

VISUALIZATION & SHINY

CEPE GENES

Author: Frédéric Logé

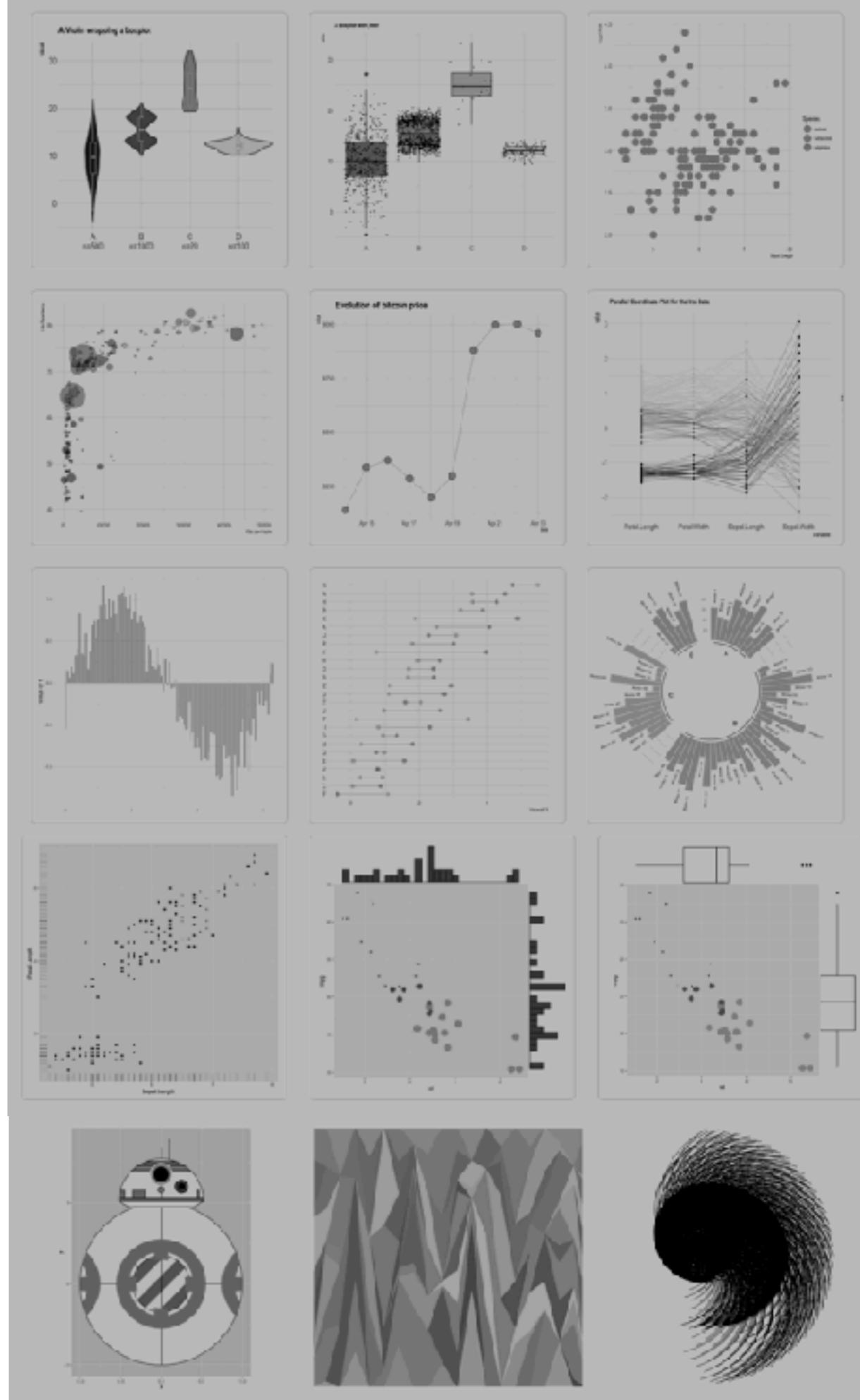
Date: Nov 2021

Program

- Visualization : building graphs with ggplot2
 - Why ggplot2 ?
 - Grammar of Graphics : data, aesthetics, geometry, design, facetting
 - Going through different geometries [practice]
 - Improving design [practice]
 - Practice
- Building dashboard and web apps with Shiny
 - Range of possibilities
 - Starting off with an example, UI vs Server [practice]
 - Playing with basic inputs and outputs [practice]
 - Event reactivity [practice]
 - Handling data properly [practice]
 - Practice

VISUALIZATION

BUILDING GRAPHS WITH



Why ggplot2 ?

R packages - what is out there ??

