, so be sure to kill of apps when you no longer want them.

Shiny Tips and Tricks

- Debugging Shiny apps can be VERY challenging because of their real-time nature
  1. Start by writing a static plotting application without Shiny
  2. Build out your UI in R Markdown without Shiny
  3. Encapsulate your application and UI within Shiny functions

rstd.io/RMAID

You can use HTML Widgets in your Shiny app by modifying your render functions

Render Functions For 2 HTML Widgets

If you want to use...	render using...	Instead of....
Plotly plots	renderPlotly	renderPlot
Leaflet maps	renderLeaflet	making your own maps

RStudio Cloud

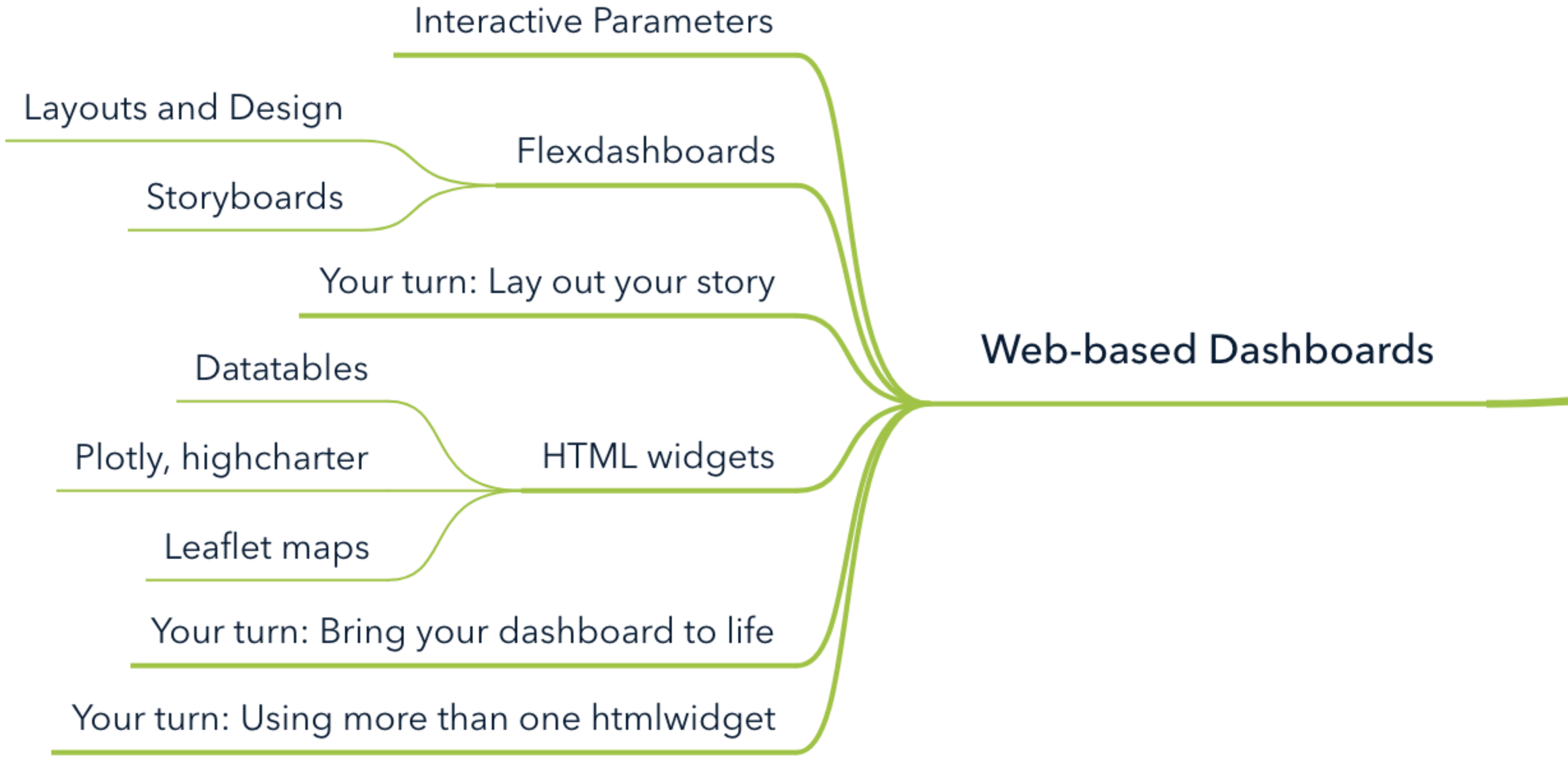
Exercise 46

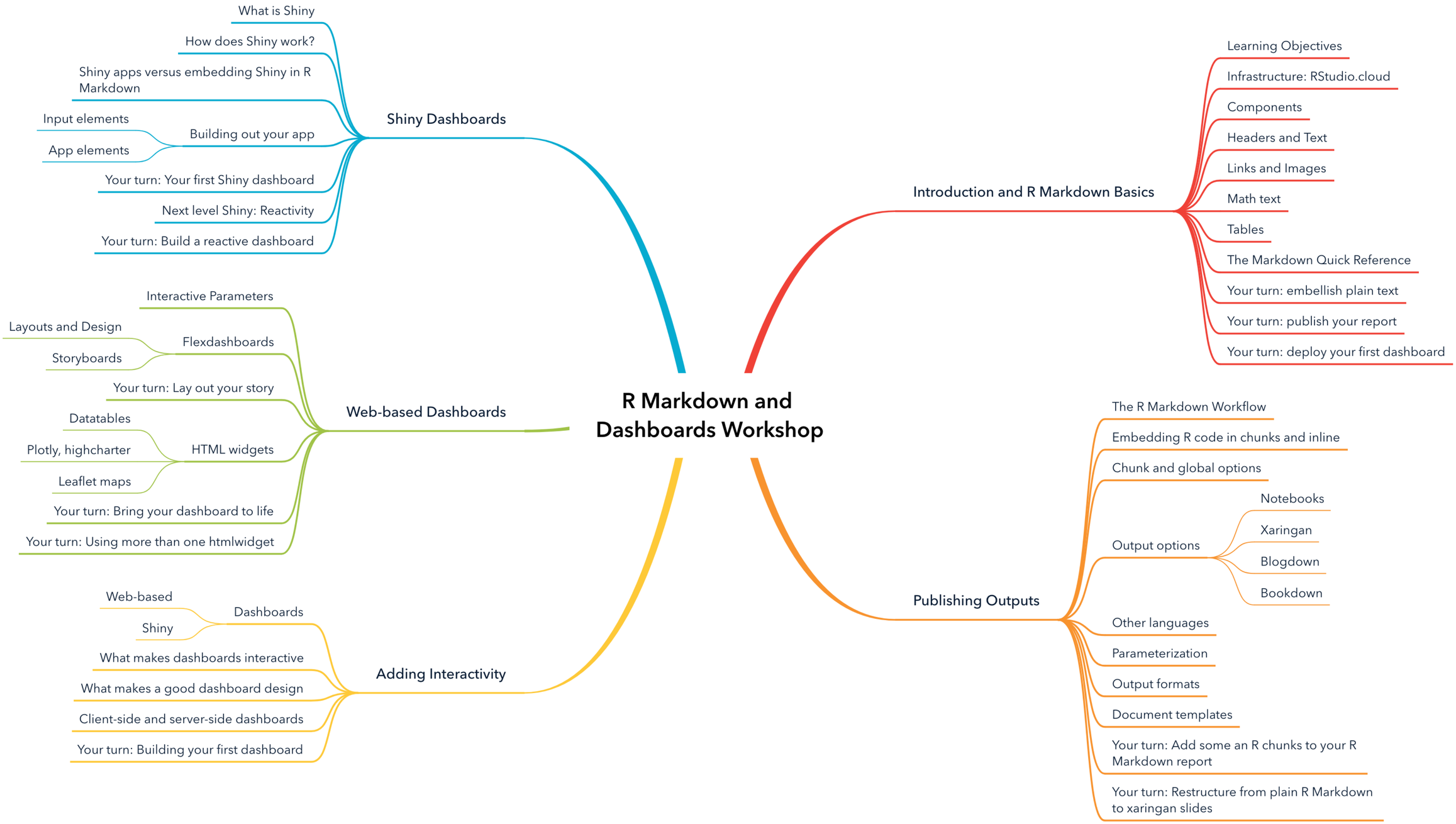
