VISUALIZATION & SHINY

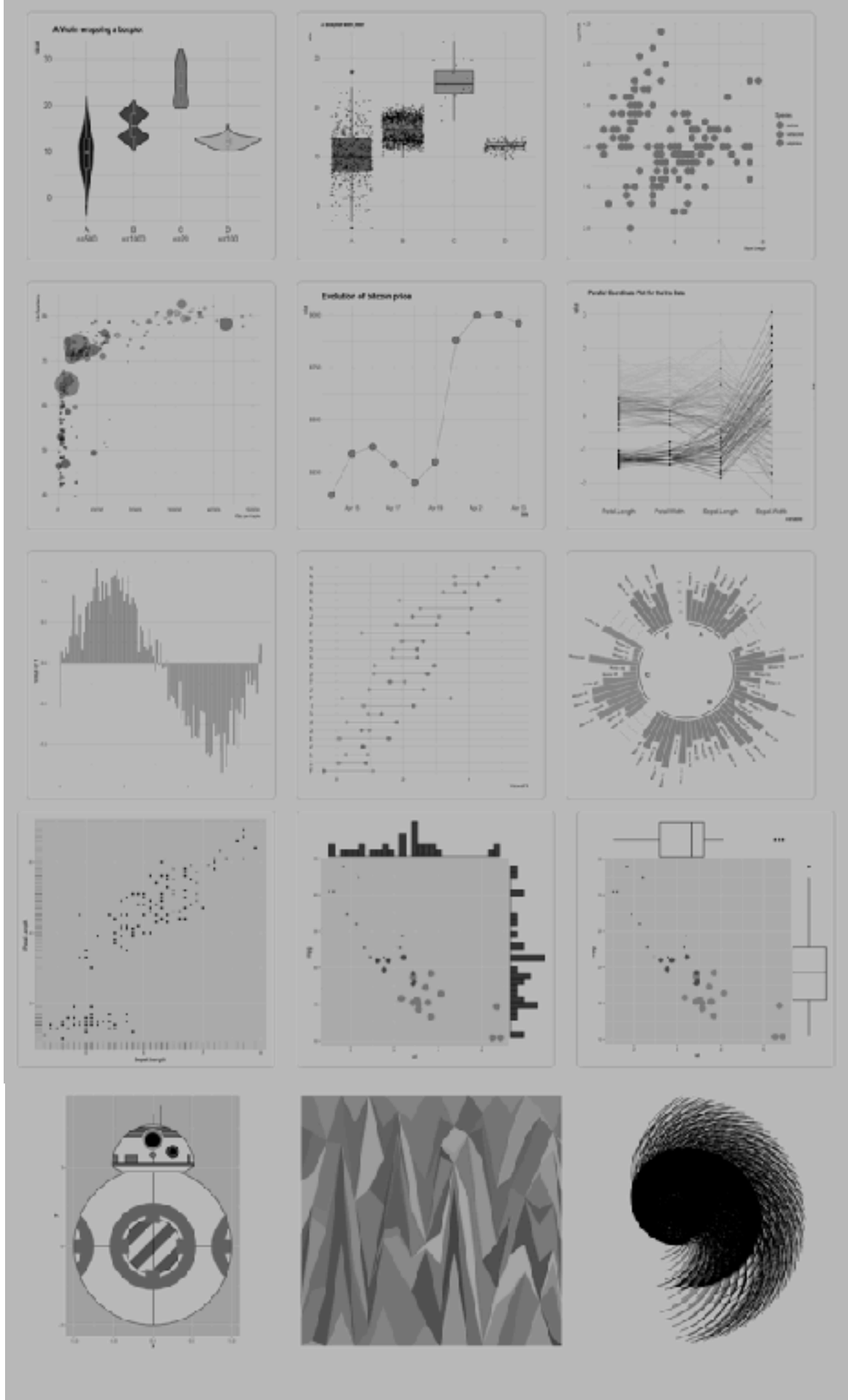
BDF 2022, CEPE GENES
Author: Frédéric Logé
Date: Nov 2021

Program

- Visualization : building graphs with ggplot2
  - Why ggplot2 ?
  - Grammar of Graphics : data, aesthetics, geometry, design, facetting
  - Practice
- Building dashboard and web apps with Shiny
  - Range of possibilities
  - Basic Shiny notions : inputs, outputs, basic reactivity, layout
  - Practice

VISUALIZATION

BUILDING GRAPHS WITH



Why ggplot2 ?

