

sowla / 2019-shiny-intro-workshop-Nijmegen-RLadies

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. Edit

Manage topics

4 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

sowla add workshop slides Latest commit 8b114f3 a minute ago

data	add workshop slides	a minute ago
example apps	add workshop slides	a minute ago
.gitignore	add refresher slides	12 days ago
190221_Nijmegen_Rladies_Shiny_workshop.pdf	add workshop slides	a minute ago
190221_Nijmegen_Rladies_Shiny_workshop.pptx	add workshop slides	a minute ago
190221_Nijmegen_Rladies_workshop_R_refresher.pdf	fix typos	9 days ago



Please download this github repository: <https://bit.ly/2BLmoGW>

Introduction to



R-Ladies Nijmegen 21/02/19

Hi!

I'm Suthira Owlarn :)

- Biologist/postdoc in Kerstin Bartscherer's lab
 - MPI for Molecular Biomedicine (Münster)
 - Hubrecht Institute (Utrecht)
- R newbie (~1.5 years)
- Shiny enthusiast!



[@s_owla](https://twitter.com/s_owla)



[sowla](https://github.com/sowla)

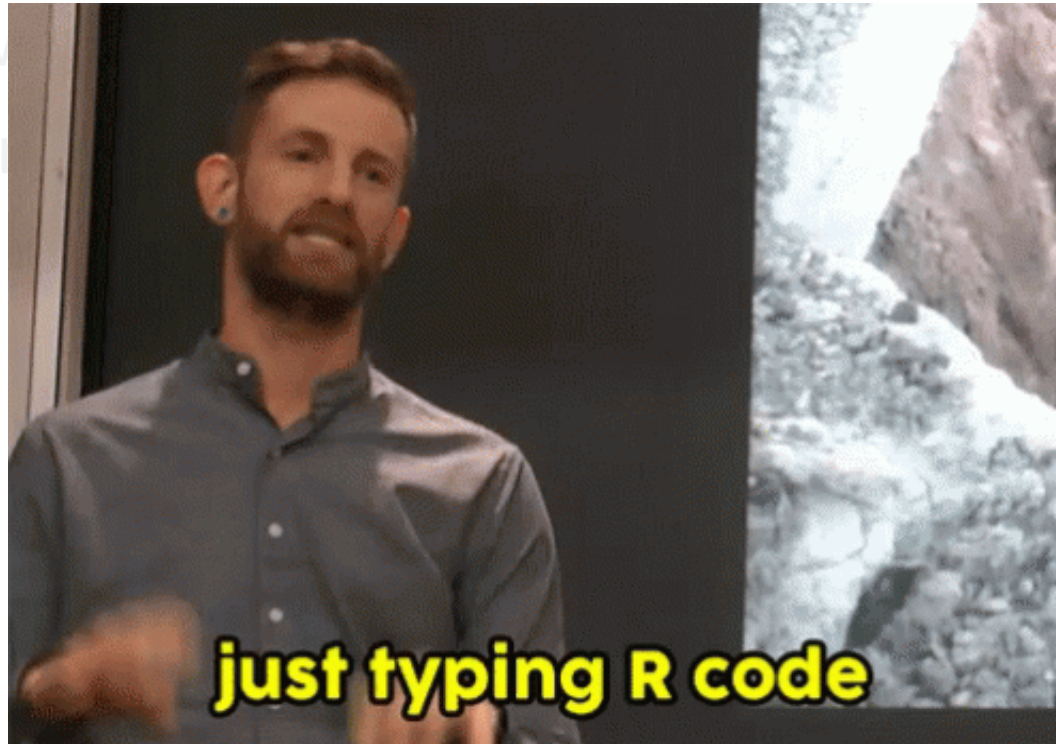
Tonight's plan

1. Introduction + demo
2. Build a sample app together (with short exercises)
3. Build your own app (longer hands-on session)
4. Q&A/summary

Tonight's plan

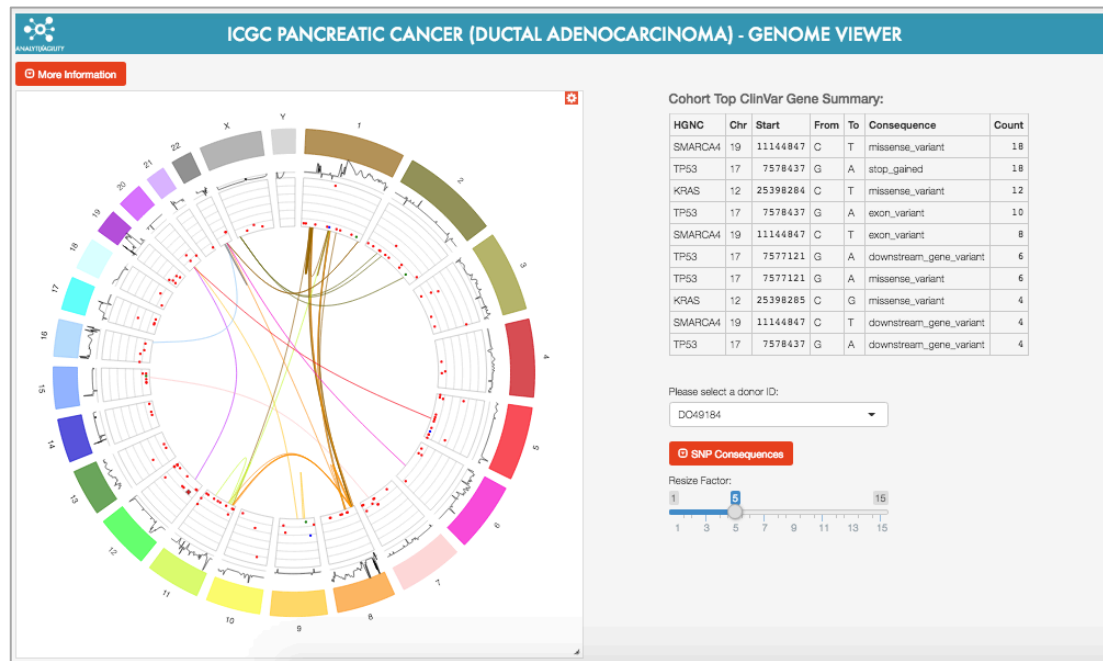
1. Introduction + demo
2. Build a sample app together (with short exercises)
3. Build your own
4. Q&A/summary

GIF by
Mara Averick
([@dataandme](https://twitter.com/dataandme))

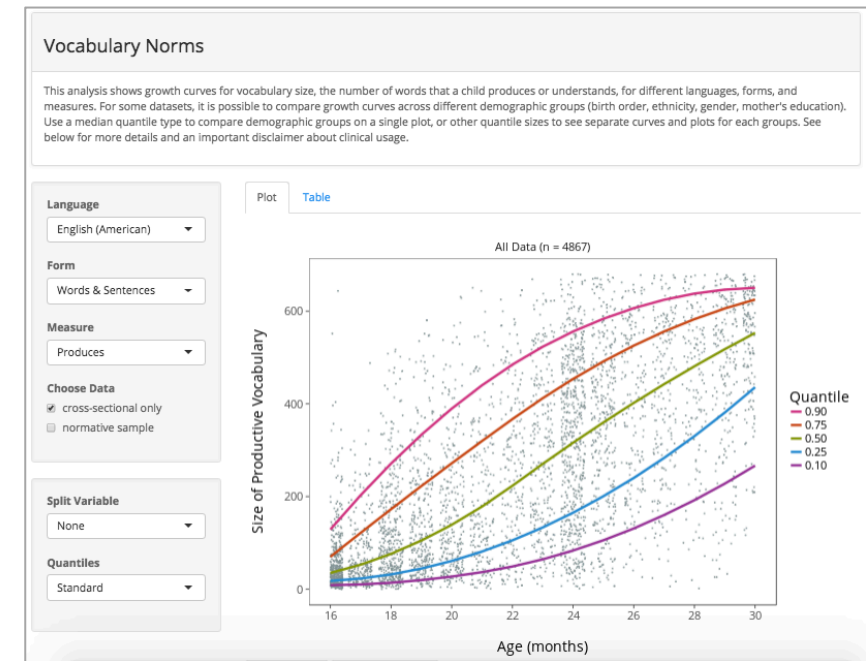


Shiny

A quick and easy way to provide a user-friendly interface to your analysis



ICGC pancreatic cancer genome viewer



Word Bank

Demo



"OH, NO -- THAT'S JUST THE
PROTOTYPE."