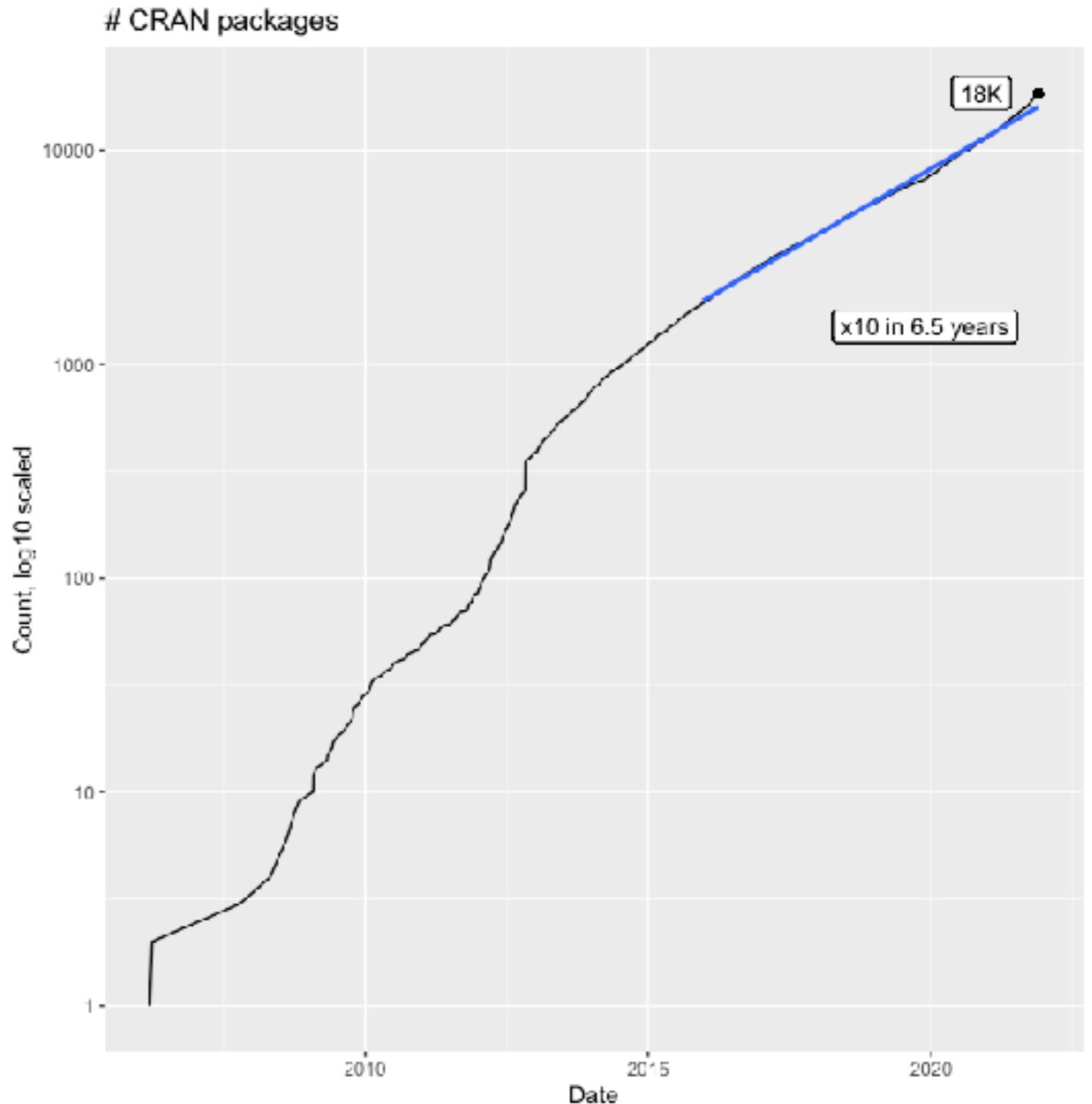
.....

A lot of base methods to know

So many packages (see plot beside), it's so hard to keep up.

Search + stack overflow + R vignettes can help get some clues as to what to use.

Since ~10 years, a new standard package arose for anything data related : the tidyverse package



And that's not counting other package repositories and packages hosted on GitHub !

Why ggplot2 ?

R packages - the tidyverse

tidyverse

opinionated collection of R packages designed for data science

`library(tidyverse)`

- **ggplot2**: data visualization
- **dplyr**: data wrangling
- **tidyr**: data tidying
- **readr**: data reading/writing
- **forcats**: working with factors
- **stringr**: working with strings
- **tibble**: modern data frames
- **purrr**: functional programming

`install.packages(tidyverse)` above + a few more



Why ggplot2 ?

R packages - the dataviz landscape

ggplot2



The Standard package for data visualization

3 300K Monthly Downloads

Extremely readable syntax (grammar of graphics)

Cost of entry : medium

Maintained by RStudio and co, part of the tidyverse

#1 choice of R community and the standard of graphics

can be re-used in Python via nineplot

so many packages are built as add-ons of ggplot2 :



Base plots

Installed by default

Hard/customized plots : nightmare

Syntax quickly becomes unreadable

Not the nicest designs

Lattice, DiagrammeR, igraph, ...

so many packages, often doing single-type plot

far less used than ggplot2 and extensions

most of what they do can be done via ggplot2 and extensions

often require less ‘work’ than ggplot2 for quick tests

Package	Nb monthly downloads
Cowplot	280K
Lattice	155K
LatticeExtra	240K
ggrepel*	500K

*ggplot2 add-on, adding nice annotation to graph

Plotly, Echarts4, ...

Cross-platform

Very pretty outputs

Interactivity

Similar grammar to ggplot2

Much less support



300K Monthly
Downloads



3K Monthly
Downloads

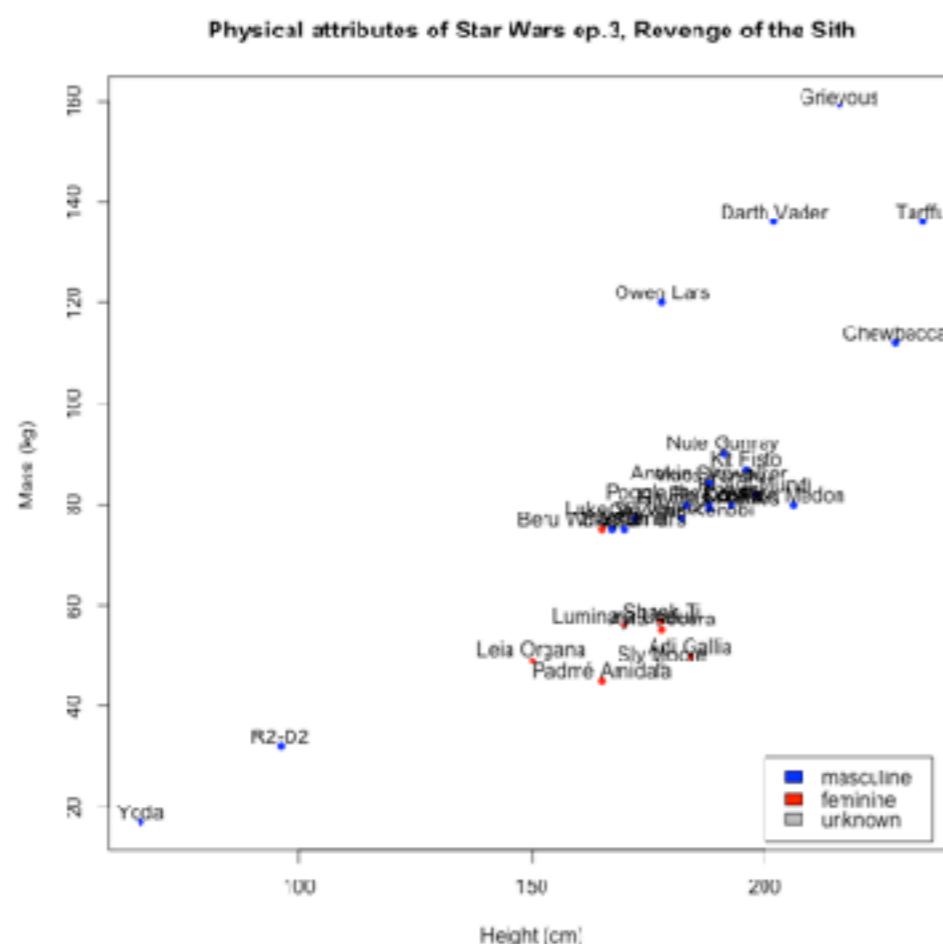
Why ggplot2 ?