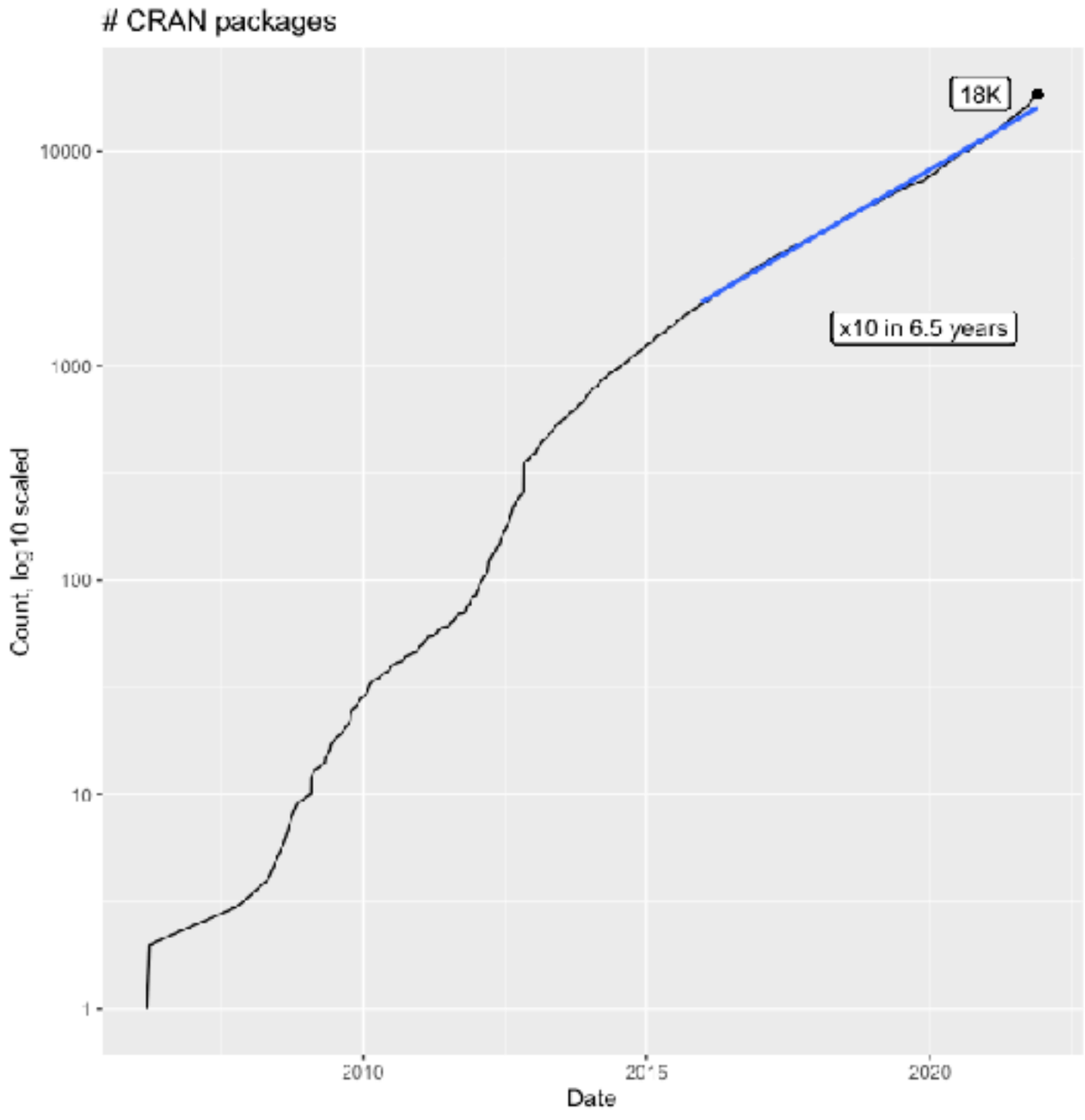
R packages - what is out there ??

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.

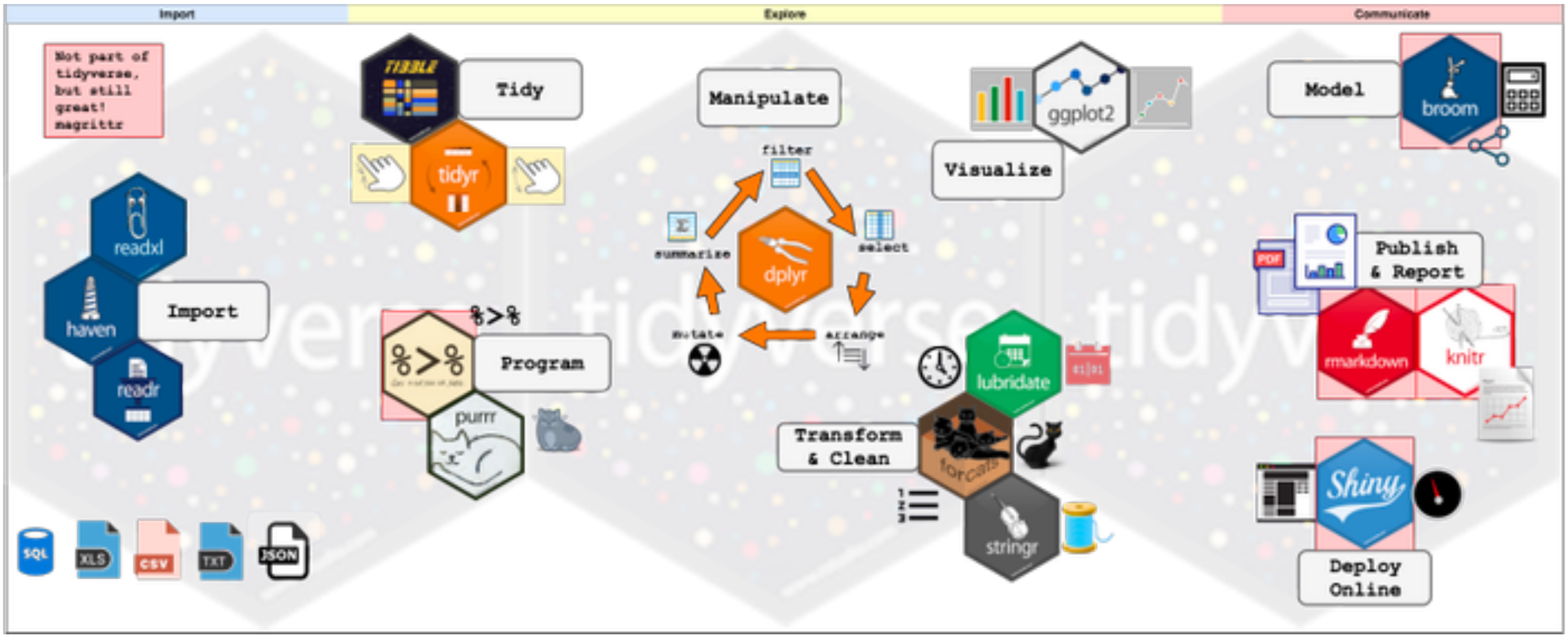
Since ~10 years, a new standard package arose for anything data related : the tidyverse package



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

Why ggplot2 ?