Getting started

sowla / 2019-shiny-intro-workshop-Nijmegen-RLadies

Unwatch ▾

1

★ Star

0

Fork

0

<> Code

! Issues 0

🔗 Pull requests 0

📁 Projects 0

📖 Wiki

📊 Insights

⚙ Settings

No description, website, or topics provided.

Edit

Manage topics

🕒 4 commits

🌿 1 branch

🏷 0 releases

👤 1 contributor

Branch: master ▾

New pull request

Create new file

Upload files

Find file

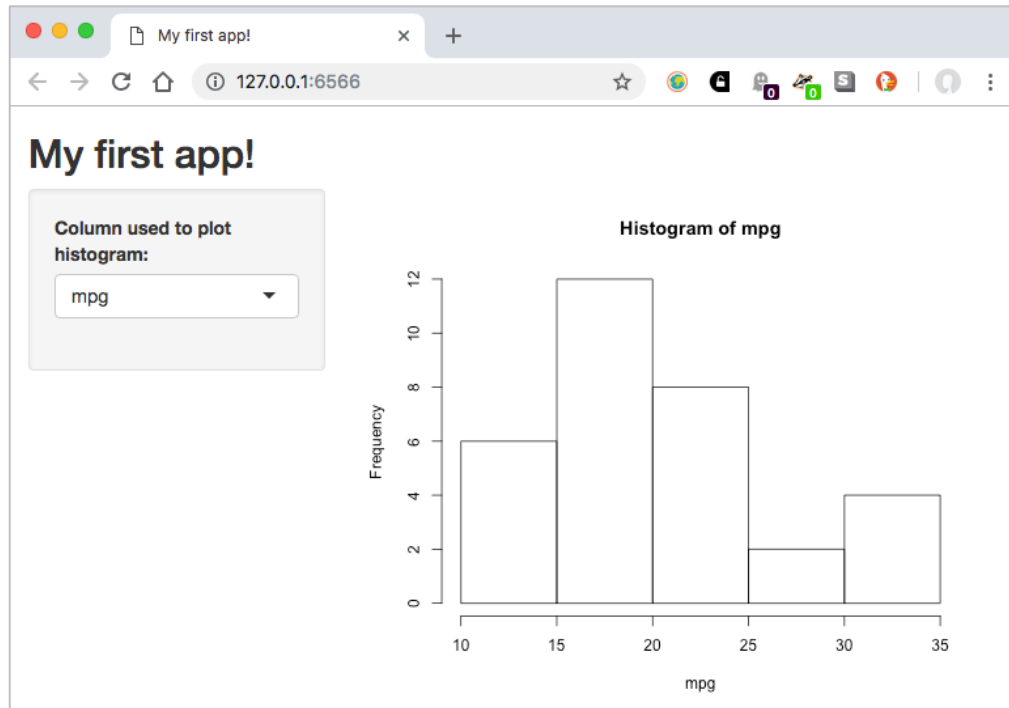
Clone or download ▾

🏠 sowla add workshop slides

Latest commit 8b114f3 a minute ago

📁 data	add workshop slides	a minute ago
📁 example apps	add workshop slides	a minute ago
📄 .gitignore	add refresher slides	12 days ago
📄 190221_Nijmegen_Rladies_Shiny_workshop.pdf	add workshop slides	a minute ago
📄 190221_Nijmegen_Rladies_Shiny_workshop.pptx	add workshop slides	a minute ago

How does shiny work?



Translate R code into code
for the user interface

```
# plot a histogram
```

```
hist(  
  x = my_data[["mpg"]],  
  main = paste("Histogram of", "mpg"),  
  xlab = "mpg"  
)
```

Use R to do the calculations
behind the scene

How does shiny work?

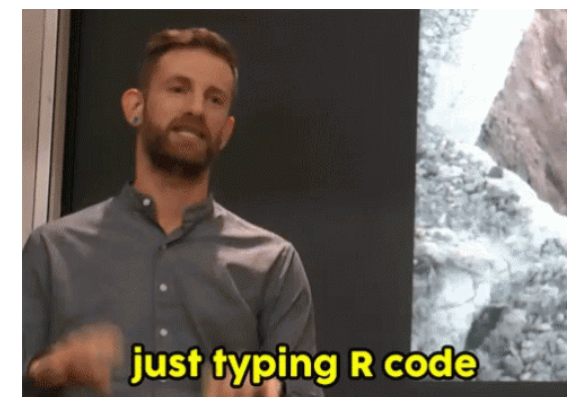
- Shiny hides many of the complexities :D
- Many possible inputs and outputs
- Lots of other customisations possible
- But first we'll focus on building one app together, step-by-step

Our target shiny app

Let's have a look at the app we're trying to build:

- Open **1-app-example.R**
- Start the app (“run app”; top right of script panel)
- Explore the app
- Stop the app (“stop”; top right of console panel)
- If you have extra time:
 - Go back to the browser
 - What happens when you try to use the app? Why?

3-5 minutes



A minimal shiny app

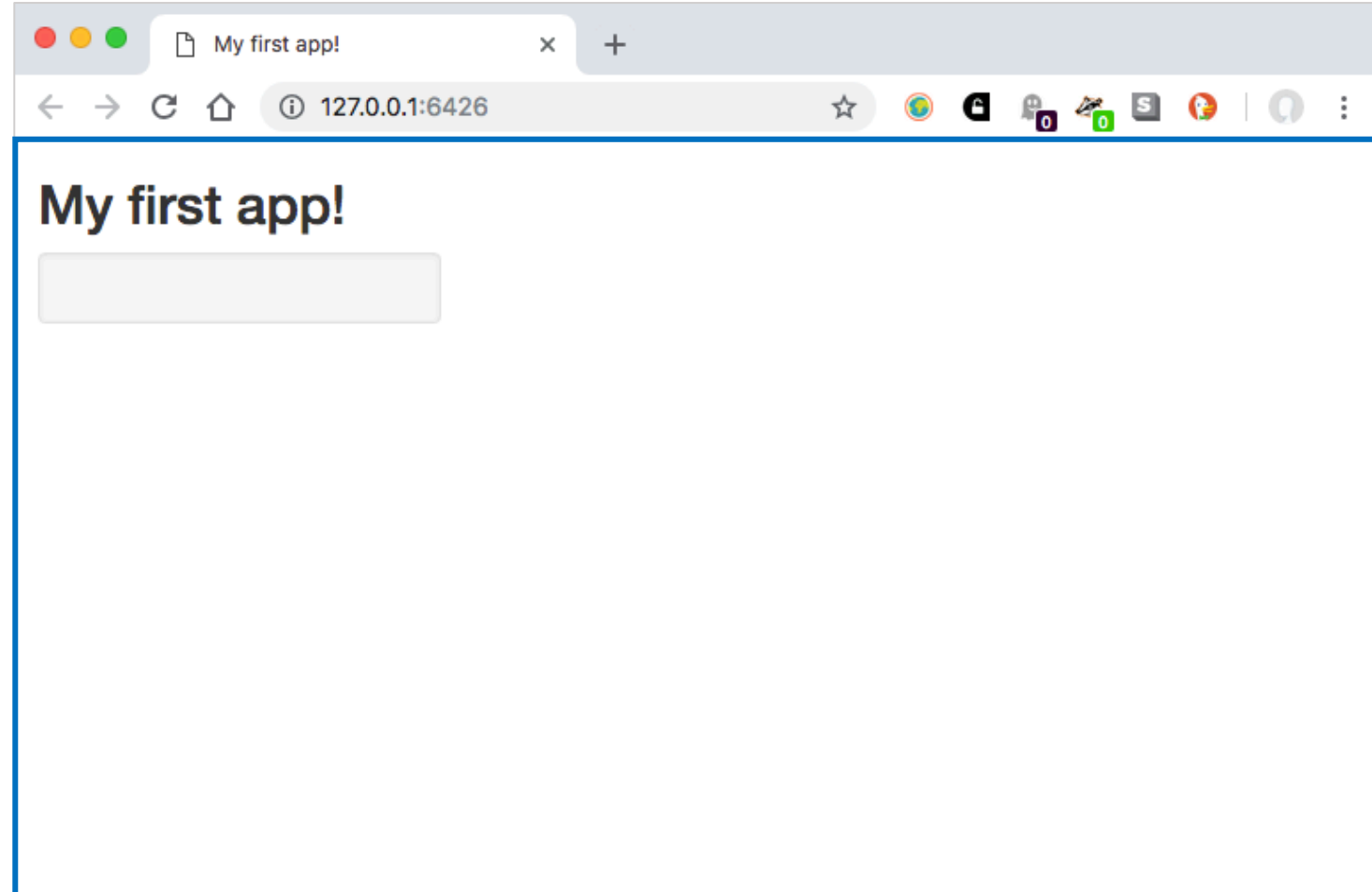
2-app-very-basic1.R:

```
library(shiny)
```

```
shinyApp(  
  ui = "",                                # what to go on the web page  
  server = function(input, output) {}    # calculations done in R  
)
```