From base to ggplot2 - radical change of grammar, for the best

.....

base

```
1 l = c('masculine' = 'blue', 'feminine' = 'red')
2 plot(x = starwars_ep3$height, y = starwars_ep3$mass,
3       col = map_var(l, starwars_ep3$gender, 'gray'), pch = 20,
4       main = 'Physical attributes of Star Wars ep.3',
5       xlab = 'Height (cm)', ylab = 'Mass (kg)')
6 text(x = starwars_ep3$height, y = starwars_ep3$mass + 2,
7       labels = starwars_ep3$name)
8 legend(x = 200, y = 30, fill = c(as.character(l), 'gray'),
9        legend = c(names(l), 'unknown'))
```



How ggplot2 ?

Key elements of the Grammar of Graphics

INIT GGPlot
Specifying a Tibble as data

AESTHETICS
What to put in x, y, color
(contour), fill (filling
color), shape, size

GEOMETRY
geom_point
geom_line
geom_hist
geom_boxplot
...

```
1 library(tidyverse)
2 library(ggrepel)
3
4 data("starwars")
5
6 starwars_ep3 = starwars %>%
7   filter(map_lgl(films, function(x) "Revenge of the Sith" %in% x))
8
9 ggplot(data = starwars_ep3) +
10   aes(x = height, y = mass, label = name, col = gender) +
11   geom_point() +
12   geom_label_repel() +
13   labs(title = 'Physical attributes of Star Wars ep.3',
14        x = 'Height (cm)', y = 'Mass (kg)') +
15   theme(text = element_text(size = 20), legend.position = 'bottom')
```

DESIGN
labs, theme

FACETTING
split graphic by
categories

+ Operator to combine
ggproto objects

How ggplot2 ?

Practice & Tips

Some Ressources



<https://raw.githubusercontent.com/rstudio/cheatsheets/main/pngs/data-visualization.png>

<https://ggplot2.tidyverse.org>

<https://www.r-graph-gallery.com>

<https://www.r-pkg.org/search.html?q=visualization>

Some tips & common issues

- > add interactivity easily : transform ggplot to plotly graph
- > check error message
- > color/fill are often confused
- > R interprets code without regard of line indexes, so make sure you don't have open code before, and that all '+' signs are present

LET's PRACTICE !