R packages - the tidyverse



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

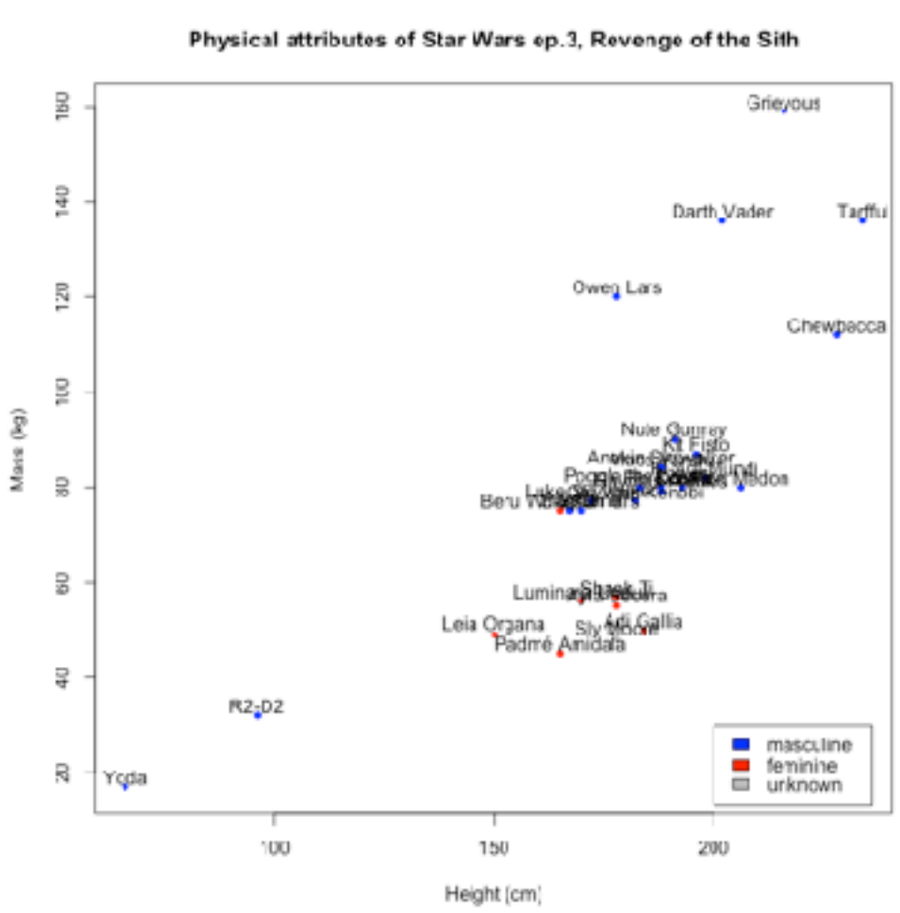
Check https://cran.r-project.org/web/views/Graphics.html

Why ggplot2 ?

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

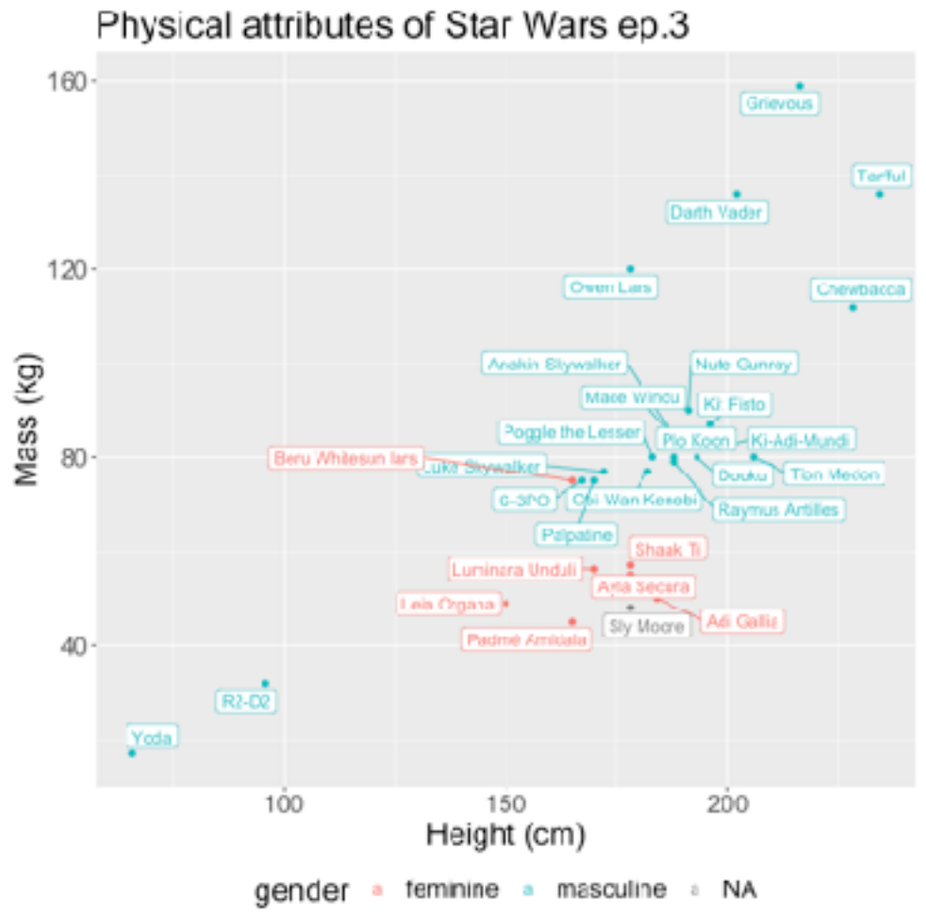
base

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



ggplot2

```
ggplot(data = starwars_ep3) +
  aes(x = height, y = mass, label = name, col = gender) +
  geom_point() +
  geom_label_repel() +
  labs(title = 'Physical attributes of Star Wars ep.3',
       x = 'Height (cm)', y = 'Mass (kg)') +
  theme(text = element_text(size = 20), legend.position = 'bottom')
```



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
...

DESIGN
labs, theme

FACETTING
split graphic by categories

```
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')
```

`+` Operator to combine ggproto objects

How ggplot2 ?