Pull It All Together

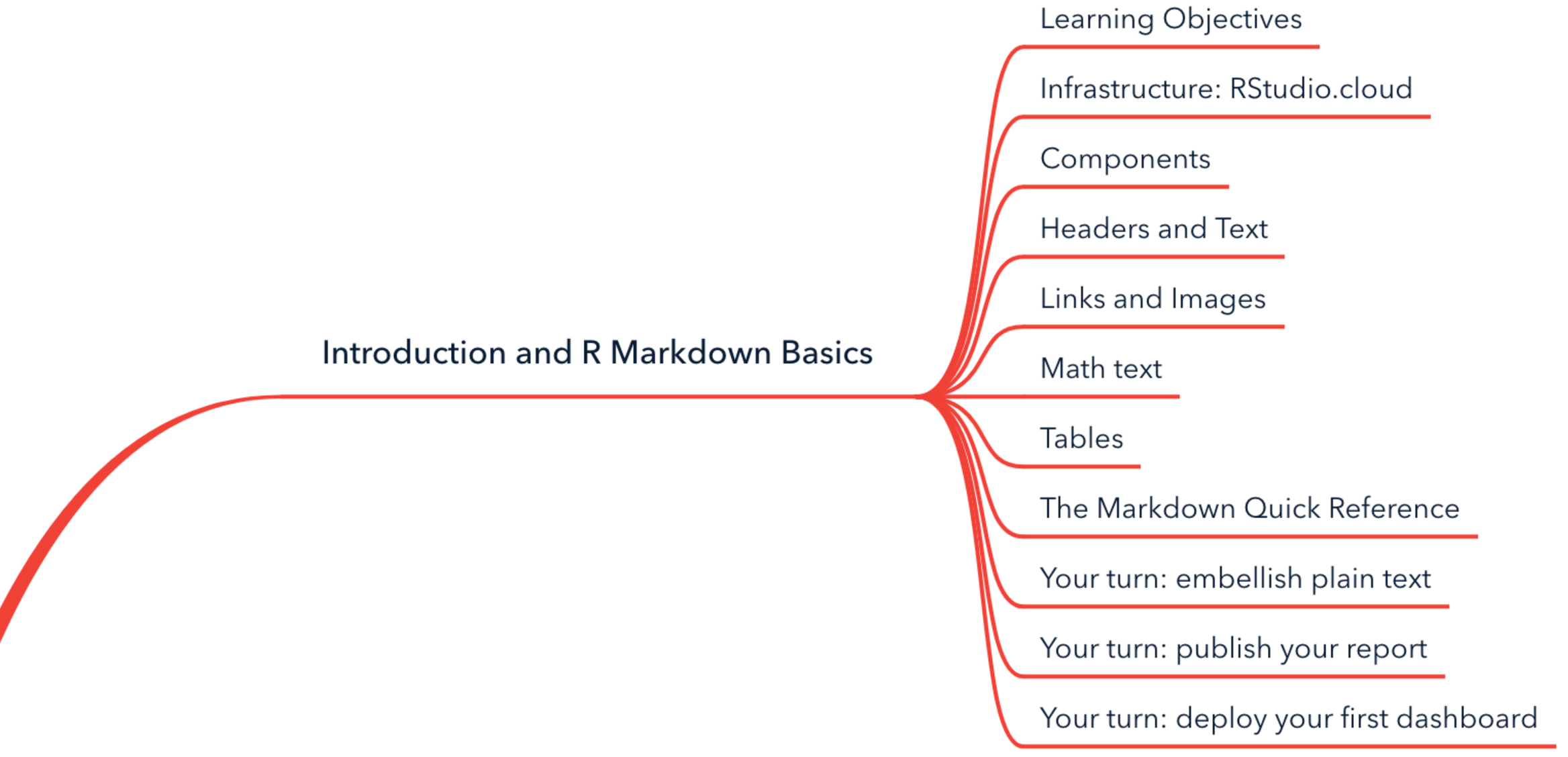
1. Open file **`56-movies-and-earthquakes.Rmd`**
2. Modify the existing plots to use **`plotly`**
3. Collect the existing materials into a single first level header **`# Movies`**
4. Create a new first level header page titled **`# Earthquakes`** that plots recent earthquakes using **`leaflet`** and **`renderLeaflet`**. You can find the leaflet code in the dashboard **`01-my-first-dashboard.Rmd`** in the **`01-Introduction`** directory; you will, however, want to remove the layer with temperatures and the grouping legend code.
5. Extra credit: add a datatable below your leaflet map with the most recent earthquakes