DASHBOARDS & WEBAPPS

WITH



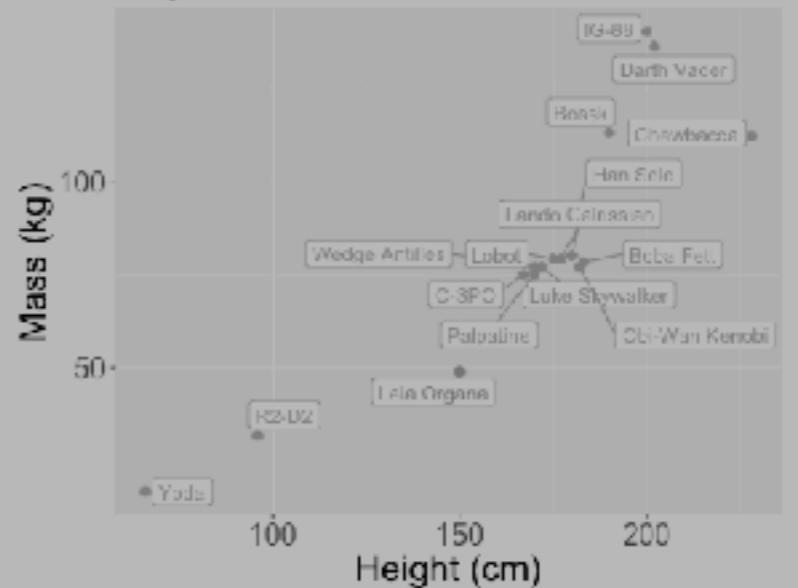
Star Wars characters

Film

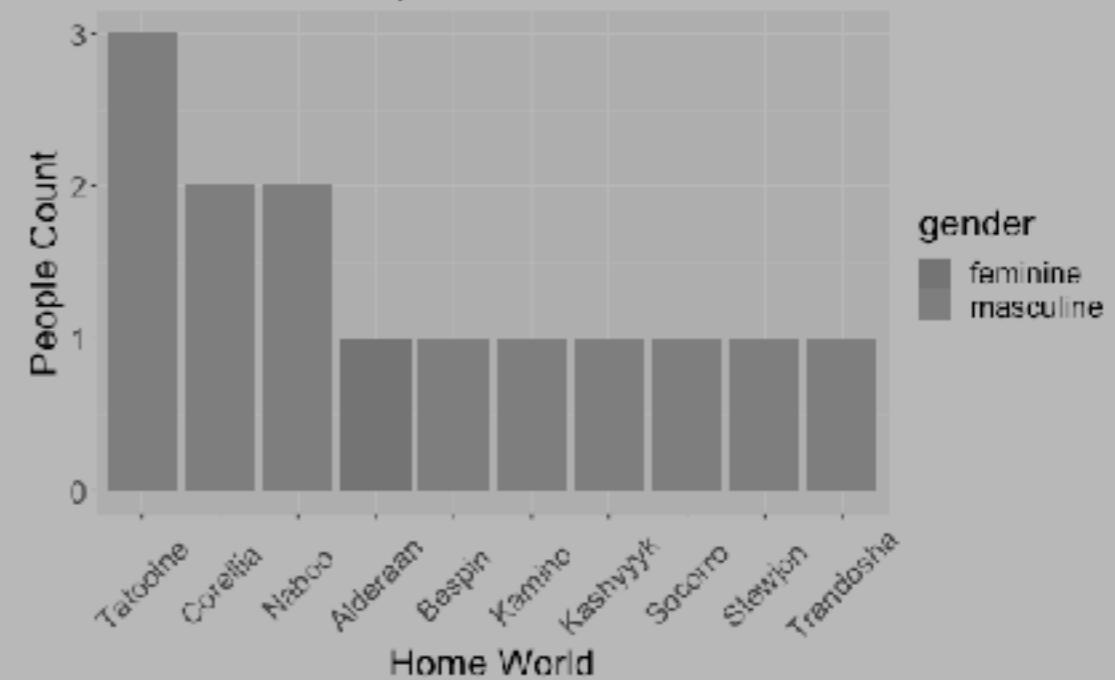
The Empire Strikes Back

	name	birth_year	species
1	Luke Skywalker	19	Human
2	C-3PO	112	Droid
3	R2-D2	33	Droid
4	Darth Vader	41.9	Human