Practice & Tips

Some Ressources

stackoverflow

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

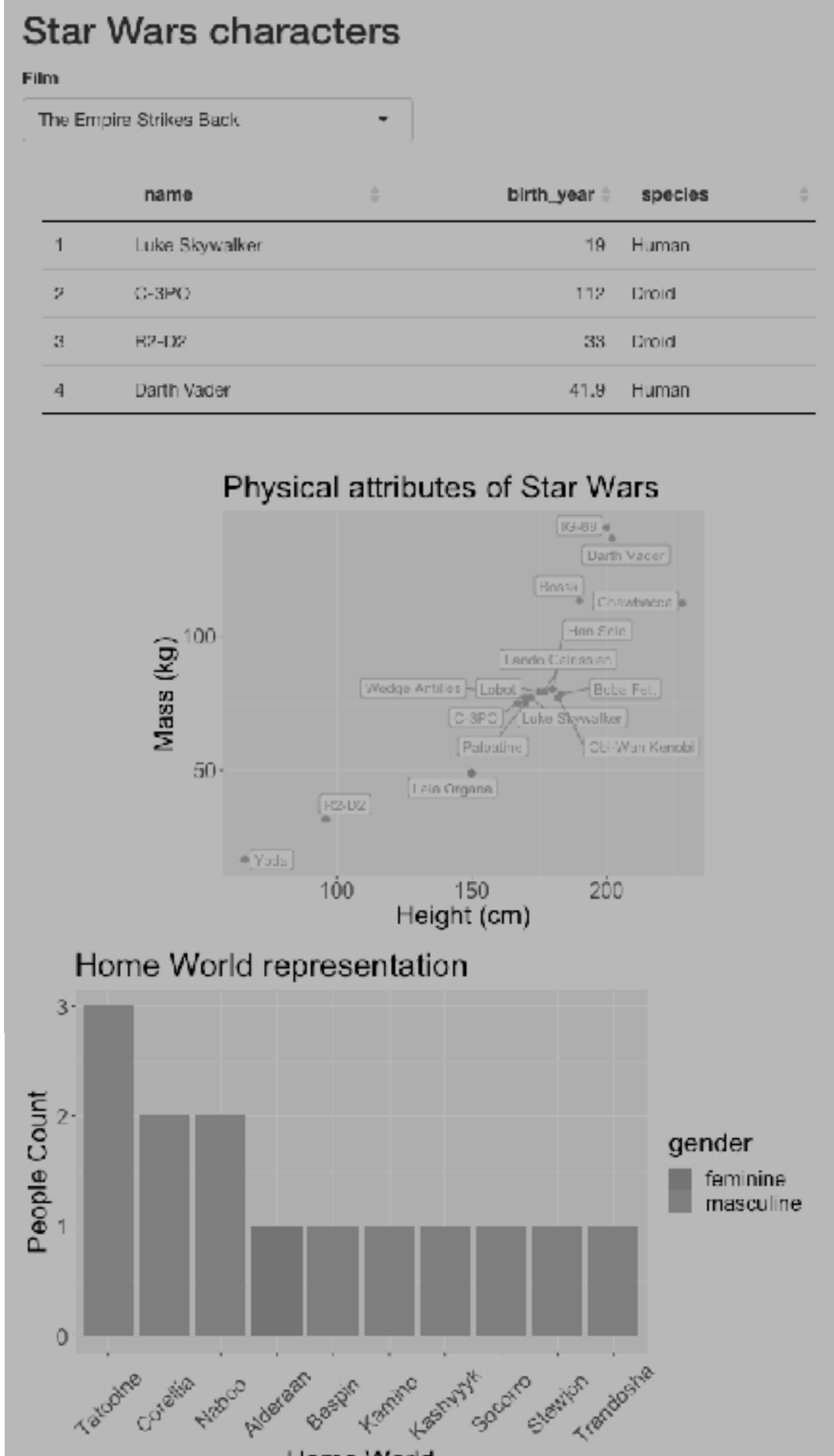
Some tips & common issues

> 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 !