RStudio Cloud

Wrap Up


Web-based Dashboards
Interactive Parameters
Flexdashboards
Layouts and Design
Storyboards
Your turn: Lay out your story
HTML widgets
Datatables
Plotly, highcharter
Leaflet maps
Your turn: Bring your dashboard to life
Your turn: Using more than one htmlwidget


R Markdown and Dashboards Workshop
Introduction and R Markdown Basics
Learning Objectives
Infrastructure: RStudio.cloud
Components
Headers and Text
Links and Images
Math text
Tables
The Markdown Quick Reference
Your turn: embellish plain text
Your turn: publish your report
Your turn: deploy your first dashboard
Publishing Outputs
The R Markdown Workflow
Embedding R code in chunks and inline
Chunk and global options
Output options
Notebooks
Xaringan
Blogdown
Bookdown
Other languages
Parameterization
Output formats
Document templates
Your turn: Add some an R chunks to your R Markdown report
Your turn: Restructure from plain R Markdown to xaringan slides
Adding Interactivity
Dashboards
Web-based
Shiny
What makes dashboards interactive
What makes a good dashboard design
Client-side and server-side dashboards
Your turn: Building your first dashboard
Web-based Dashboards
Flexdashboards
Interactive Parameters
Layouts and Design
Storyboards
Your turn: Lay out your story
HTML widgets
Datatables
Plotly, highcharter
Leaflet maps
Your turn: Bring your dashboard to life
Your turn: Using more than one htmlwidget
Shiny Dashboards
What is Shiny
How does Shiny work?
Shiny apps versus embedding Shiny in R Markdown
Building out your app
Input elements
App elements
Your turn: Your first Shiny dashboard
Next level Shiny: Reactivity
Your turn: Build a reactive dashboard


Introduction and R Markdown Basics
Learning Objectives
Infrastructure: RStudio.cloud
Components
Headers and Text
Links and Images
Math text
Tables
The Markdown Quick Reference
Your turn: embellish plain text
Your turn: publish your report
Your turn: deploy your first dashboard

R Markdown is a language for creating ***computational documents****

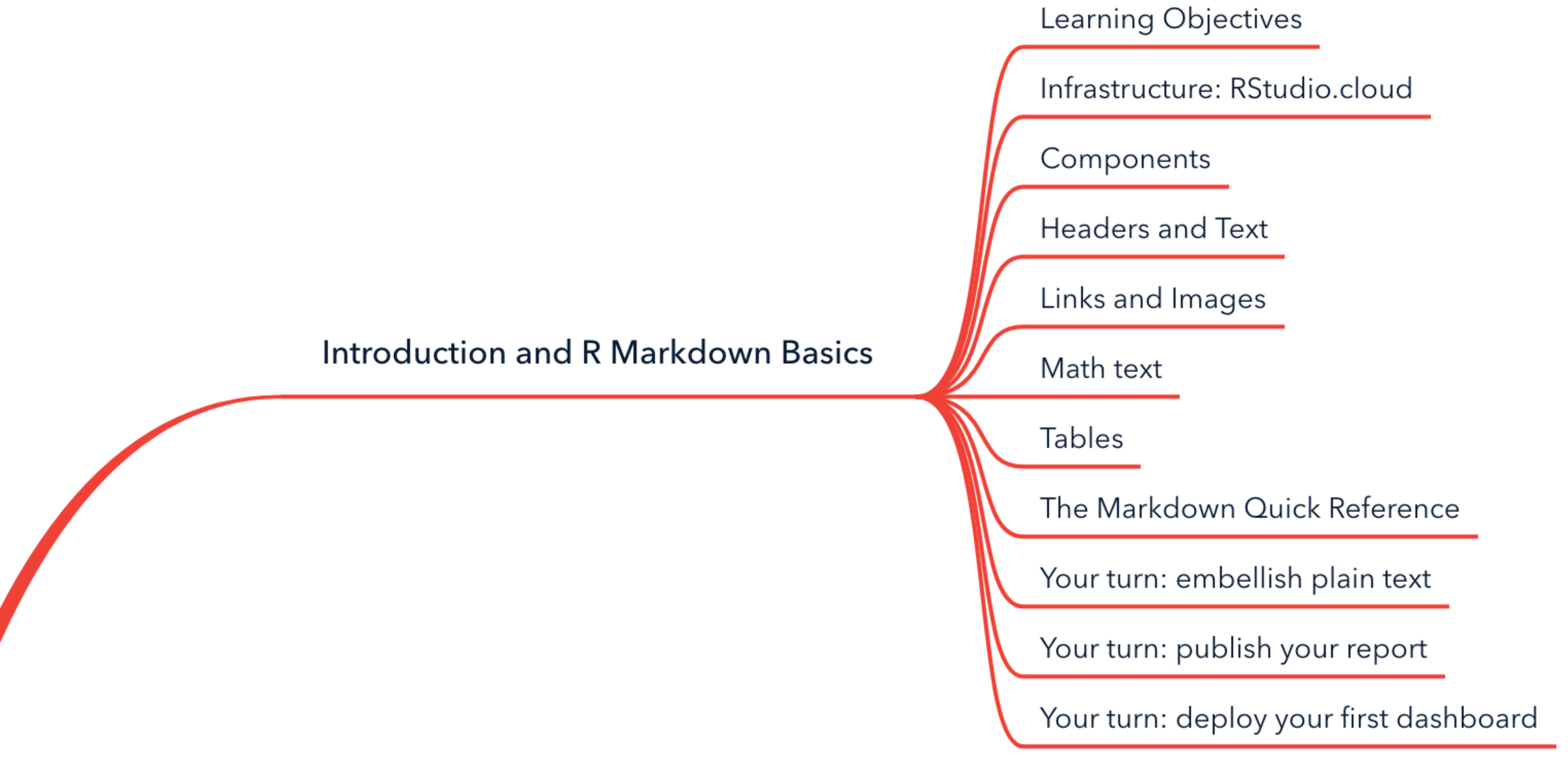
Donald Knuth's name for computational documents was *Literate Programming*, which he defined in a book of the same name. I prefer computation documents because it is more intuitively descriptive.