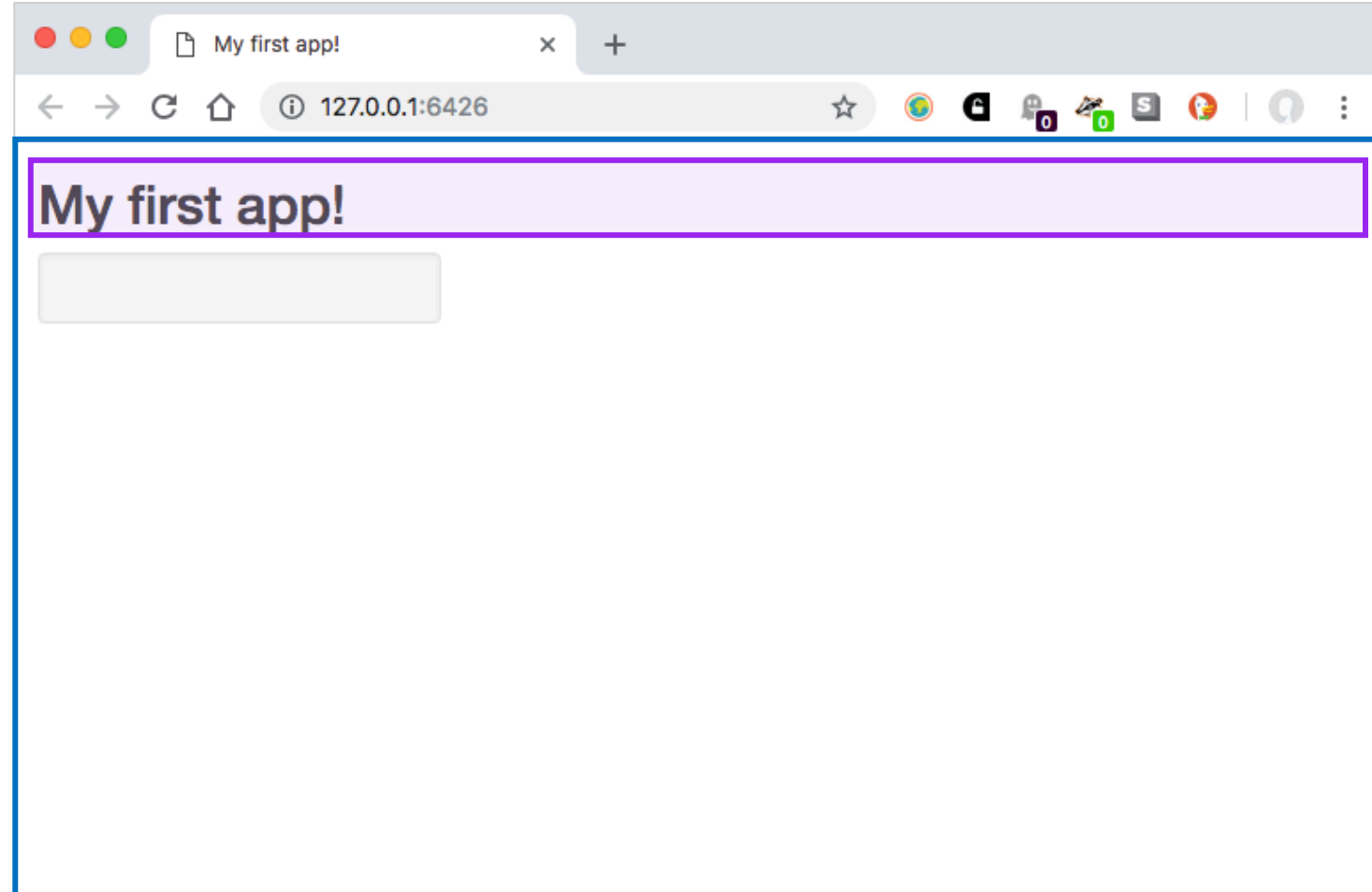
Building a user interface

```
ui <-  
  fluidPage(  
    titlePanel("My first app!"),  
    sidebarLayout(  
      sidebarPanel(),  
  
      mainPanel()  
    )  
  )
```



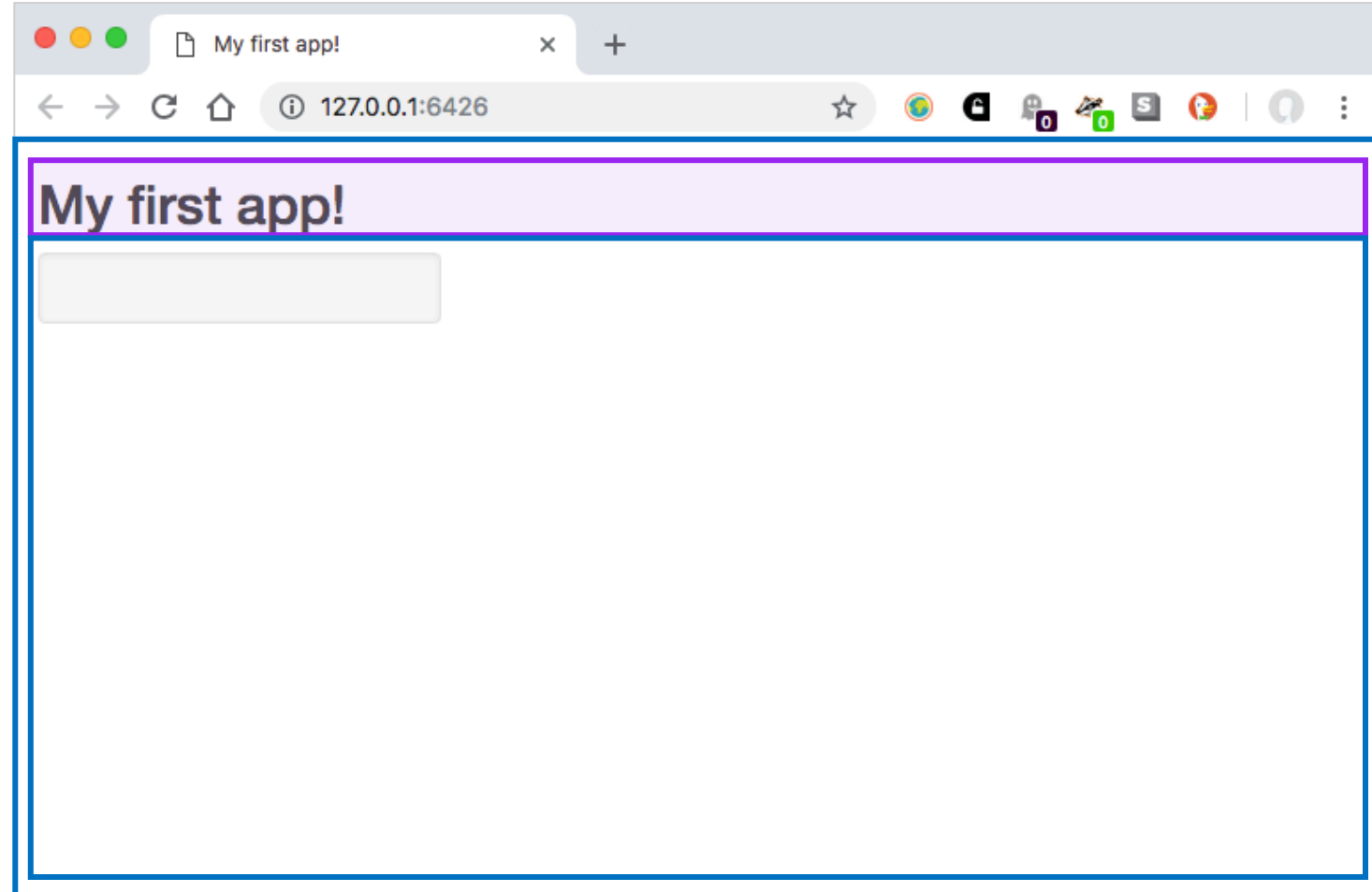
Building a user interface

```
ui <-  
  fluidPage(  
    titlePanel("My first app!"),  
    sidebarLayout(  
      sidebarPanel(),  
  
      mainPanel()  
    )  
  )
```