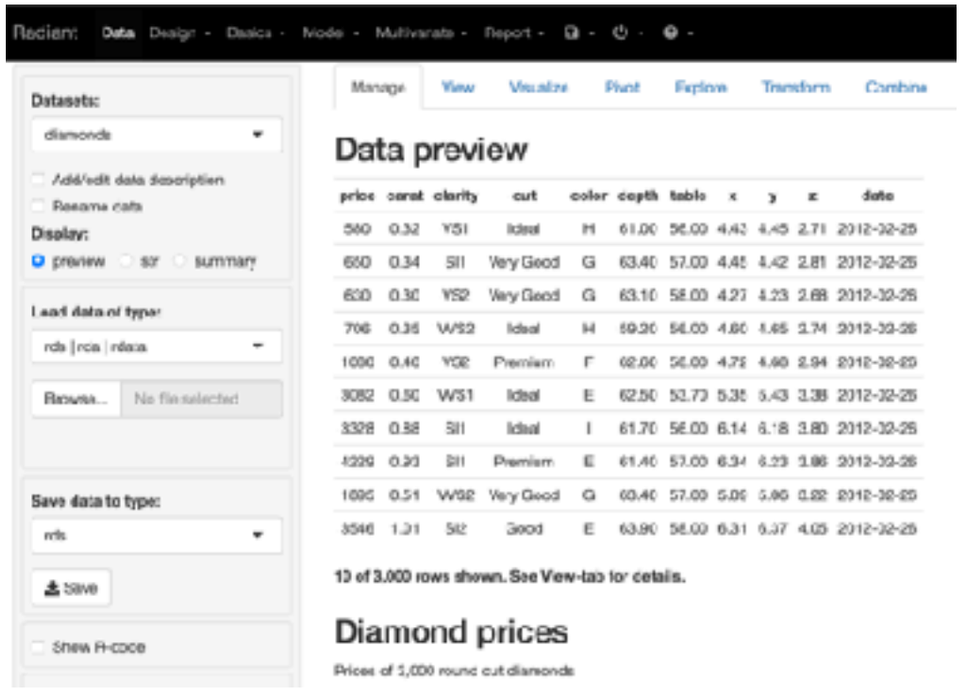
DASHBOARDS & WEBAPPS

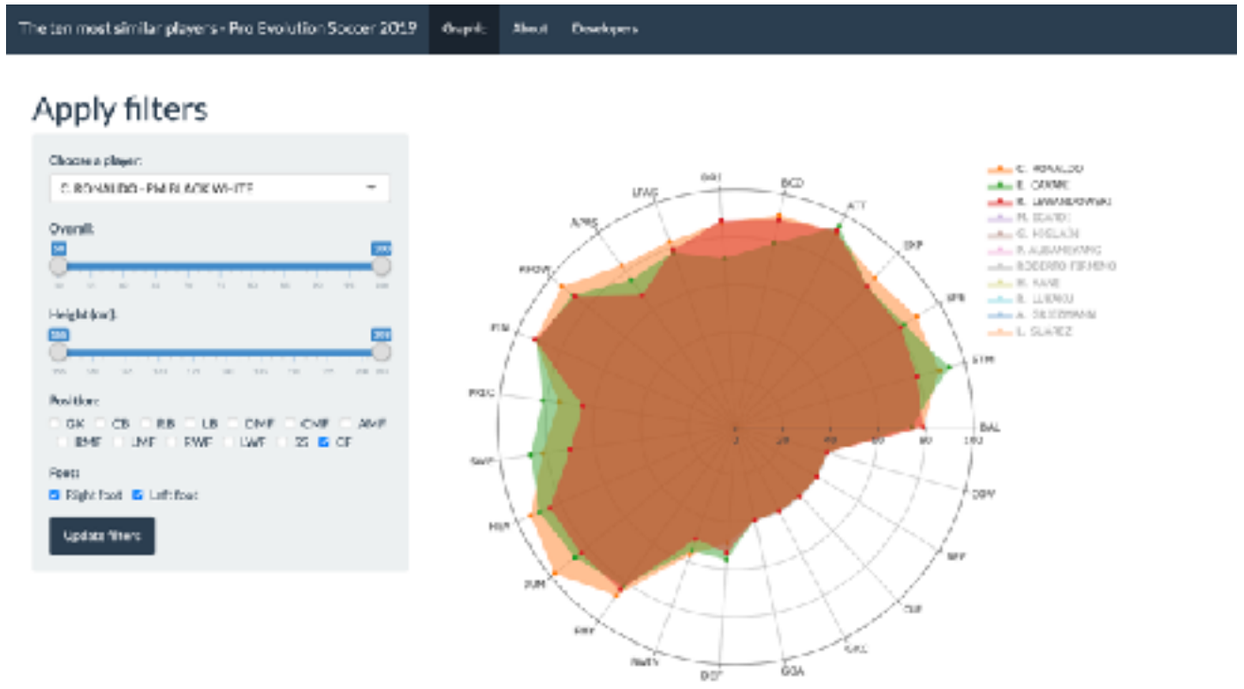
WITH

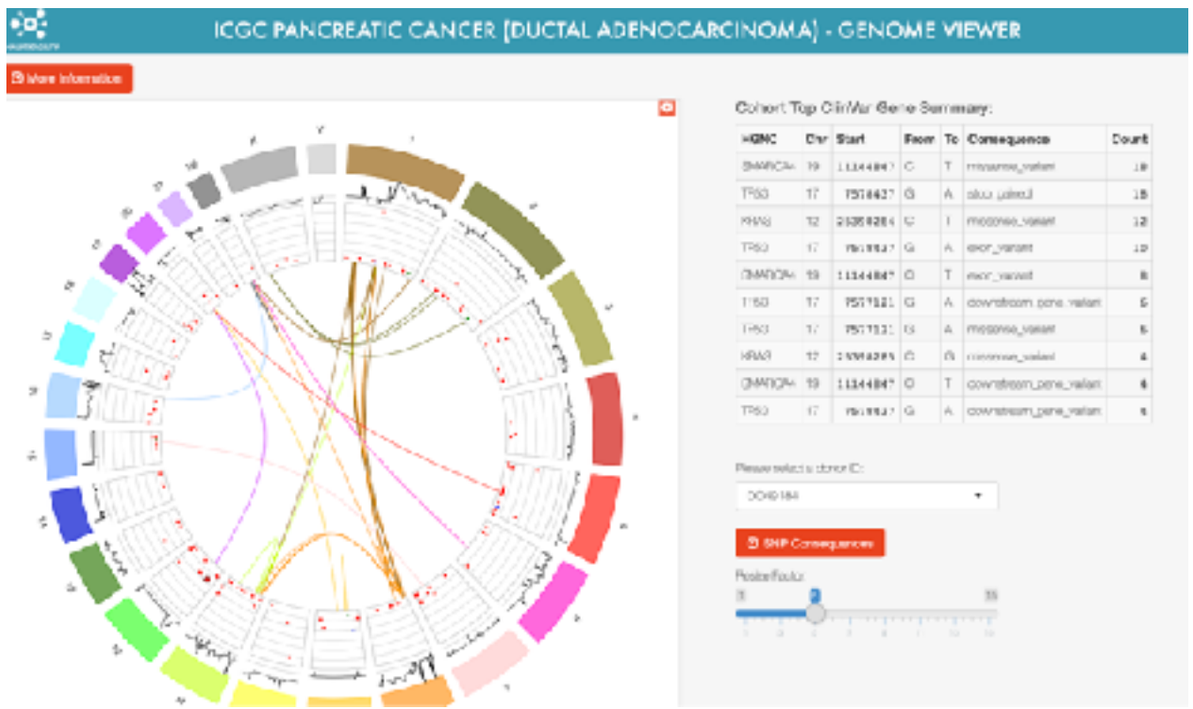


Objective

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







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 !