Physical attributes of Star Wars

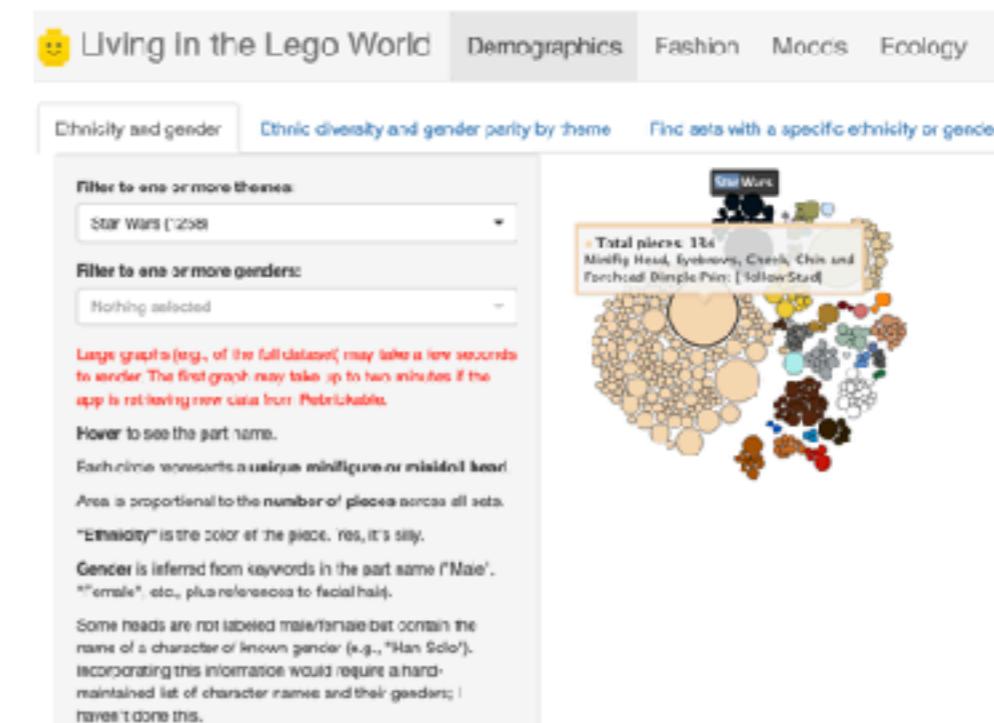
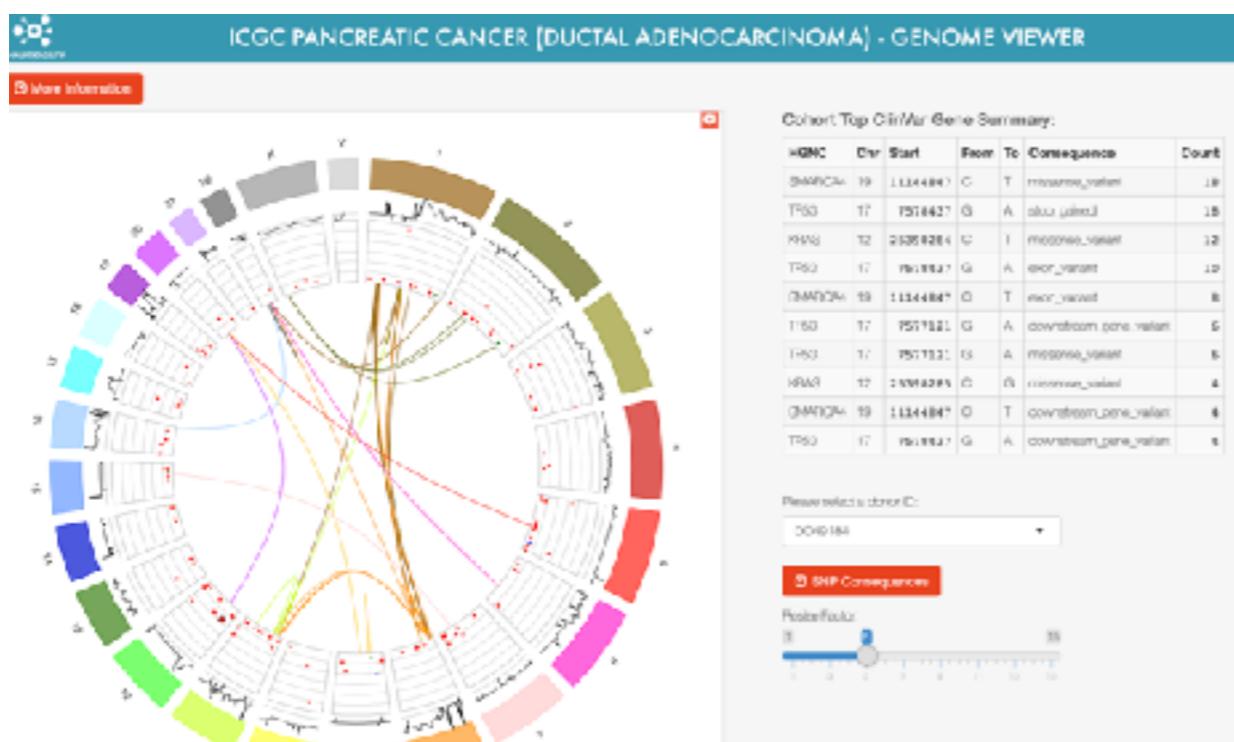
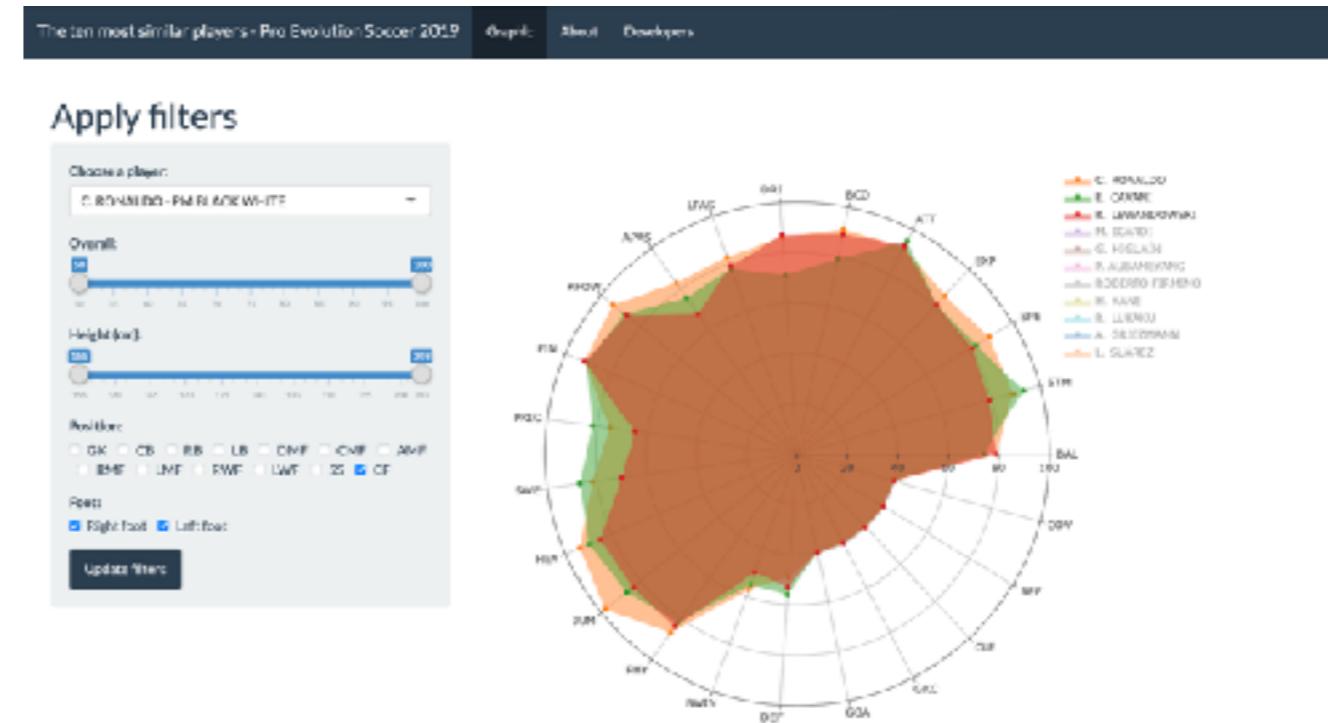
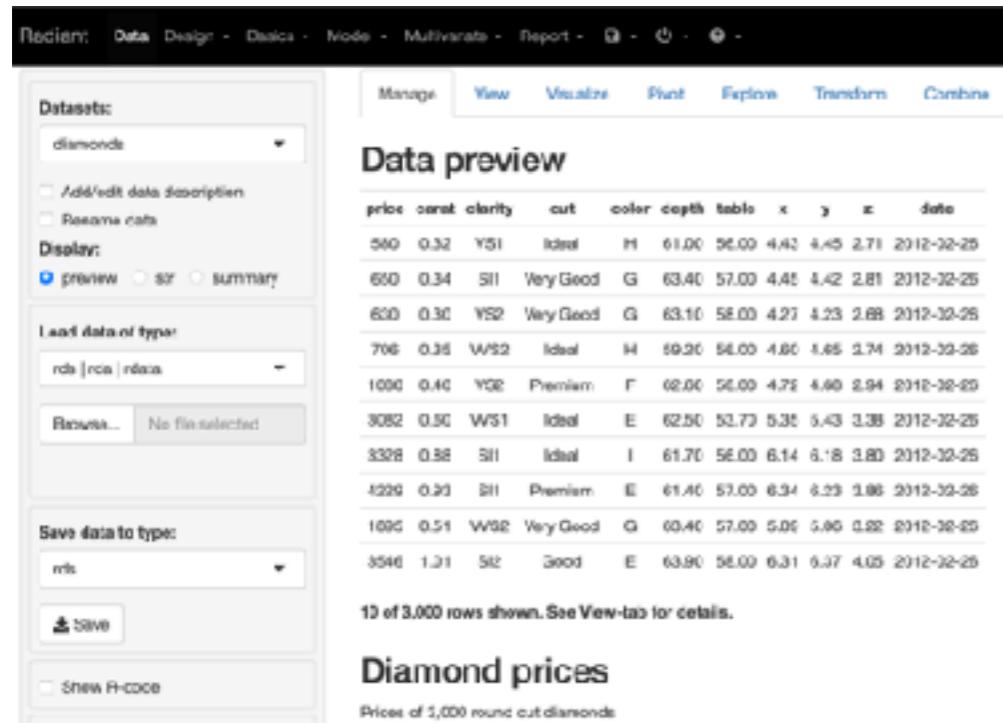