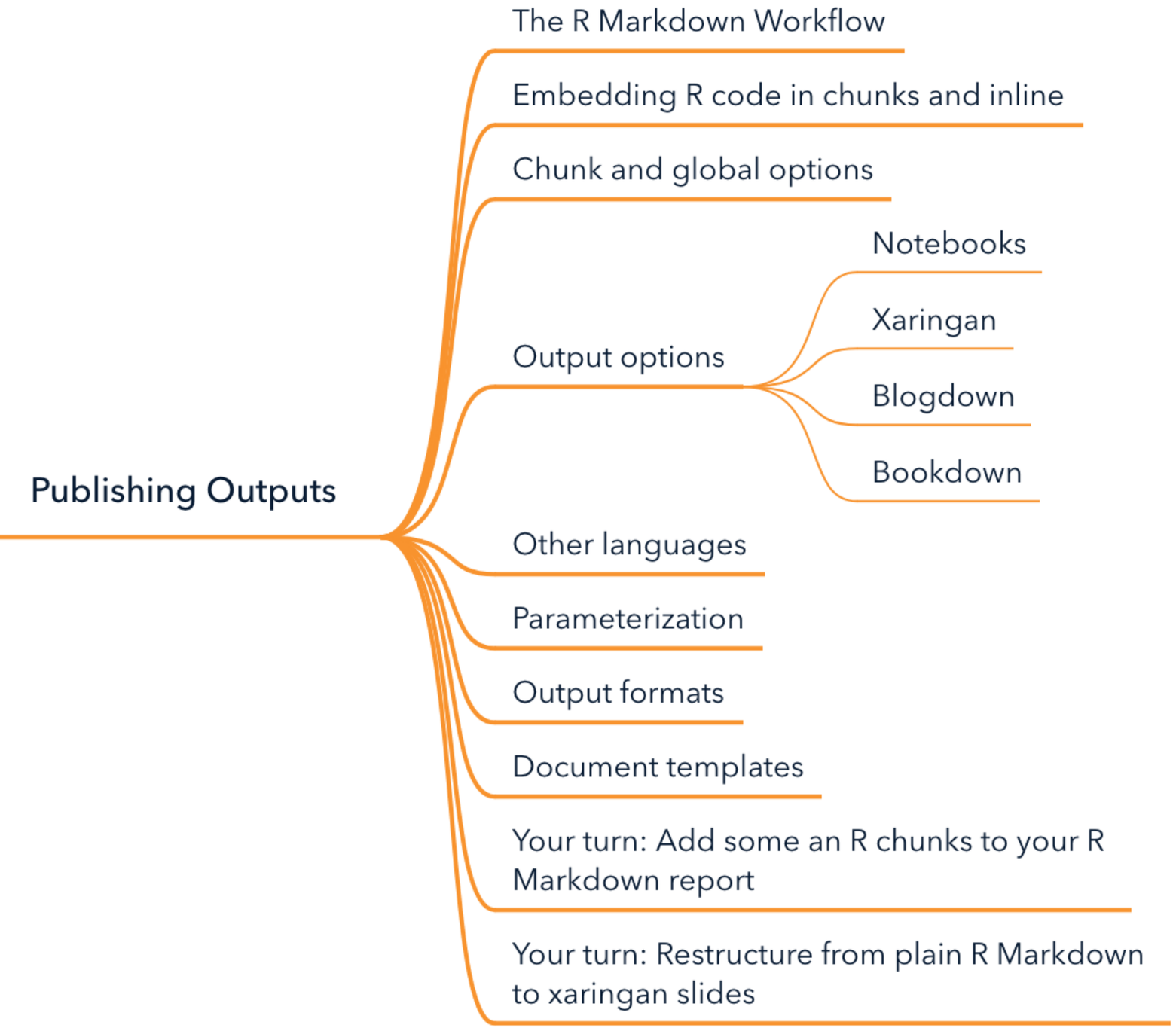
Metadata for an R Markdown document is written at the top of the document in YAML

R Markdown flags structural elements of our text for special treatment

Code chunks delimited by ``` glyphs allow us to add computation to our document.

R Markdown code chunks support many languages.


Introduction and R Markdown Basics
Learning Objectives
Infrastructure: RStudio.cloud
Components
Headers and Text
Links and Images
Math text
Tables
The Markdown Quick Reference
Your turn: embellish plain text
Your turn: publish your report
Your turn: deploy your first dashboard