In ui.R

```r
# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Old Faithful Geyser Data"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),

        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("distPlot")
        )
    )
)
```

Check value of ui to see html code

In server.R

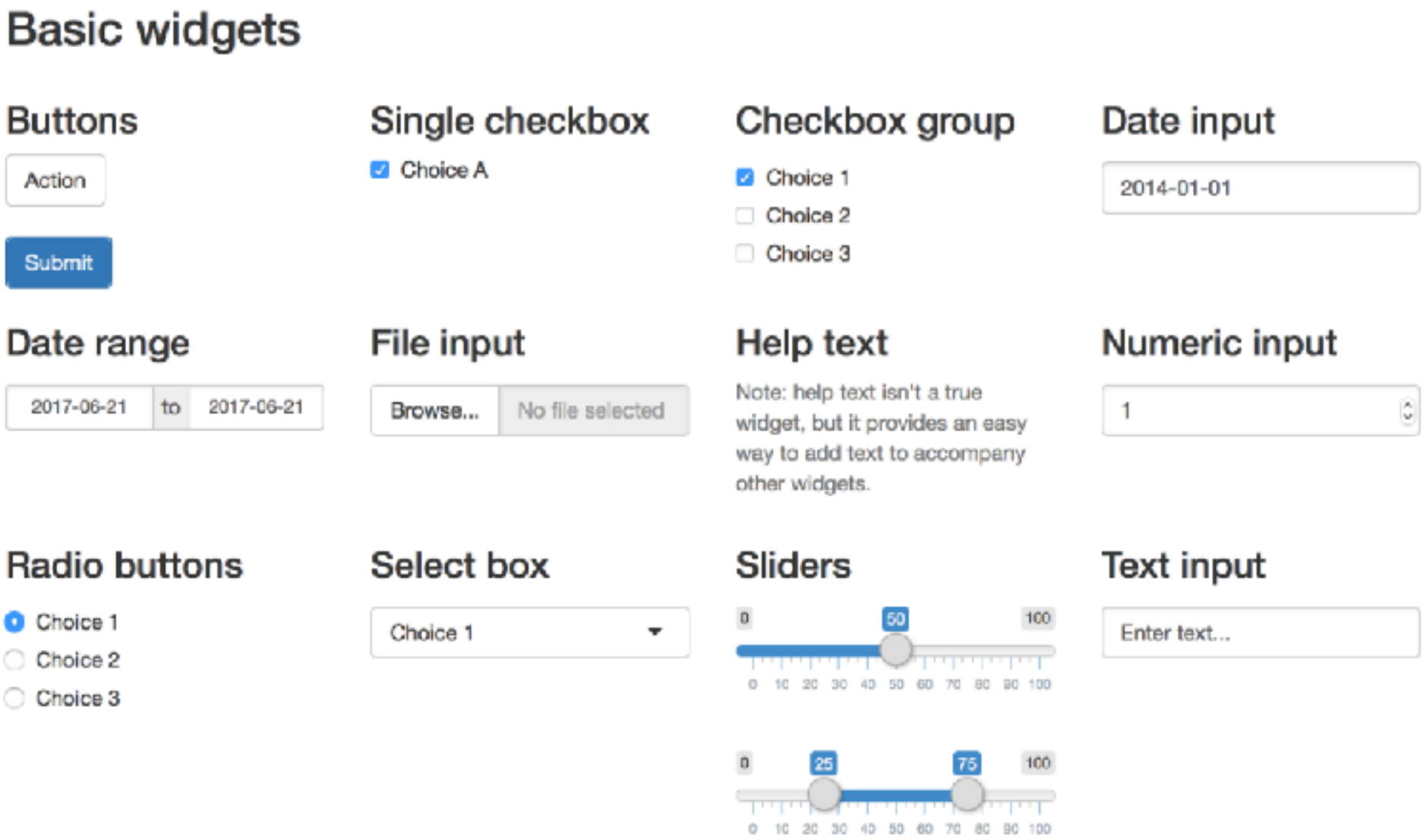
```r
# Define server logic required to draw a histogram
server <- function(input, output) {

    output$distPlot <- renderPlot({
        # generate bins based on input$bins from ui.R
        x    <- faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)

        # draw the histogram with the specified number of bins
        hist(x, breaks = bins, col = 'darkgray', border = 'white')
    })
}

# Run the application
shinyApp(ui = ui, server = server)
```

LAYOUT INPUT OUTPUT

Three basic components with which you'll be working

Input widgets

Input is called in server.R with input$some_input_id



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

Handling outputs

Creating :	In ui.R	In server.R
Raw HTML	`htmlOutput('id')` `uiOutput('id')`	`output$id = renderUI({ …})`
Text	`textOutput('id')` `verbatimTextOutput('id')`	`output$id = renderText({ …})` `output$id = renderPrint({ …})`
Table	`tableOutput('id')`	`output$id = renderTable({ …})`
DataTable	`dataTableOutput('id')`	`output$id = renderDataTable({ …})`
Image	`imageOutput('id')`	`output$id = renderImage({ …})`
Plot	`plotOutput('id')`	`output$id = renderPlot({…})`
Plotly	`plotlyOutput('id')`	`output$id = renderPlotly({ …})`

Handling reactivity

Reactivity functions	Objective	Output
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
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

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

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

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

```
ui <- fluidPage(
  textInput(inputId = "free_input", label = "Free Input"),
  selectInput(inputId = "film",label = "Film", choices = unique(unlist(starwars$films))),
  textOutput('output_text')
)

server <- function(input, output) {

  output$output_text <- renderText({
    paste0(isolate(input$film), " - ", input$free_input)
  })

}

shinyApp(ui, server)
```

LAYOUT | INPUT | OUTPUT | REACTIVITY

Reactivity

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

```r
ui <- fluidPage(
  selectInput(inputId = "film",label = "Film", choices = unique(unlist(starwars$films))),
  textOutput('genericInfo'),
  br(),
  h5('First 4 characters : '),
  dataTableOutput("myTable")
)

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

  df <- reactive({
    starwars %>%
      filter(map_lgl(films, function(x) input$film %in% x))
  })

  output$genericInfo <- renderText({
    paste0('Nb of characters : ', nrow(df()))
  })

  output$myTable <- renderDataTable({
    datatable(head(df()[,c('name', 'birth_year', 'species')], 4),
              options = list(dom = 't'))
  })


}

shinyApp(ui, server)
```