Home World representation



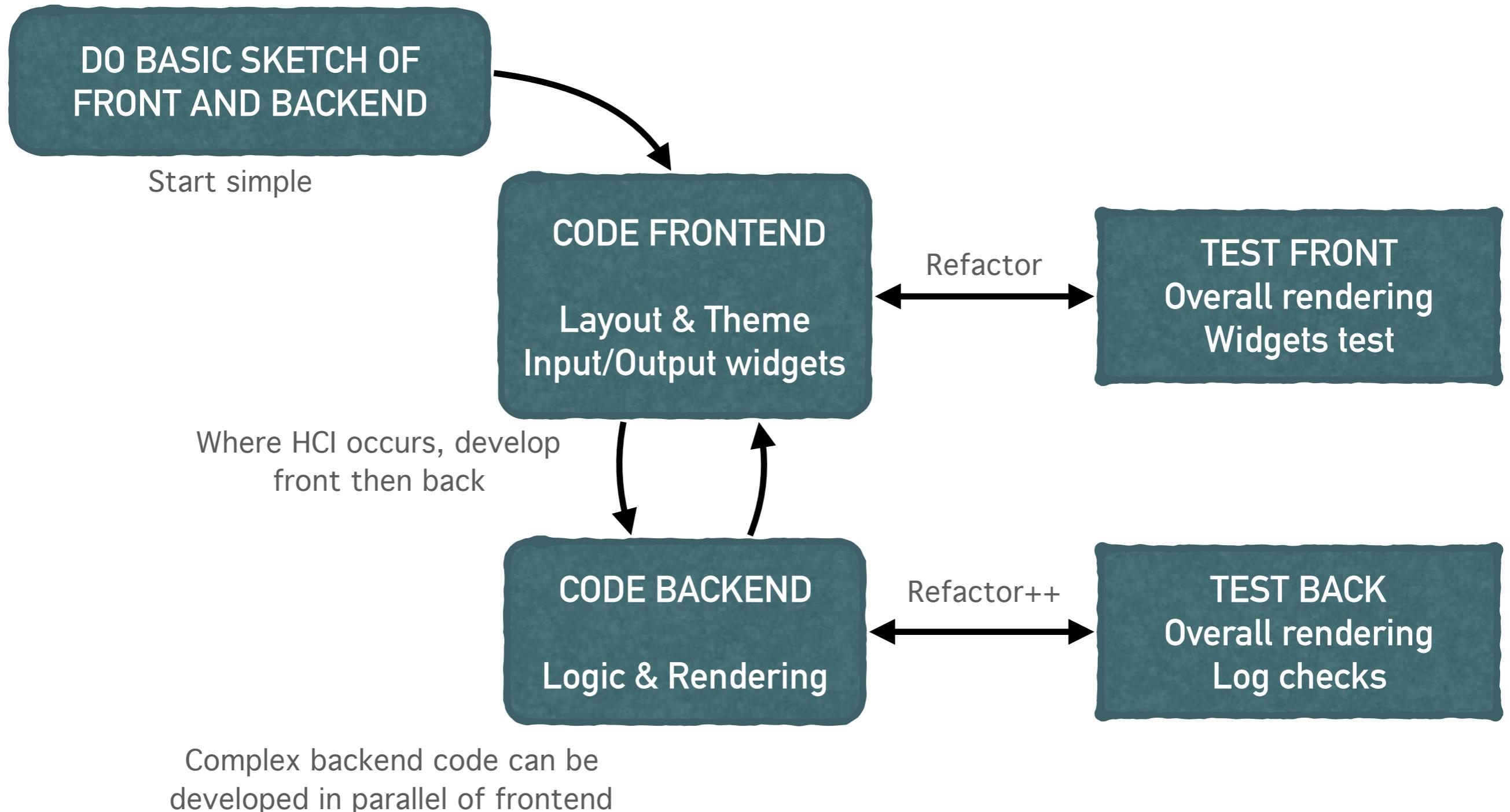
Objective

Learn how to build such beautiful apps, with R code.



Generic process for building the app

.....



Geyser example

Follow these steps :

1. File > New File > Shiny Web App
2. A popup appears : put a random name (modifiable after), and click on `Single file`
3. A folder is created with an `app` R file. Open it. `<!> install.packages('shiny')`
4. Click on the RunApp button to see the result !

User Interface

```
12 # Define UI for application that draws a histogram
13 ui <- fluidPage(
14
15   # Application title
16   titlePanel("Old Faithful Geyser Data"),
17
18   # Sidebar with a slider input for number of bins
19   sidebarLayout(
20     sidebarPanel(
21       sliderInput("bins",
22                 "Number of bins:",
23                 min = 1,
24                 max = 50,
25                 value = 30)
26     ),
27
28   # Show a plot of the generated distribution
29   mainPanel(
30     plotOutput("distPlot")
31   )
32 )
33 )
```

Check value of ui to see html code

Server

```
35 # Define server logic required to draw a histogram
36 server <- function(input, output) {
37
38   output$distPlot <- renderPlot({
39     # generate bins based on input$bins from ui.R
40     x <- faithful[, 2]
41     bins <- seq(min(x), max(x), length.out = input$bins + 1)
42
43     # draw the histogram with the specified number of bins
44     hist(x, breaks = bins, col = 'darkgray', border = 'white')
45   })
46 }
47
48 # Run the application
49 shinyApp(ui = ui, server = server)
```

LAYOUT INPUT OUTPUT

Three basic components with
which you'll be working

Plan

- Playing with basic inputs and outputs
- Event reactivity
- Handling data properly
- Create your own Shiny App !

Input widgets

Inputs are called in server function with `input$some_input_id`, as defined in the UI

UI input
actionButton
checkboxGroupInput
checkboxInput
dateInput
dateRangeInput
fileInput
helpText
numericInput
radioButtons
selectInput
sliderInput
textInput

Basic widgets

Buttons

Action

Submit

Date range

2017-06-21 to 2017-06-21

Single checkbox

Choice A

Checkbox group

- Choice 1
 Choice 2
 Choice 3

Date input

2014-01-01

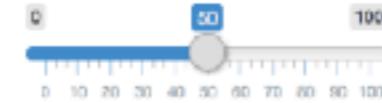
Radio buttons

- Choice 1
 Choice 2
 Choice 3

Select box

Choice 1

Sliders



Numeric input

1

Text input

Enter text...