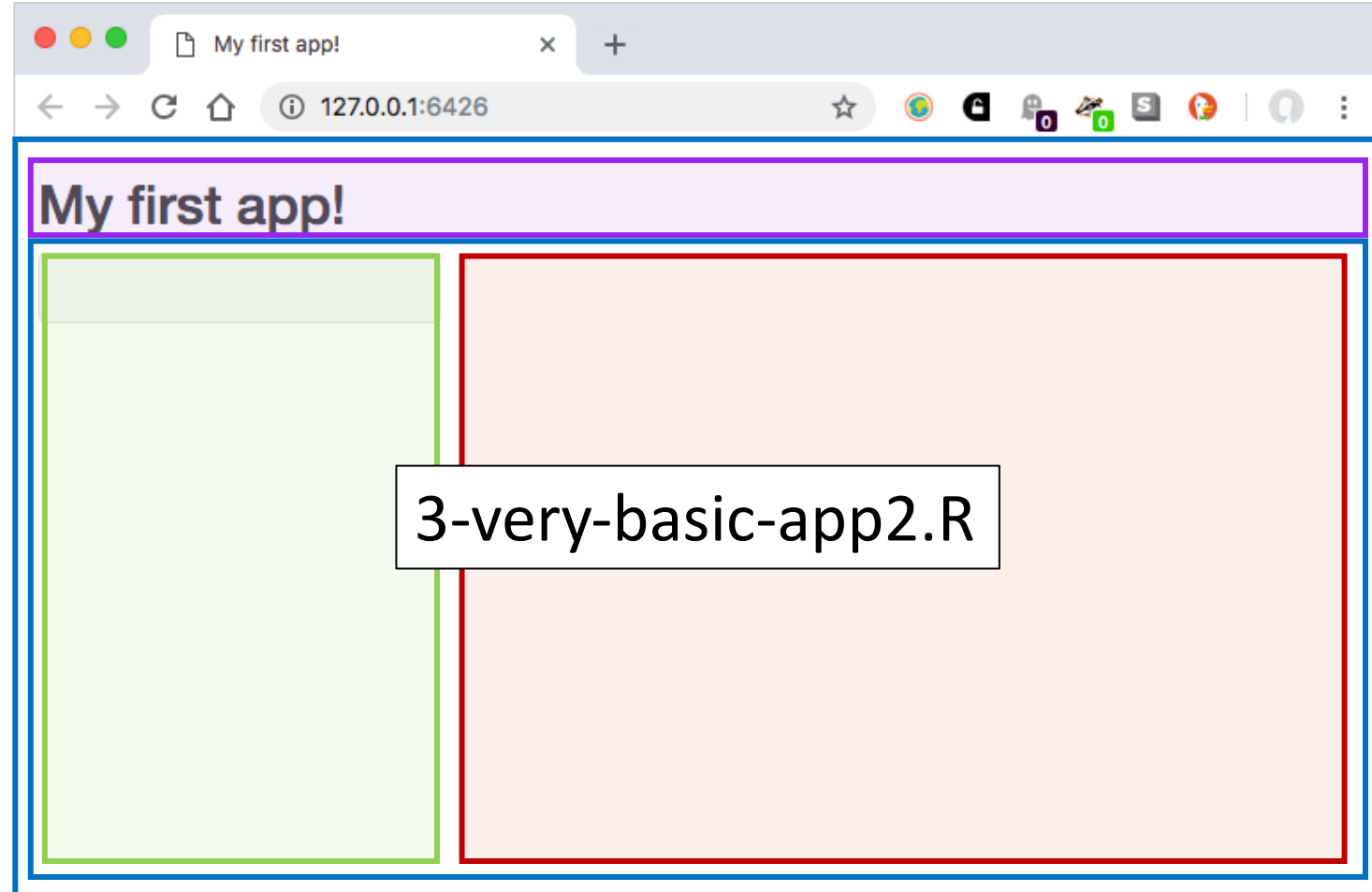
Building a user interface

```
ui <-  
  fluidPage(  
    titlePanel("My first app!"),  
    sidebarLayout(  
      sidebarPanel(),  
  
      mainPanel()  
    )  
  )
```



Building a user interface

```
ui <-  
  fluidPage(  
    titlePanel("My first app!"),  
    sidebarLayout(  
      sidebarPanel(),  
  
      mainPanel()  
    )  
  )
```



Building a user interface

```
ui <-  
  fluidPage(  
    titlePanel("My first app!"),  
    sidebarLayout(  
      sidebarPanel(),  
  
      mainPanel()  
    )  
  )
```

Value of ui in 3-very-basic-app2.R:

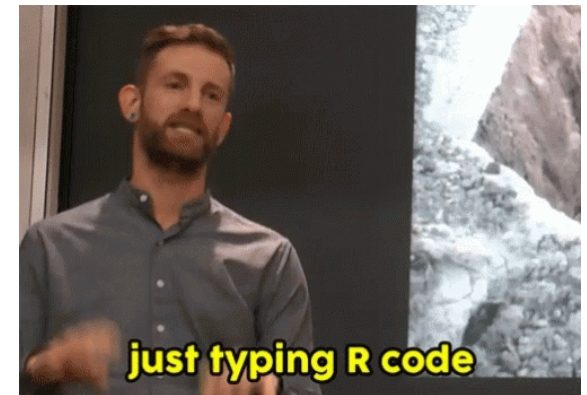
```
<div class="container-fluid">  
  <h2>My first app!</h2>  
  <div class="row">  
    <div class="col-sm-4">  
      <form class="well"></form>  
    </div>  
    <div class="col-sm-8">  
    </div>  
  </div>  
</div>
```

Building a user interface: your turn

Could you open **3-very-basic-app2.R** and:

1. change the title of your app?
 2. move the sidebar panel to the right?
- If you have extra time:
 - Did your neighbours use the same method as you?
 - There's more than one way to do this! :)

3-5 minutes



Building a user interface: your turn

Canonical way:

```
ui <-  
  fluidPage(  
    titlePanel("My new title!"),  
    sidebarLayout(  
      position = "right",  
      sidebarPanel(),  
      mainPanel()  
    )  
  )
```

Gives the same result:

```
ui <-  
  fluidPage(  
    h2("My new title!"),  
    sidebarLayout(  
      mainPanel(),  
      sidebarPanel()  
    )  
  )
```

Building a user interface: your turn

My general mind set:

- use the canonical way when there is one!
- check documentation/internet for “lesser known” canonical way(s)
- but don't be afraid to be creative when there isn't one! :)

Building a user interface: inputs



actionButton(inputId, label, icon, ...)

Link

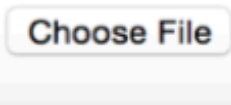
actionLink(inputId, label, icon, ...)



checkboxInput(inputId, label, value)



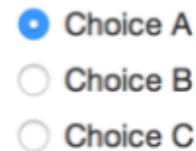
dateInput(inputId, label, value, min, max, format, startview, weekstart, language)



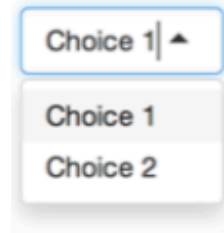
fileInput(inputId, label, multiple, accept)



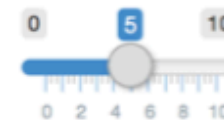
numericInput(inputId, label, value, min, max, step)



radioButtons(inputId, label, choices, selected, inline)



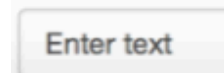
selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also **selectizeInput()**)



sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)



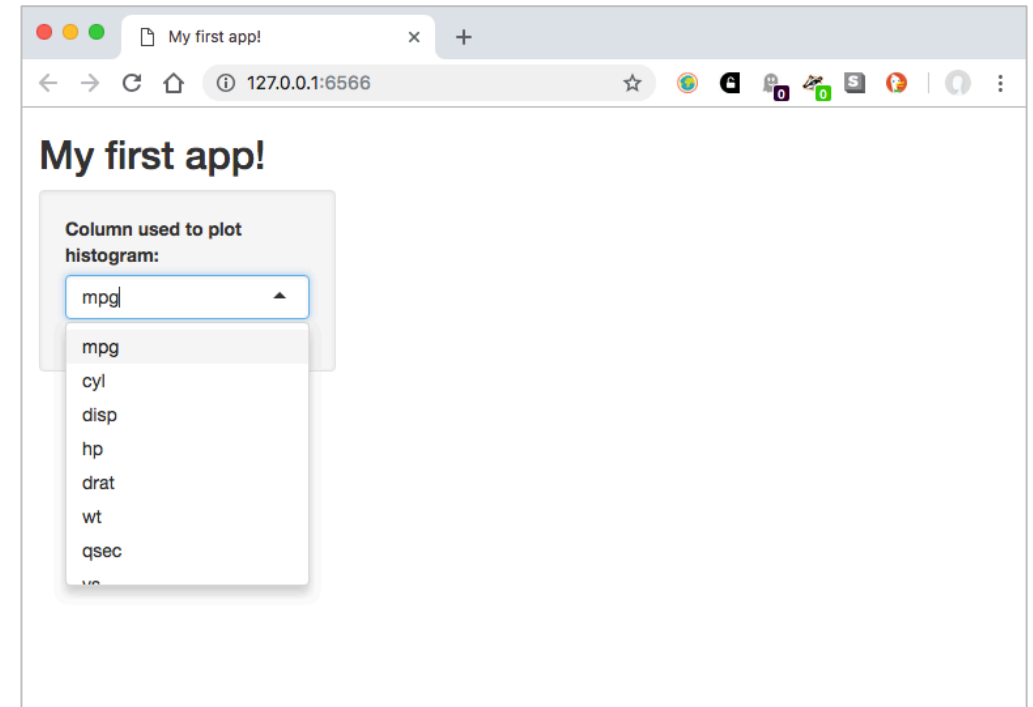
submitButton(text, icon)
(Prevents reactions across entire app)



textInput(inputId, label, value)