LAYOUT | INPUT | OUTPUT | REACTIVITY

Note that reactive outputs a function callable anywhere in server

Reactivity

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

```
ui <- fluidPage(
  headerPanel("Example eventReactive"),

  mainPanel(

    # input field
    textInput("user_text", label = "Enter some text:", placeholder = "Please enter some text."),

    # submit button
    actionButton("submit", label = "Submit"),

    # display text output
    textOutput("text"))
)

server <- function(input, output) {

  # reactive expression
  text_reactive <- eventReactive(input$submit, {
    input$user_text
  })

  # text output
  output$text <- renderText({
    text_reactive()
  })
}
```

Note that eventReactive outputs a function callable anywhere in server

LAYOUT | INPUT | OUTPUT | REACTIVITY

Reactivity

reactiveValues({}) : handling data

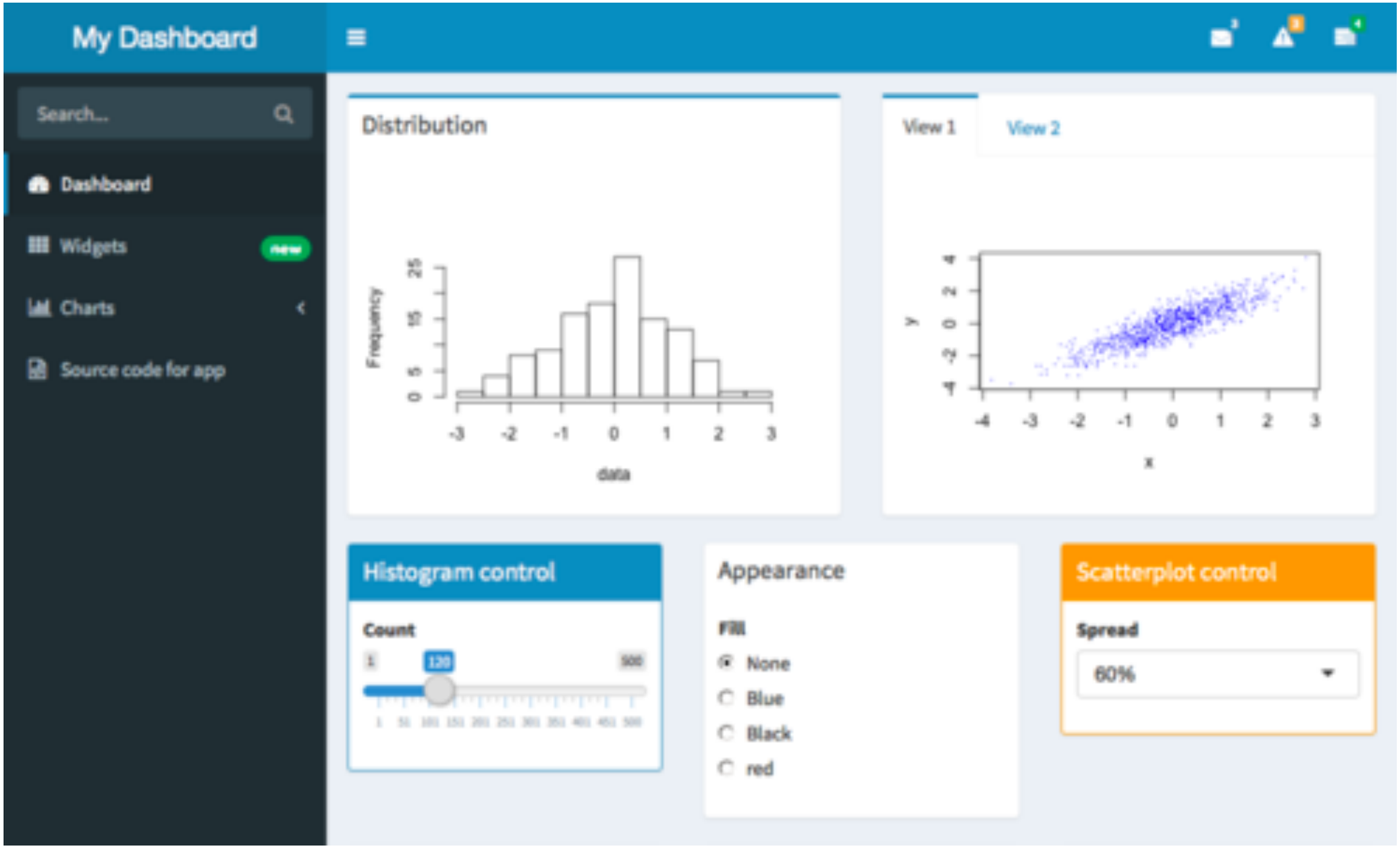
```r
ui <- fluidPage(
    # input field
    textInput("user_text", label = "Enter some text:", placeholder = "Please enter some text."),
    actionButton("submit", label = "Submit"),

    # display text output
    textOutput("text")
)

server <- function(input, output) {

  # observe event for updating the reactiveValues
  observeEvent(input$submit,
               {
                 text_reactive$text <- input$user_text
               })

  # reactiveValues
  text_reactive <- reactiveValues(
    text = "No text has been submitted yet."
  )

  # text output
  output$text <- renderText({
    text_reactive$text
  })
}
```