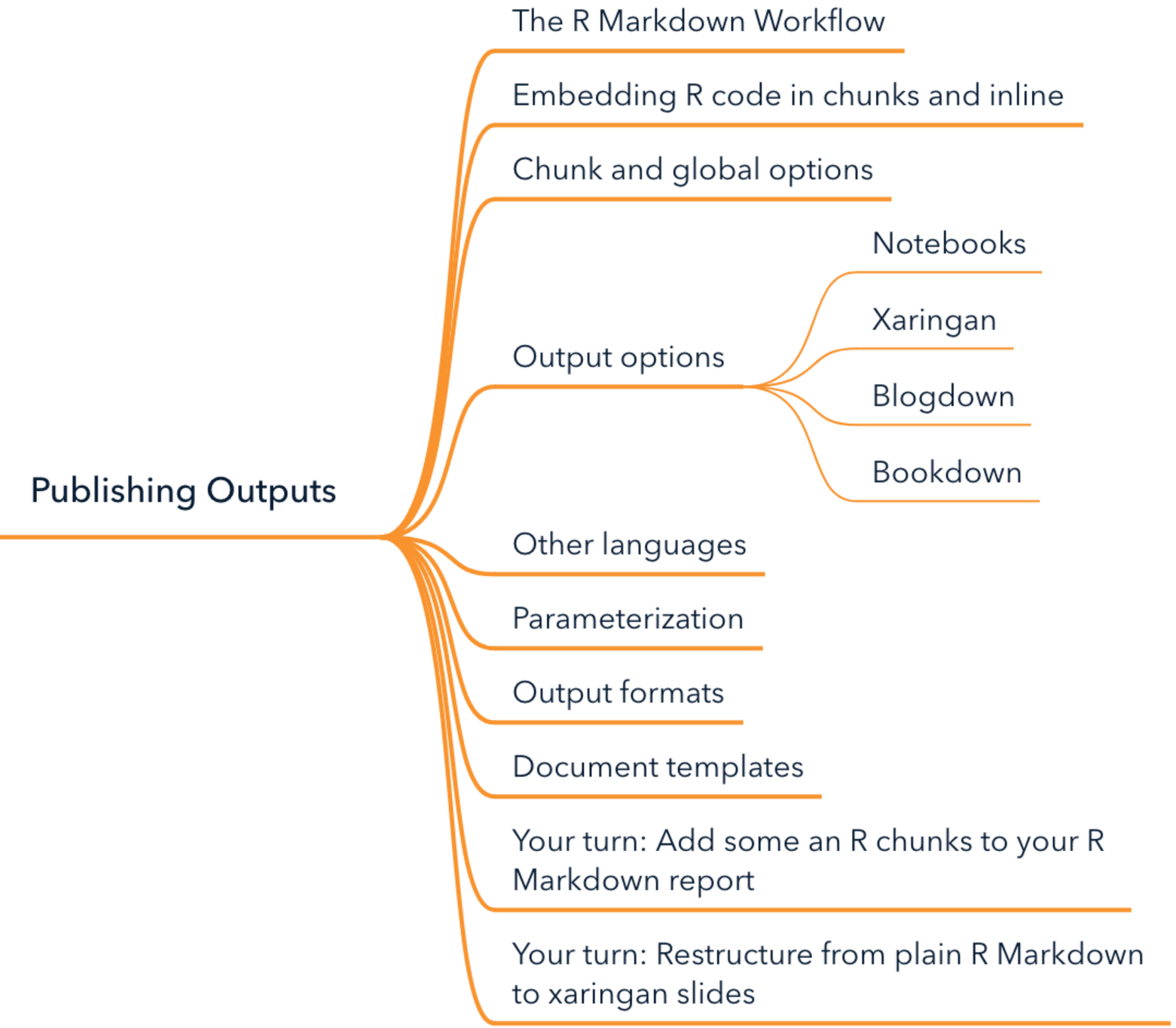

Publishing Outputs
The R Markdown Workflow
Embedding R code in chunks and inline
Chunk and global options
Output options
Notebooks
Xaringan
Blogdown
Bookdown
Other languages
Parameterization
Output formats
Document templates
Your turn: Add some an R chunks to your R Markdown report
Your turn: Restructure from plain R Markdown to xaringan slides

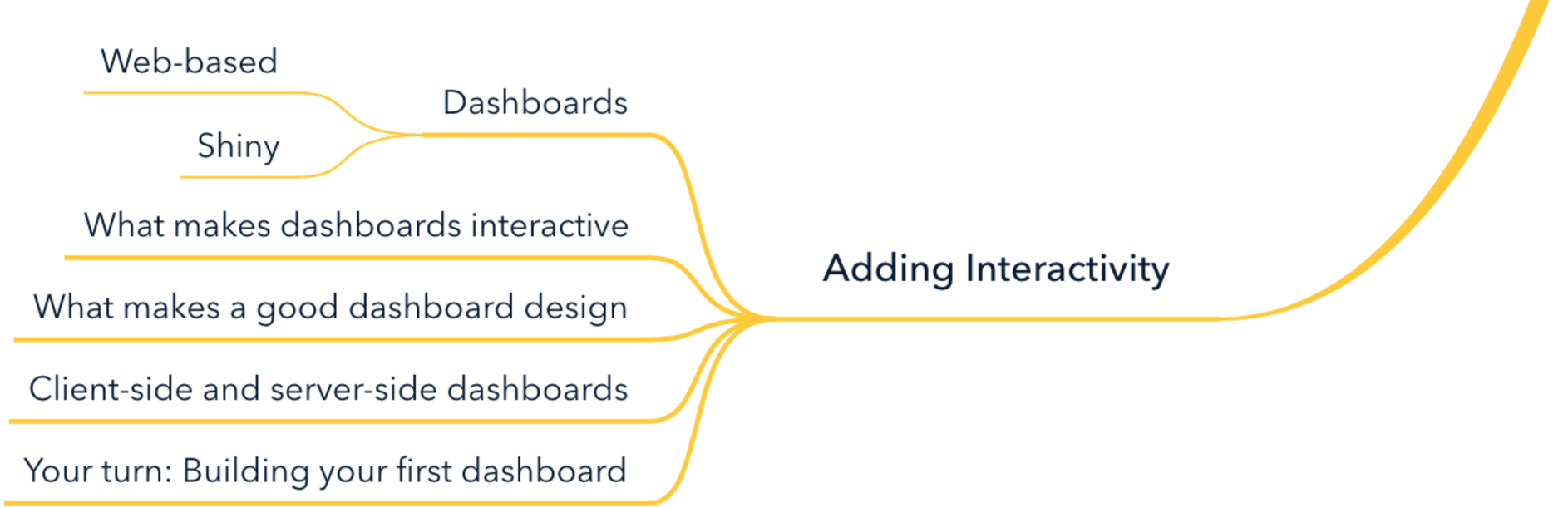
R Markdown is a system for creating **structured** *computational documents*

R Markdown can knit to many document types

HTML-based R Markdown documents can be shared using the **`publish`** *button in the RStudio IDE*

R Markdown similar to that for reports and slides can also generate dashboards.