Building a user interface: inputs

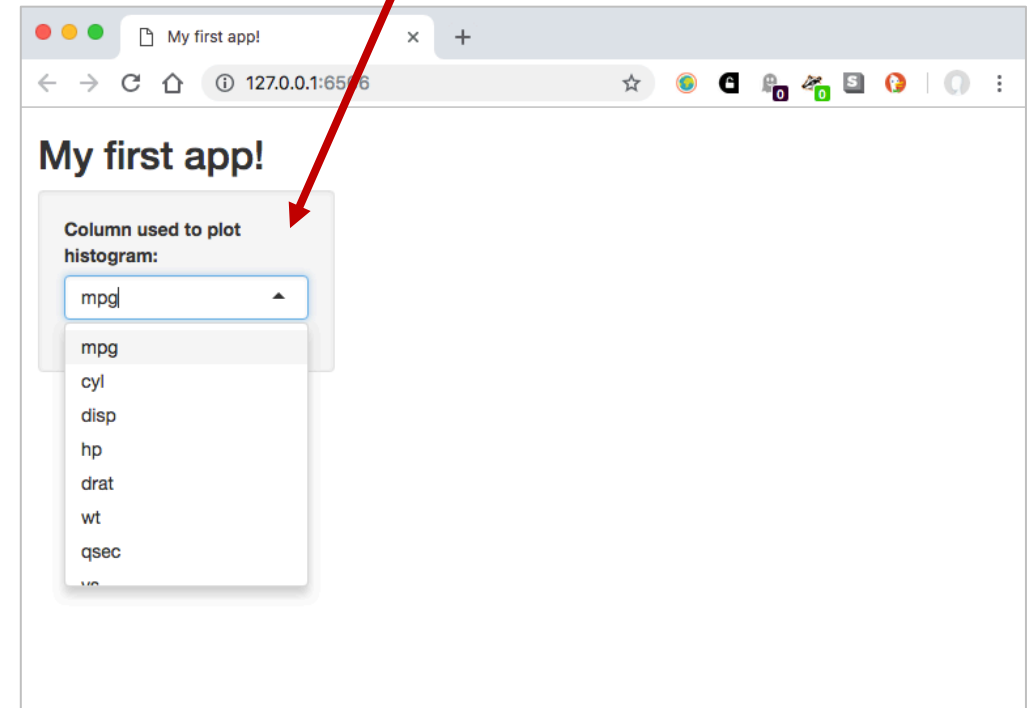
```
sidebarLayout(  
  sidebarPanel(  
    selectInput(  
      inputId = "column",  
      label = "Column used to plot histogram:",  
      choices = names(mtcars)  
    )  
  ),  
  mainPanel(  
    plotOutput(outputId = "plot")  
  )  
)
```



Building a user interface: inputs

```
sidebarLayout(  
  sidebarPanel(  
    selectInput(  
      inputId = "column",  
      label = "Column used to plot histogram:",  
      choices = names(mtcars)  
    )  
  ),  
  mainPanel(  
    plotOutput(outputId = "plot")  
  )  
)
```

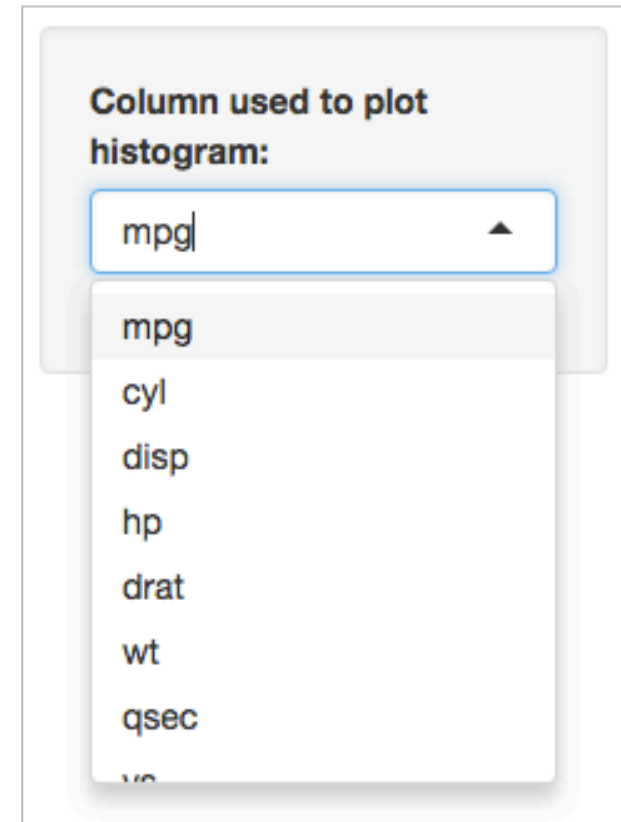
The input element goes
in the sidebar panel



Building a user interface: inputs

```
sidebarLayout(  
  sidebarPanel(  
    selectInput(  
      inputId = "column",  
      label = "Column used to plot histogram:",  
      choices = names(mtcars)  
    )  
  ),  
  mainPanel(  
    plotOutput(outputId = "plot")  
  )  
)
```

Type of input
(dropdown menu)



Building a user interface: inputs

```
sidebarLayout(  
  sidebarPanel(  
    selectInput(  
      inputId = "column",  
      label = "Column used to plot histogram:",  
      choices = names(mtcars)  
    )  
  ),  
  mainPanel(  
    plotOutput(outputId = "plot")  
  )  
)
```

Unique ID so we can
retrieve the correct
values from it

Building a user interface: inputs

```
sidebarLayout(  
  sidebarPanel(  
    selectInput(  
      inputId = "column",  
      label = "Column used to plot histogram:",  
      choices = names(mtcars)  
    )  
  ),  
  mainPanel(  
    plotOutput(outputId = "plot")  
  )  
)
```

Label to show
above the input

Column used to plot
histogram:

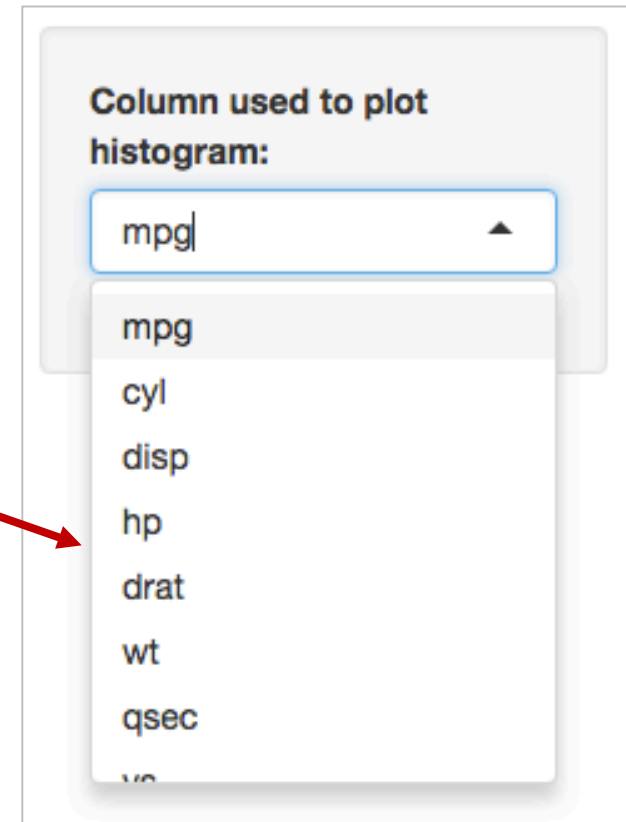
mpg

mpg
cyl
disp
hp
drat
wt
qsec
vs

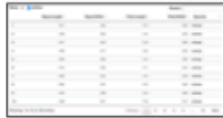
Building a user interface: inputs

```
sidebarLayout(  
  sidebarPanel(  
    selectInput(  
      inputId = "column",  
      label = "Column used to plot histogram:",  
      choices = names(mtcars)  
    )  
  ),  
  mainPanel(  
    plotOutput(outputId = "plot")  
  )  
)
```