Check out <http://shinyapps.dreamrs.fr/shinyWidgets/> for advanced selection of widgets

Handling outputs

Outputs are specified in UI (e.g. `textOutput`) and the rendering is specified in server
(e.g. `output$some_output_id <- renderOutput({ ... })`)

Creating :	In UI	In server
Raw HTML	<code>htmlOutput('id')</code> <code>uiOutput('id')</code>	<code>output\$id = renderUI({ ... })</code>
Text	<code>textOutput('id')</code> <code>verbatimTextOutput('id')</code>	<code>output\$id = renderText({ ... })</code> <code>output\$id = renderPrint({ ... })</code>
Table	<code>tableOutput('id')</code>	<code>output\$id = renderTable({ ... })</code>
DataTable	<code>dataTableOutput('id')</code>	<code>output\$id = renderDataTable({ ... })</code>
Image	<code>imageOutput('id')</code>	<code>output\$id = renderImage({ ... })</code>
Plot	<code>plotOutput('id')</code>	<code>output\$id = renderPlot({ ... })</code>
Plotly	<code>plotlyOutput('id')</code>	<code>output\$id = renderPlotly({ ... })</code>

Render your plots interactive, with hovering and clicking options

<https://shiny.rstudio.com/articles/plot-interaction.html>

The need for refactoring

Before

```
18 server = shinyServer(function(input, output) {  
19   df <- reactive({ starwars %>% filter(map_lgl(films, function(x) input$film %in% x)) })  
20  
21   output$table1 <- renderDataTable({  
22     datatable(head(df(),c('name', 'birth_year', 'species')), 40,  
23       options = list(dom = 't'))  
24   })  
25  
26   output$physic1 <- renderPlot({  
27     h = df()  
28     print(h)  
29     ggplot(data = h) +  
30       aes(x = height, y = mass, label = name, col = gender) +  
31       geom_point(cex = 2) +  
32       geom_label_repel() +  
33       coord_fixed() +  
34       labs(title = "Physical attributes of Star Wars",  
35             x = "Height (cm)", y = "Mass (kg)") +  
36             theme(text = element_text(size = 20), legend.position = "bottom")  
37   })  
38  
39  
40   output$physic2 <- renderPlot({  
41     hh = df() %>%  
42       filter(is.na(homeworld) == FALSE) %>%  
43       count(homeworld, gender) %>%  
44       group_by(homeworld) %>%  
45       mutate(nn = sum(n)) %>%  
46       slice_max(order_by = nn, n = 10)  
47     ggplot(data = hh) +  
48       aes(x = reorder(homeworld, n), y = n, fill = gender) +  
49       geom_bar(stat = "identity") +  
50       labs(x = "Home World", y = "People Count", title = "Home World representation") +  
51         theme(text = element_text(size = 20),  
52               axis.text.x = element_text(vjust = 0.5, angle = 45))  
53   })  
54 }  
55 })
```

After

```
1 height_vs_mass_graph <- function(filtered_df){  
2  
3 homeworld_count <- function(filtered_df){  
4  
5  
6  
7 server = shinyServer(function(input, output) {  
8   df <- reactive({ starwars %>% filter(map_lgl(films, function(x) input$film %in% x)) })  
9  
10  output$table1 <- renderDataTable({  
11    datatable(head(df(),c('name', 'birth_year', 'species')), 40,  
12      options = list(dom = 't'))  
13  })  
14  
15  output$physic1 <- renderPlot({ height_vs_mass_graph(df()) })  
16  
17  output$physic2 <- renderPlot({ homeworld_count(df()) })  
18 })  
19  
20 })  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61 })
```

Graph code and data process can render overall code readability a nightmare => create outside app functions for increased readability + testing

The app can be organized in different ways :
► functions called can be in separate files, sourced at the beginning
► ui and server can be defined in separate files
You can even integrate it in markdown document !!

Introducing reactivity

	Reactivity functions	Objective	Output
Trigger	observe	Do some operations whenever a called input changes	None
	observeEvent	Do some operations whenever a trigger input is modified	None
	isolate	Isolate an input such that reactivity is not affected by change of its value	None
Data handling	reactive	Compute some data whenever a called input changes	Function
	eventReactive	Compute some data whenever a trigger input is modified	Function
	reactiveValues	Store data that you wish to be accessible and modifiable from any part of the server -> best way to handle loaded data on app	None

Check out <https://rstudio.github.io/reactlog/> to test your reactivity processes

Reactivity - Trigger

observe({}) - do some operations whenever a called input changes, no output

.....

```
4 ui <- fluidPage(  
5  textInput(inputId = "free_input", label = "Free Input"),  
6   selectInput(inputId = "film", label = "Film",  
7               choices = unique(unlist(starwars$films)))  
8 )  
9  
10 random_code <- function(){  
11   paste0(sample(letters, 10), collapse = '')  
12 }  
13  
14 server <- function(input, output, session) {  
15  
16   observe({  
17     print(input$film)  
18     updateSliderInput(inputId = "free_input",  
19                         session = session,  
20                         value = random_code())  
21   })  
22  
23 }  
24  
25 shinyApp(ui, server)
```

LAYOUT

INPUT

OUTPUT

REACTIVITY

Reactivity - Trigger

observeEvent({}) : do some operations whenever a specific input is modified, no output

.....

```
3 ui <- fluidPage(  
4   actionButton("doMagicTrick", "Show something"),  
5   textOutput("myText")  
6 )  
7  
8 server <- function(input, output) {  
9   observeEvent(input$doMagicTrick, {  
10     output$myText <- renderText({  
11       paste0('You are ...', sample(starwars$name, 1), '!')  
12     })  
13   })  
14 }  
15  
16 shinyApp(ui, server)
```

LAYOUT

INPUT

OUTPUT

REACTIVITY

Reactivity - Trigger / or not

isolate({}) - stop input modification from causing changes

.....

```
4 ui <- fluidPage(  
5  textInput(inputId = "free_input", label = "Free Input"),  
6   selectInput(inputId = "film", label = "Film", choices = unique(unlist(starwars$films))),  
7   textOutput('output_text')  
8 )  
9  
10 server <- function(input, output) {  
11  
12   output$output_text <- renderText({  
13     paste0(isolate(input$film), " - ", input$free_input)  
14   })  
15  
16 }  
17  
18 shinyApp(ui, server)
```