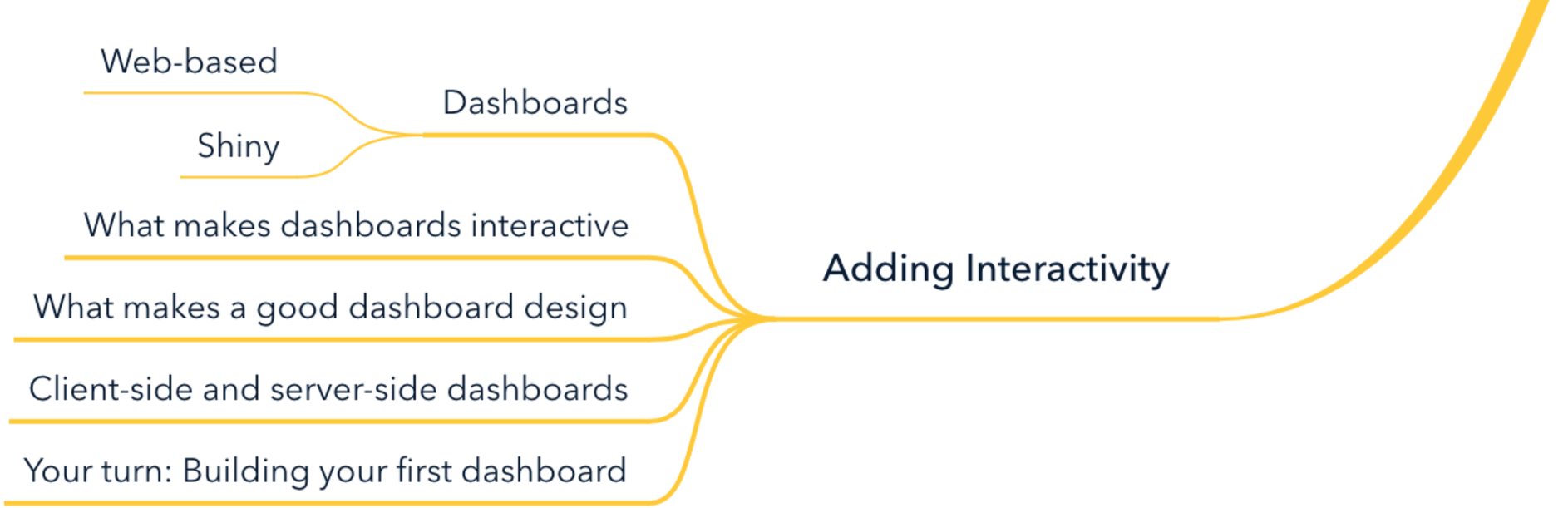

Publishing Outputs
The R Markdown Workflow
Embedding R code in chunks and inline
Chunk and global options
Output options
Notebooks
Xaringan
Blogdown
Bookdown
Other languages
Parameterization
Output formats
Document templates
Your turn: Add some an R chunks to your R Markdown report
Your turn: Restructure from plain R Markdown to xaringan slides

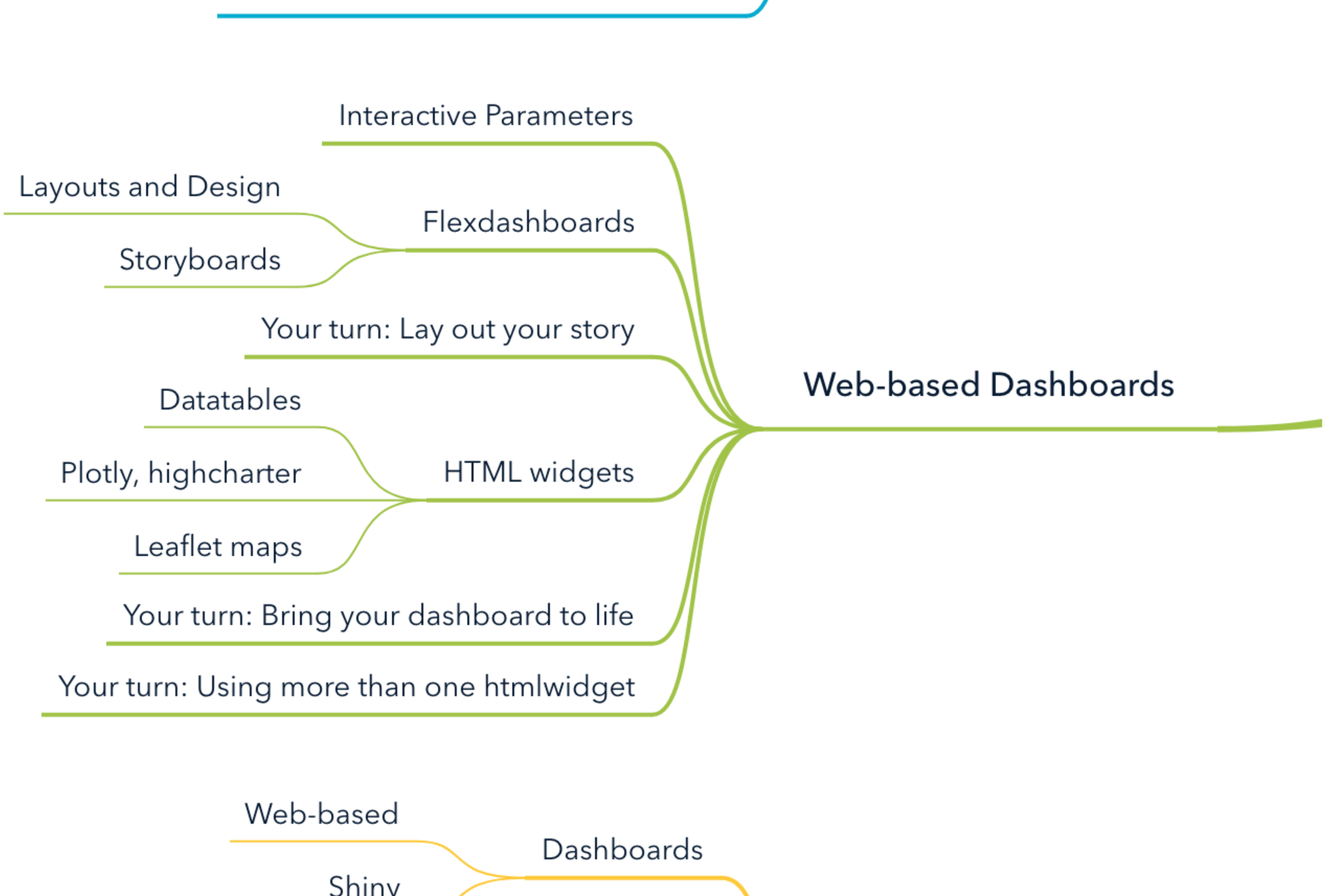

Web-based
Dashboards
Shiny
What makes dashboards interactive
What makes a good dashboard design
Client-side and server-side dashboards
Your turn: Building your first dashboard
Adding Interactivity