Choices to give in
the dropdown menu



Building a user interface: outputs



DT::renderDataTable(expr,
options, callback, escape,
env, quoted)

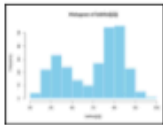


dataTableOutput(outputId, icon, ...)



renderImage(expr, env, quoted, deleteFile)

imageOutput(outputId, width, height, click,
dblclick, hover, hoverDelay, hoverDelayType,
brush, clickId, hoverId, inline)



renderPlot(expr, width, height, res, ..., env,
quoted, func)

plotOutput(outputId, width, height, click,
dblclick, hover, hoverDelay, hoverDelayType,
brush, clickId, hoverId, inline)

data.frame() 3 rows, 4 variables
\$ Sepal.Length: num 5.2 4.9 4.7
\$ Sepal.Width: num 3.5 3.3 3.2

renderPrint(expr, env, quoted, func,
width)

verbatimTextOutput(outputId)



renderTable(expr, ..., env, quoted, func)

tableOutput(outputId)

foo

renderText(expr, env, quoted, func)

textOutput(outputId, container, inline)

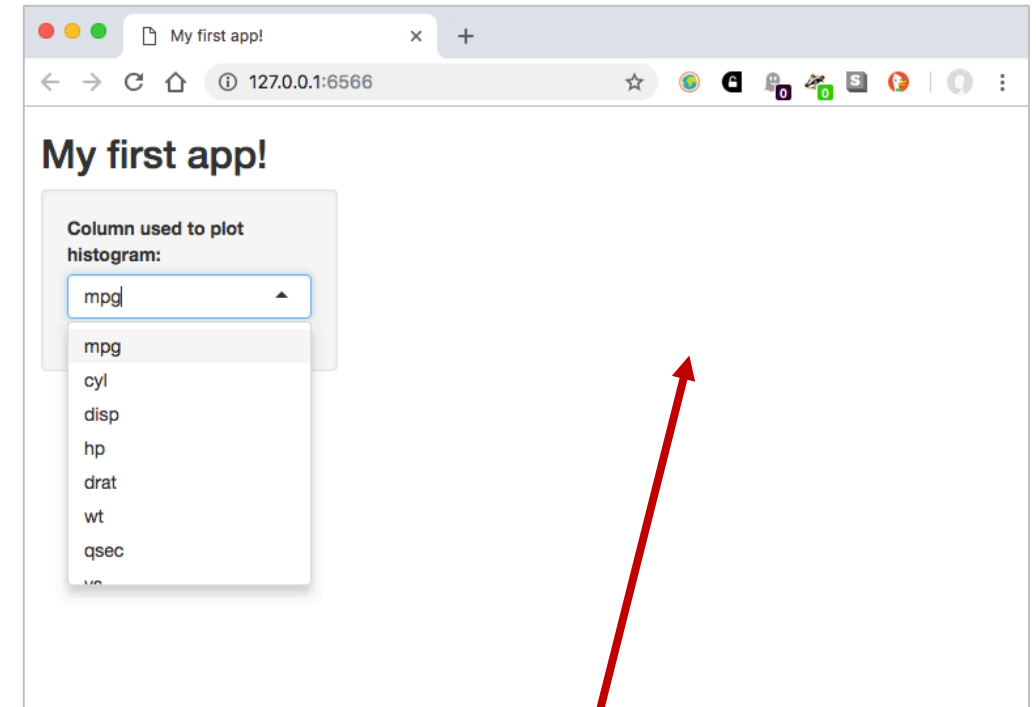


renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, ...)
& **htmlOutput**(outputId, inline, container, ...)

Building a user interface: outputs

```
sidebarLayout(  
  sidebarPanel(  
    selectInput(  
      inputId = "column",  
      label = "Column used to plot histogram:",  
      choices = names(mtcars)  
    )  
  ),  
  mainPanel(  
    plotOutput(outputId = "plot")  
  )  
)
```



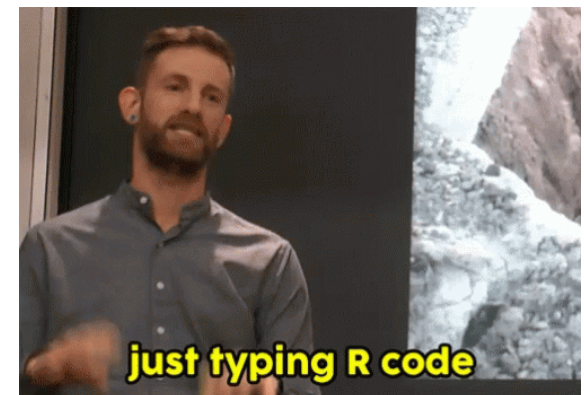
The output element will
go in the main panel

Building a user interface: your turn

Sticking with **3-very-basic-app2.R**, could you:

1. add a **selectInput** element for selecting the **my_data** column to plot?
 - Don't forget: input ID, label and choices
 2. add a **plotOutput** area for the histogram?
 - Don't forget: output ID
- If you have extra time:
Type e.g. `?selectInput` to see available arguments and try changing some of the defaults!

10-15 minutes



Building a user interface: summary

```
ui <- ""

server <- function(input, output) {
}

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

Start with a minimal template, then build on it

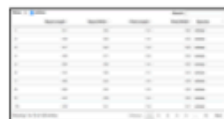
```
...
selectInput(
  inputId = "column",
  label = "Column used to
plot histogram:",
  choices = names(my_data)
)
...
```

***Input()** code:
- define input UI
(- where input UI goes)

```
...
plotOutput(
  outputId = "plot"
)
...
```

***Output()** code:
- what output accepted
(- where output goes)

Writing the server logic: revisiting outputs



DT::renderDataTable(expr,
options, callback, escape,
env, quoted)

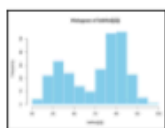


dataTableOutput(outputId, icon, ...)



renderImage(expr, env, quoted, deleteFile)

imageOutput(outputId, width, height, click,
dblclick, hover, hoverDelay, hoverDelayType,
brush, clickId, hoverId, inline)



renderPlot(expr, width, height, res, ..., env,
quoted, func)

plotOutput(outputId, width, height, click,
dblclick, hover, hoverDelay, hoverDelayType,
brush, clickId, hoverId, inline)

data.frame() 3 rows, 4 variables
\$ Sepal.Length: num 5.1 4.9 4.7
\$ Sepal.Width: num 3.5 3.3 3.2

renderPrint(expr, env, quoted, func,
width)

verbatimTextOutput(outputId)



renderTable(expr, ..., env, quoted, func)

tableOutput(outputId)

foo

renderText(expr, env, quoted, func)

textOutput(outputId, container, inline)



renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, ...)

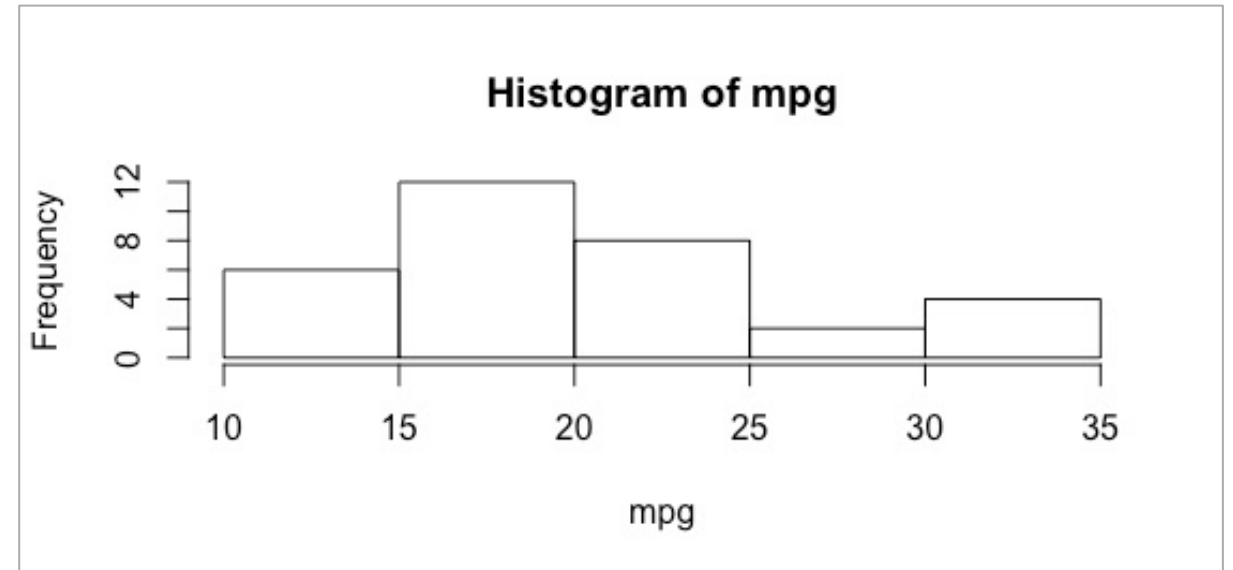
& **htmlOutput**(outputId, inline, container, ...)

Writing the server logic: revisiting outputs

Generic R code:

```
# plot a histogram of mtcars$mpg

hist(
  x = mtcars[["mpg"]],
  main = paste("Histogram of", "mpg"),
  xlab = "mpg"
)
```



Writing the server logic: revisiting outputs

Generic R code:

```
# plot a histogram of mtcars$mpg

hist(
  x = mtcars[["mpg"]],
  main = paste("Histogram of", "mpg"),
  xlab = "mpg"
)
```

Shiny server code:

```
# shiny output version
output$plot <- renderPlot({
  hist(
    x = mtcars[["mpg"]],
    main = paste("Histogram of", "mpg"),
    xlab = "mpg"
  )
})
```

Writing the server logic: revisiting outputs

What kind of output (matches type in UI code)

relevant ui code:

```
mainPanel(  
  plotOutput(outputId = "plot")  
)
```

relevant server code:

```
output$plot <- renderPlot({  
  hist(  
    x = mtcars[["mpg"]],  
    main = paste("Histogram of", "mpg"),  
    xlab = "mpg"  
  )  
})
```

Writing the server logic: revisiting outputs

Where the output should appear (matches output ID in UI code)

relevant ui code:

```
mainPanel(  
  plotOutput(outputId = "plot")  
)
```

relevant server code:

```
output$plot <- renderPlot({  
  hist(  
    x = mtcars[["mpg"]],  
    main = paste("Histogram of", "mpg"),  
    xlab = "mpg"  
  )  
})
```

Writing the server logic: revisiting outputs

``output`` is a named list, values can be stored with the help of ``$``

relevant ui code:

```
mainPanel(  
  plotOutput(outputId = "plot")  
)
```

relevant server code:

```
output$plot <- renderPlot({  
  hist(  
    x = mtcars[["mpg"]],  
    main = paste("Histogram of", "mpg"),  
    xlab = "mpg"  
  )  
})
```

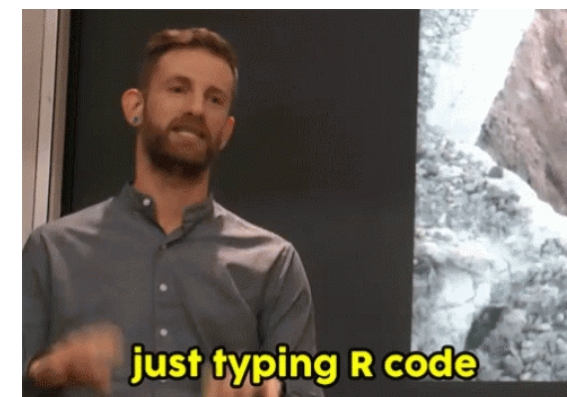
Writing the server logic: your turn

Without looking at previous slides, could you:

1. Edit the server logic as explained in **5-app-with-server.R**

- If you have extra time:
 - What other attributes in the output might you want to change?
 - What kind of inputs would you need?
 - Which input elements might be suitable for them? (see [this widget gallery](#) for ideas)

3-5 minutes



Writing the server logic: linking inputs and outputs

Incorporating the input in the output code:

- what does our input data look like?
- can't type into console while shiny app is running..
- **print()** and **cat()** are your friends!

Let's try with **6-app-with-server2.R**

Reactivity: notice the code is rerun every time we pick a new input
-> this is why the output will update as the input changes

Writing the server logic: linking inputs and outputs

``input`` is also a named list; values can be retrieved with ``$``

relevant ui code:

```
selectInput(  
  
  inputId = "column",  
  label = "Column used to  
plot histogram:",  
  choices = names(mtcars)  
  
)
```

linked to input

```
output$plot <- renderPlot({  
  hist(  
    x = mtcars[[input$column]],  
    main = paste("Histogram of", input$column),  
    xlab = input$column  
  )  
})
```


Writing the server logic: linking inputs and outputs

Which input to retrieve data from (matches input ID in UI code)

```
# relevant ui code:
```

```
selectInput(
```

```
  inputId = "column",
```

```
  label = "Column used to  
plot histogram:",
```

```
  choices = names(mtcars)
```

```
)
```

```
# linked to input
```

```
output$plot <- renderPlot({
```

```
  hist(
```

```
    x = mtcars[[input$column]],
```

```
    main = paste("Histogram of", input$column),
```

```
    xlab = input$column
```

```
  )
```

```
})
```

Writing the server logic: your turn

Without looking at previous slides, could you:

1. Edit the server logic in **6-app-with-server2.R** so the output changes depending on the input
- If you have extra time:
 - What other attributes in the output might you want to change?
 - What kind of inputs would you need?
 - Which input elements might be suitable for them? (see [this widget gallery](#) for ideas)

3-5 minutes



Building a user interface: summary

```
output$plot <- renderPlot({  
  
  hist(x =  
    mtcars[[input$col]]  
  )  
  
})
```

Use input values in calculation function with *input\$input_id*

```
output$plot <- renderPlot({  
  
  hist(x =  
    mtcars[[input$col]]  
  )  
  
})
```

Wrap calculation function with *render**().

```
output$plot <- renderPlot({  
  
  hist(x =  
    mtcars[[input$col]]  
  )  
  
})
```

Assign function output to *output\$output_id*

Adding complexity: reading data from a file

see [8-csv-data.R](#)

Before:

```
# "read in" data  
my_data <- mtcars
```

After:

```
# read in data  
my_data <-  
  read.csv("../data/mtcars.csv", row.names = 1)
```

Adding complexity: reading data from a file

Before:

```
# "read in" data  
my_data <- mtcars
```

After:

```
# read in data  
my_data <-  
  read.csv("../data/mtcars.csv", row.names = 1)
```

First column is row names



Function to read file



Path to file

Adding complexity: replacing inputs/outputs

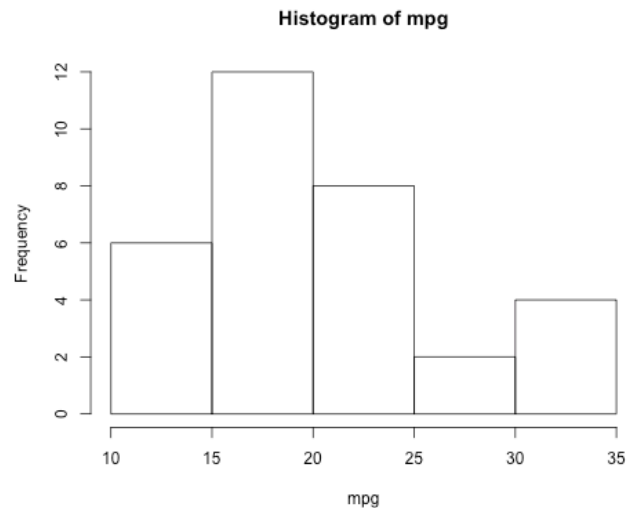
see [9-replace-io.R](#)

Before:

My first app!

Column used to plot histogram:

mpg ▼



After:

My more complicated app!

Columns to display:

- ☒ mpg
- ☒ cyl
- ☒ disp
- ☒ hp
- ☒ drat
- ☒ wt
- ☒ qsec
- ☒ vs
- ☒ am
- ☒ gear
- ☒ carb

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
21.00	6.00	160.00	110.00	3.90	2.62	16.46	0.00	1.00	4.00	4.00
21.00	6.00	160.00	110.00	3.90	2.88	17.02	0.00	1.00	4.00	4.00
22.80	4.00	108.00	93.00	3.85	2.32	18.61	1.00	1.00	4.00	1.00
21.40	6.00	258.00	110.00	3.08	3.21	19.44	1.00	0.00	3.00	1.00
18.70	8.00	360.00	175.00	3.15	3.44	17.02	0.00	0.00	3.00	2.00
18.10	6.00	225.00	105.00	2.76	3.46	20.22	1.00	0.00	3.00	1.00
14.30	8.00	360.00	245.00	3.21	3.57	15.84	0.00	0.00	3.00	4.00
24.40	4.00	146.70	62.00	3.69	3.19	20.00	1.00	0.00	4.00	2.00
22.80	4.00	140.80	95.00	3.92	3.15	22.90	1.00	0.00	4.00	2.00
19.20	6.00	167.60	123.00	3.92	3.44	18.30	1.00	0.00	4.00	4.00

Adding complexity: replacing inputs/outputs

Change output

relevant ui code:

```
mainPanel(  
  tableOutput(outputId = "table")  
)
```

relevant server code:

```
output$table <- renderTable({  
  mtcars[1:10]  
})
```

Adding complexity: replacing inputs/outputs

Change output

relevant ui code:

```
mainPanel(  
  tableOutput(outputId = "table")  
)
```

relevant server code:

```
output$table <- renderTable(  
  mtcars[1:10]  
)
```


Adding complexity: replacing inputs/outputs

How do you plan to transform the data?

relevant ui code:

```
checkboxGroupInput(  
  inputId = "columns",  
  label = "Columns to display:",  
  choices = names(my_data),  
  selected = names(my_data)  
)
```

relevant server code:

```
output$table <- renderTable({  
  
  mtcars[1:10, 1:3, drop = FALSE]  
  
})
```

Pick which column(s) to show



Adding complexity: replacing inputs/outputs

What's the corresponding input for that?

relevant ui code:

```
checkboxGroupInput(  
  inputId = "columns",  
  label = "Columns to display:",  
  choices = names(my_data),  
  selected = names(my_data)  
)
```

relevant server code:

```
output$table <- renderTable({  
  
  mtcars[1:10, 1:3, drop = FALSE]  
  
})
```

Adding complexity: replacing inputs/outputs

What does your input look like?

relevant ui code:

```
checkboxGroupInput(  
  inputId = "columns",  
  label = "Columns to display:",  
  choices = names(my_data),  
  selected = names(my_data)  
)
```

relevant server code:

```
output$table <- renderTable({  
  
  cat(input$columns)  
  class(input$columns)  
  
  mtcars[1:10, 1:3, drop = FALSE]  
  
})
```

Adding complexity: replacing inputs/outputs

Link input to output

relevant ui code:

```
checkboxGroupInput(  
  inputId = "columns",  
  label = "Columns to display:",  
  choices = names(my_data),  
  selected = names(my_data)  
)
```

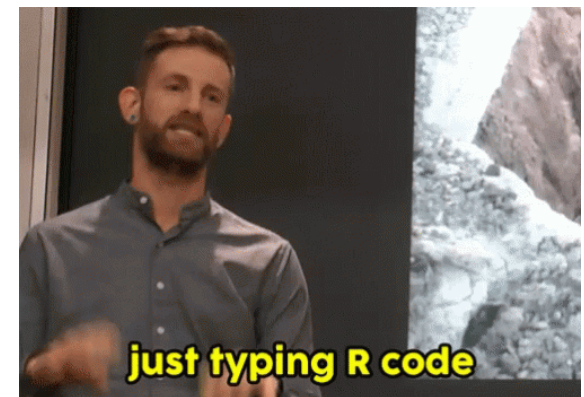
relevant server code:

```
output$table <- renderTable({  
  
  mtcars[1:10, input$columns,  
    drop = FALSE]  
  
})
```

Adding complexity: your turn!

- Multiple inputs
 - using two variables and adding shape/colour option
 - allowing users to add their own text for title/labels
 - giving a choice between base and ggplot
- Multiple/different outputs
 - add table of summary statistics
 - add a logo and text summarising data
- Different data
 - read in your own data
 - use the survey data

Until 8:10pm



Reminder: inputs



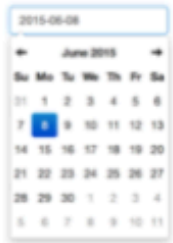
actionButton(inputId, label, icon, ...)

Link

actionLink(inputId, label, icon, ...)



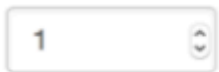
checkboxInput(inputId, label, value)



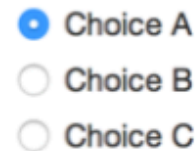
dateInput(inputId, label, value, min, max, format, startview, weekstart, language)

Choose File

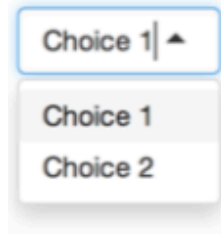
fileInput(inputId, label, multiple, accept)



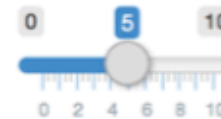
numericInput(inputId, label, value, min, max, step)



radioButtons(inputId, label, choices, selected, inline)



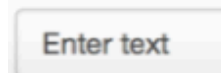
selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also **selectizeInput()**)



sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)

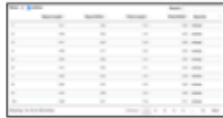


submitButton(text, icon)
(Prevents reactions across entire app)



textInput(inputId, label, value)

Reminder: revisiting outputs



DT::renderDataTable(expr,
options, callback, escape,
env, quoted)

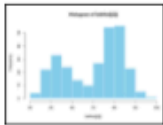


dataTableOutput(outputId, icon, ...)



renderImage(expr, env, quoted, deleteFile)

imageOutput(outputId, width, height, click,
dblclick, hover, hoverDelay, hoverDelayType,
brush, clickId, hoverId, inline)



renderPlot(expr, width, height, res, ..., env,
quoted, func)

plotOutput(outputId, width, height, click,
dblclick, hover, hoverDelay, hoverDelayType,
brush, clickId, hoverId, inline)

data.frame() 3 rows, 4 variables:
\$ Sepal.Length: num 5.2 4.9 4.7
\$ Sepal.Width : num 3.5 3.3 3.2

renderPrint(expr, env, quoted, func,
width)

verbatimTextOutput(outputId)



renderTable(expr, ..., env, quoted, func)

tableOutput(outputId)

foo

renderText(expr, env, quoted, func)

textOutput(outputId, container, inline)



renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, ...)

& **htmlOutput**(outputId, inline, container, ...)

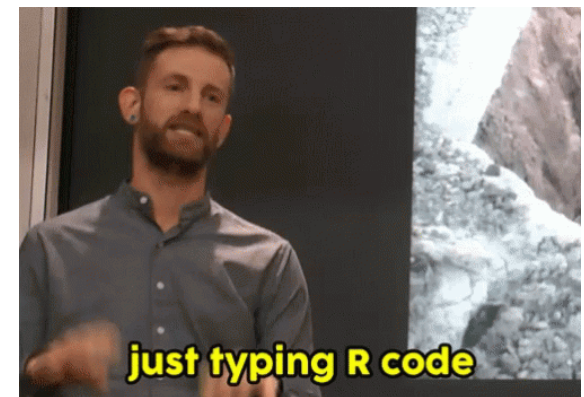
Adding complexity: your turn (more advanced)!

Do you plan to have users upload their data onto your app?

Example in: [10-uploaded-data.R](#)

Requires **reactive()**, **observeEvent()** and **req()**

Until 8:10pm



Tips

- My general workflow
 1. Analyse/visualise the data until you're clear what you want to show
 2. Write functions that work outside of a shiny app
 3. Imagine/draw the layout, then write the UI code (provides structure)
 4. Edit the functions into server code
 5. Repeat if you want to add more features!
- Cmd/ Ctrl + I to auto-indent for readability
- **cat()**, **print()** and **verbatimText** for simple debugging



USING
THE RIGHT TOOLS
SAVES
A LOT OF PAIN

Tools/resources that make our lives easier

- shiny.rstudio.com
 - [gallery](#)
 - [tutorial](#), [cheatsheet](#)
 - [reference](#) or type `?<function_name>` in your console
 - [articles](#), e.g. [debugging shiny apps](#) (quite different from normal scripts)
- [Shiny section](#) of Rstudio community
- More from me:
 - shiny “extensions” [talk](#) and corresponding notes for more tools!
 - feel free to ask me questions: [@s owla](#) on twitter :)

Tools/resources that make our lives easier: your turn!

- I (and I think many others) would love to learn from you! :)
- [R-Ladies chapters](#)
- [R user groups](#)
- [R-Ladies rotating curator twitter account](#)
- Put your code on [github](#)
- Share your ideas/code/writing on [#rstats](#) twitter
- So many more ways!!

On-going! :)