Note that eventReactive outputs a function callable anywhere in server

Layout

Overall setup

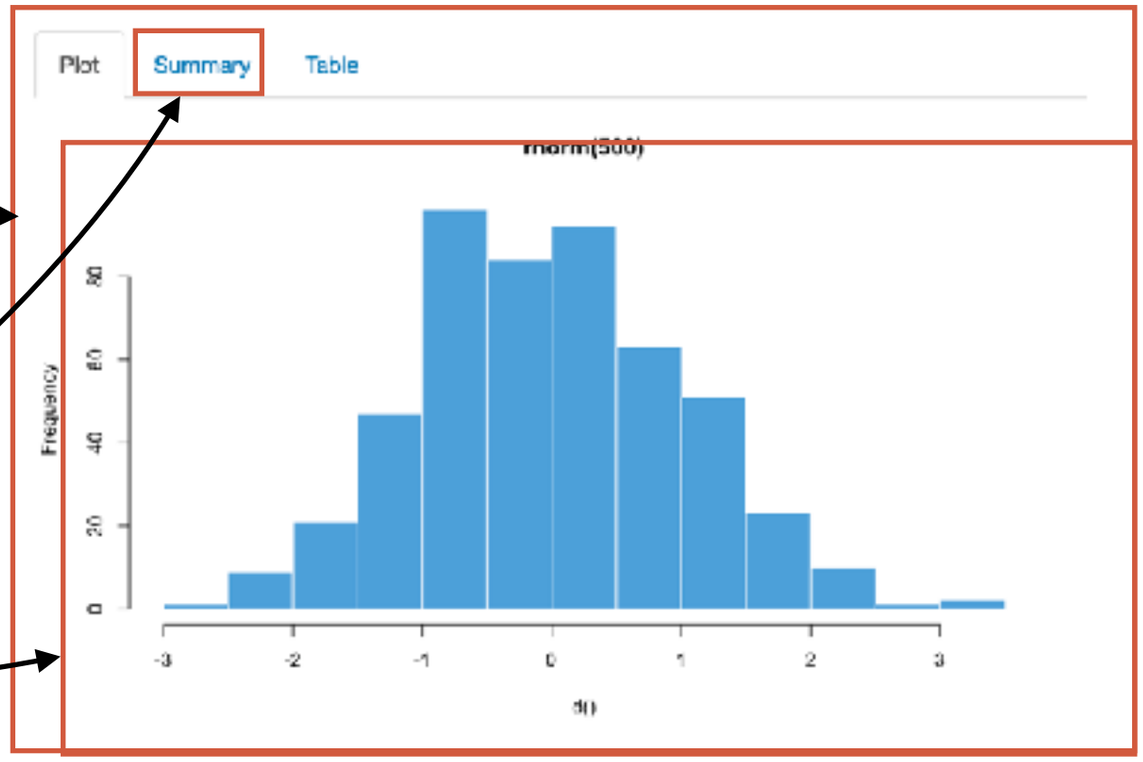
You can start off with a blank shiny app, or the classic layout offered by the default Shiny app.
Another awesome template is given by shinydashboard :



Layout

Tabsets

```
ui <- fluidPage(

  titlePanel("Tabsets"),

  sidebarLayout(

    sidebarPanel(
      # Inputs excluded for brevity
    ),

    mainPanel(
      tabsetPanel(
        tabPanel("Plot", plotOutput("plot")),
        tabPanel("Summary", verbatimTextOutput("summary")),
        tabPanel("Table", tableOutput("table"))
      )
    )
  )
)
```



Layout

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

Interactivity with plots

ggplotly

Hover and click options with renderPlot({...}), see https://shiny.rstudio.com/articles/plot-interaction-advanced.html

Customize theme with https://shiny.rstudio.com/articles/themes.html

Customizing UI with html and css

For custom layout, visit :
https://shiny.rstudio.com/articles/layout-guide.html

Refactoring

Before

```
ui = shinyUI(fluidPage(

  titlePanel("Star Wars characters"),

  selectInput(inputId = 'film',label = 'Film',
              choices = unique(unlist(starwars$films))),

  mainPanel(
    dataTableOutput('table1'),
    hr(''),
    plotOutput('physic1'),
    hr(''),
    plotOutput('physic2')
  )
)
)

server = shinyServer(function(input, output) {

  df <- reactive({ starwars %>%  filter(map_lgl(films, function(x) input$film %in% x)) })

  output$table1 <- renderDataTable({
    datatable(head(df()[,c('name', 'birth_year', 'species')], 4),
              options = list(dom = 't'))
  })

  output$physic1 <  renderPlot({
    h = df()
    print(h)
    ggplot(data = h) +
      aes(x = height, y = mass, label = name, col = gender) +
      geom_point(size = 2) +
      geom_label_repel() +
      coord_fixed() +
      labs(title = 'Physical attributes of Star Wars',
           x = 'Height (cm)', y = 'Mass (kg)') +
      theme(text = element_text(size = 20), legend.position = 'bottom')
  })

  output$physic2 <- renderPlot({
    hh = df() %>%
      filter(is.na(homeworld) == FALSE) %>%
      count(homeworld, gender) %>%
      group_by(homeworld) %>%
      mutate(nn = sum(n)) %>%
      slice_max(order_by = nn, n = 10)
    ggplot(data = hh) +
      aes(x = reorder(homeworld,  n), y = n, fill = gender) +
      geom_bar(stat = 'identity') +
      labs(x = 'Home World', y = 'People Count', title = 'Home World representation') +
      theme(text = element_text(size = 20),
            axis.text.x = element_text(vjust = 0.5, angle = 45))
  })

})

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

After

```
height_vs_mass_graph <- function(filtered_df){}

homeworld_count <- function(filtered_df){}


ui = shinyUI(fluidPage(

  titlePanel("Star Wars characters"),

  selectInput(inputId = "film",label = "Film",
              choices = unique(unlist(starwars$films))),

  mainPanel(
    dataTableOutput("table1"),
    hr(''),
    plotOutput("physic1"),
    hr(''),
    plotOutput("physic2")
  )
)
)

server = shinyServer(function(input, output) {

  df <- reactive({ starwars %>%  filter(map_lgl(films, function(x) input$film %in% x)) })

  output$table1 <- renderDataTable({
    datatable(head(df()[,c('name', 'birth_year', 'species')], 4),
              options = list(dom = 't'))
  })

  output$physic1 <  renderPlot({ height_vs_mass_graph(df()) })

  output$physic2 <- renderPlot({ homeworld_count(df()) })

})

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

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

Generic process for building the app