Interactivity is the gateway drug that hooks users on your content

R Markdown supports three different types of interactivity: one-time, web-based, and server-side

Tailor your interactivity plan to your data and user goals


Web-based
Dashboards
Shiny
What makes dashboards interactive
What makes a good dashboard design
Client-side and server-side dashboards
Your turn: Building your first dashboard
Adding Interactivity


Interactive Parameters
Layouts and Design
Flexdashboards
Storyboards
Your turn: Lay out your story
Datatables
Plotly, highcharter
HTML widgets
Leaflet maps
Your turn: Bring your dashboard to life
Your turn: Using more than one htmlwidget
Web-based Dashboards
Web-based
Dashboards
Shiny

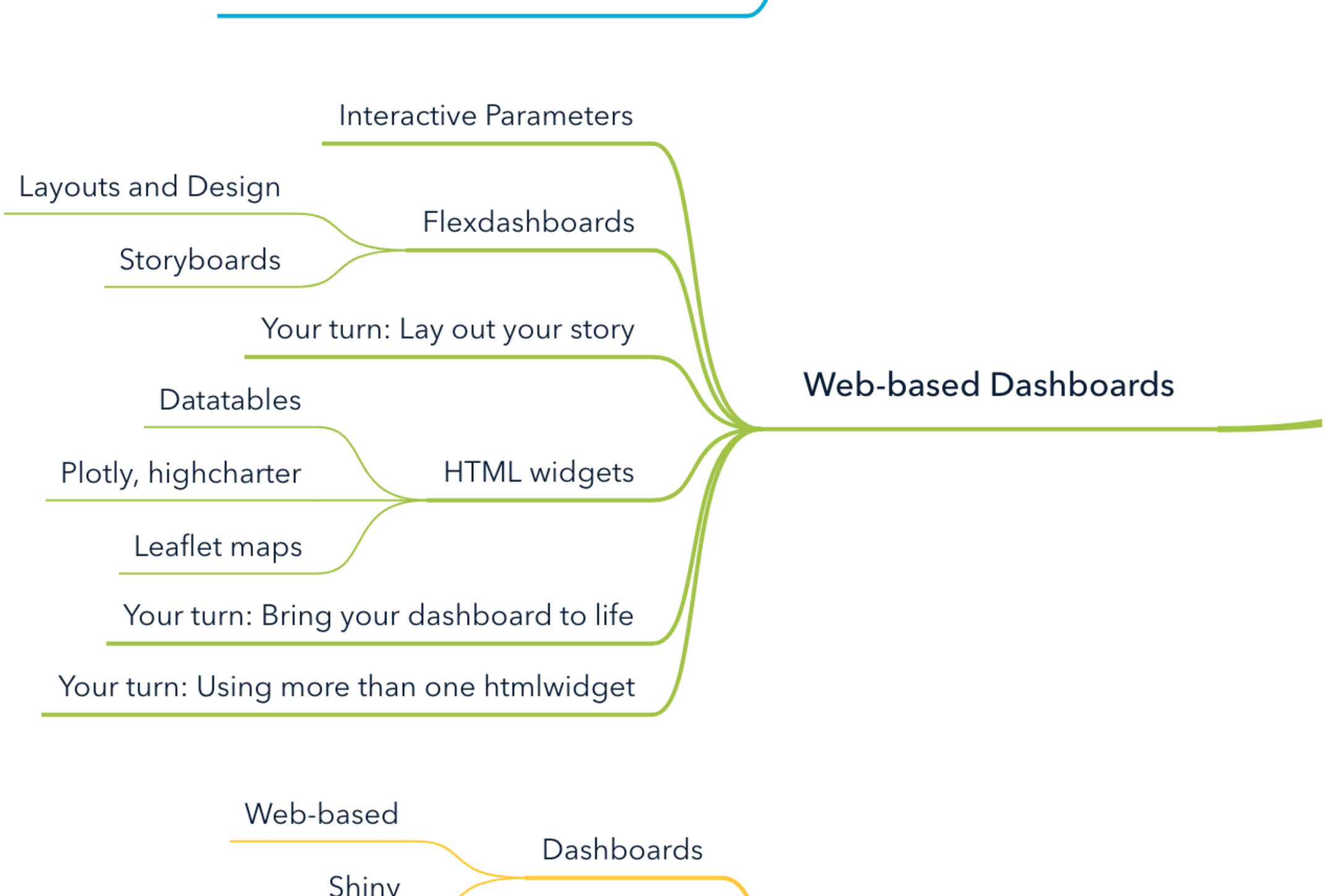
Dashboards have three building blocks:

1. Controls
2. Displays
3. Layouts

R Markdown layouts use header information to mark rows and columns on pages

HTML widgets create display interactivity using the user's browser

We build interactive controls using either kniting with parameters or using Shiny


Interactive Parameters
Layouts and Design
Flexdashboards
Storyboards
Your turn: Lay out your story
Datatables
Plotly, highcharter
HTML widgets
Leaflet maps
Your turn: Bring your dashboard to life
Your turn: Using more than one htmlwidget
Web-based Dashboards
Web-based
Dashboards
Shiny


What is Shiny
How does Shiny work?
Shiny apps versus embedding Shiny in R Markdown
Input elements
Building out your app
App elements
Shiny Dashboards
Your turn: Your first Shiny dashboard
Next level Shiny: Reactivity
Your turn: Build a reactive dashboard