LAYOUT

INPUT

OUTPUT

REACTIVITY

Reactivity - Data handling

reactive({}) - update dataset required for several outputs when input changes

```
5 ui <- fluidPage(  
6   selectInput(inputId = "film", label = "Film", choices = unique(unlist(starwars$films))),  
7   textOutput('genericInfo'),  
8   br(),  
9   h5('First 4 characters :'),  
10  dataTableOutput("myTable")  
11 )  
12  
13 server <- function(input, output, session) {  
14  
15   df <- reactive({  
16     starwars %>%  
17     filter(map_lgl(films, function(x) input$film %in% x))  
18   })  
19  
20   output$genericInfo <- renderText({  
21     paste0('Nb of characters : ', nrow(df()))  
22   })  
23  
24   output$myTable <- renderDataTable({  
25     datatable(head(df(),c('name', 'birth_year', 'species')), 4),  
26     options = list(dom = 't'))  
27   })  
28  
29  
30 }  
31  
32 shinyApp(ui, server)
```

Note that reactive outputs a function
callable anywhere in server

LAYOUT

INPUT

OUTPUT

REACTIVITY

Reactivity - Data handling

eventReactive({}) : when a single input is modified, output new data

```
3 ui <- fluidPage(  
4   headerPanel("Example eventReactive"),  
5  
6   mainPanel(  
7  
8     # input field  
9    textInput("user_text", label = "Enter some text:", placeholder = "Please enter some text."),  
10  
11    # submit button  
12    actionButton("submit", label = "Submit"),  
13  
14    # display text output  
15    textOutput("text"))  
16 )  
17  
18 server <- function(input, output) {  
19  
20   # reactive expression  
21   text_reactive <- eventReactive(input$submit, {  
22     input$user_text  
23   })  
24  
25   # text output  
26   output$text <- renderText({  
27     text_reactive()  
28   })  
29 }
```

Note that eventReactive outputs a function callable anywhere in server

LAYOUT INPUT OUTPUT REACTIVITY

Reactivity - Data handling

reactiveValues({}) : handling data

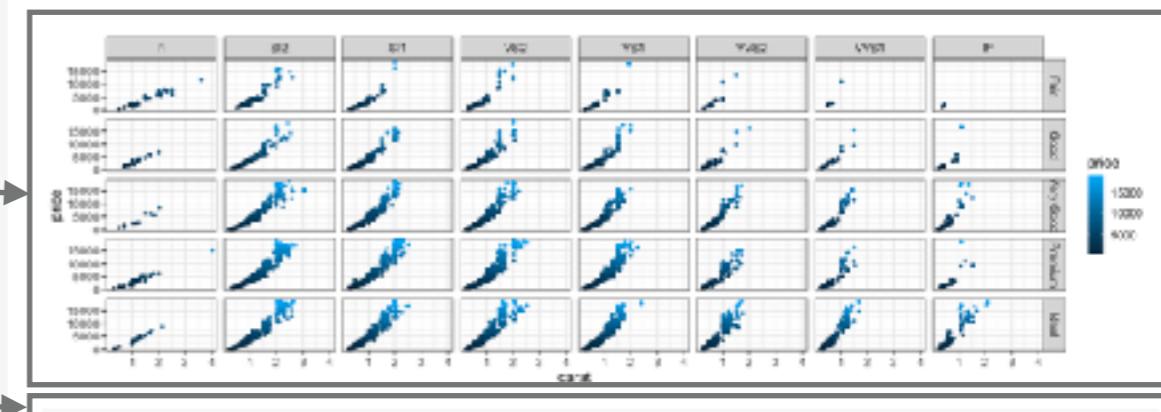
```
3 ui <- fluidPage(  
4   # input field  
5  textInput("user_text", label = "Enter some text:", placeholder = "Please enter some text."),  
6   actionButton("submit", label = "Submit"),  
7  
8   # display text output  
9   textOutput("text")  
10 )  
11  
12 server <- function(input, output) {  
13  
14   # observe event for updating the reactiveValues  
15   observeEvent(input$submit,  
16     {  
17       text_reactive$text <- input$user_text  
18     })  
19  
20   # reactiveValues  
21   text_reactive <- reactiveValues(  
22     text = "No text has been submitted yet."  
23   )  
24  
25   # text output  
26   output$text <- renderText({  
27     text_reactive$text  
28   })  
29 }
```

Note that reactiveValues outputs a list callable anywhere in server

LAYOUT INPUT OUTPUT REACTIVITY

Grid Layout

```
ui <- fluidPage(  
  
  title = "Diamonds Explorer",  
  
  plotOutput('plot'),  
  
  hr(),  
  
  fluidRow(  
    column(3,  
      h4("Diamonds Explorer"),  
      sliderInput('sampleSize', 'Sample Size',  
        min=1, max=nrow(dataset), value=min(1000, nrow(dataset)),  
        step=500, round=0),  
      br(),  
      checkboxInput('jitter', 'Jitter'),  
      checkboxInput('smooth', 'Smooth')  
    ),  
    column(4, offset = 1,  
      selectInput('x', 'X', names(dataset)),  
      selectInput('y', 'Y', names(dataset), names(dataset)[[2]]),  
      selectInput('color', 'Color', c('None', names(dataset))),  
    ),  
    column(4,  
      selectInput('facet_row', 'Facet Row', c(None='.', names(dataset))),  
      selectInput('facet_col', 'Facet Column', c(None='.', names(dataset)))  
    )  
  )  
)
```



Going further

Improving layout

For custom layout

<https://shiny.rstudio.com/articles/layout-guide.html>

Rely on shiny extensions

<https://rstudio.github.io/flexdashboard/articles/examples.html> [flexdashboard]

<https://rstudio.github.io/shinydashboard/examples.html> [shinydashboard]

<https://rinterface.github.io/shinydashboardPlus/index.html> [shinydashboardPlus]

Improve rendering

Customize theme

<https://shiny.rstudio.com/articles/themes.html>

Customizing UI with html and css