Shiny is an R package that makes it easy to build interactive web apps from R.

shiny.rstudio.com

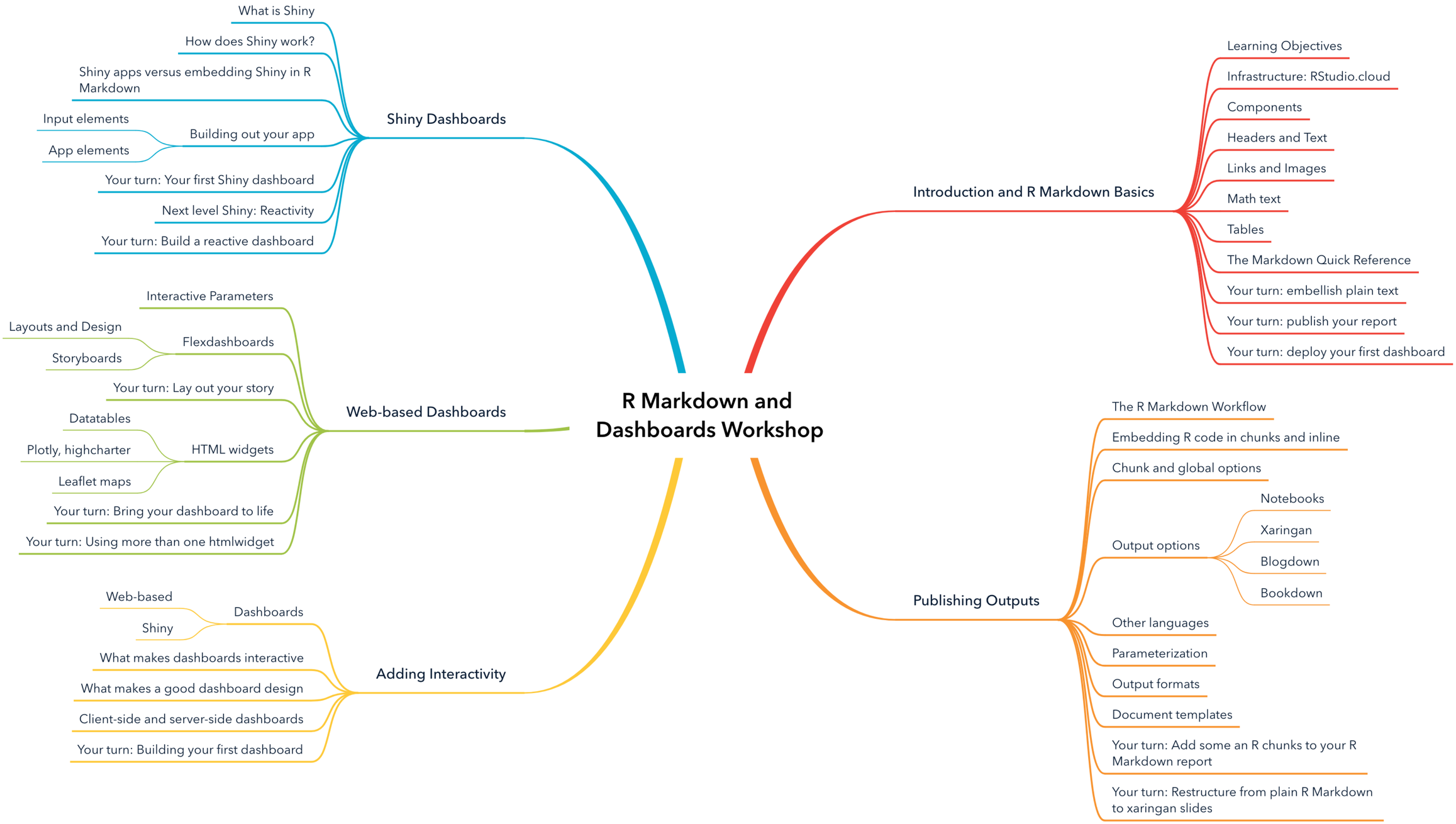
Shiny offers user interface and rendering functions for input and output controls.

Shiny inputs drive outputs through ***reactivity***

Publishing Shiny apps requires a server that runs R. shinyapps.io and RStudio Connect are two easy options

You can use HTML Widgets in your Shiny app by modifying your render functions


What is Shiny
How does Shiny work?
Shiny apps versus embedding Shiny in R Markdown
Input elements
Building out your app
App elements
Shiny Dashboards
Your turn: Your first Shiny dashboard
Next level Shiny: Reactivity
Your turn: Build a reactive dashboard


R Markdown and Dashboards Workshop
Introduction and R Markdown Basics
Learning Objectives
Infrastructure: RStudio.cloud
Components
Headers and Text
Links and Images
Math text
Tables
The Markdown Quick Reference
Your turn: embellish plain text
Your turn: publish your report
Your turn: deploy your first dashboard
Publishing Outputs
The R Markdown Workflow
Embedding R code in chunks and inline
Chunk and global options
Output options
Notebooks
Xaringan
Blogdown
Bookdown
Other languages
Parameterization
Output formats
Document templates
Your turn: Add some an R chunks to your R Markdown report
Your turn: Restructure from plain R Markdown to xaringan slides
Adding Interactivity
Dashboards
Web-based
Shiny
What makes dashboards interactive
What makes a good dashboard design
Client-side and server-side dashboards
Your turn: Building your first dashboard
Web-based Dashboards
Flexdashboards
Interactive Parameters
Layouts and Design
Storyboards
Your turn: Lay out your story
HTML widgets
Datatables
Plotly, highcharter
Leaflet maps
Your turn: Bring your dashboard to life
Your turn: Using more than one htmlwidget
Shiny Dashboards
What is Shiny
How does Shiny work?
Shiny apps versus embedding Shiny in R Markdown
Building out your app
Input elements
App elements
Your turn: Your first Shiny dashboard
Next level Shiny: Reactivity
Your turn: Build a reactive dashboard

RStudio Education

education.rstudio.com

R Markdown and Interactive Dashboards

Carl Howe, Director of Education, RStudio
Kelly O'Brient, Solutions Engineer, RStudio

rstd.io/RMAID