THE BIG-R BOOK

FROM DATA SCIENCE TO LEARNING MACHINES AND BIG DATA

— PART 07—

Dr. Philippe J.S. De Brouwer

last compiled: September 1, 2021
Version 0.1.1

THE BIG R-BOOK:
From Data Science to Big Data and Learning Machines

♡— PART 07: Report —♡

These slides are to be used in with the book – for best experience, teachers will read the book *before* using the slides and students have access to the book and the code.

part 07: Report

↓

chapter 31:

A Grammar of Graphics with ggplot2

PART 07: REPORT

↓

CHAPTER 31: A GRAMMAR OF GRAPHICS WITH GGPLOT2

↓

SECTION 1:

The Basics of ggplot2

Installing and Loading `ggplot2`

To explore `ggplot2`, we will use the dataset `mtcars` from the library `datasets`. It is usually loaded as the start of R, and it is already known from other sections in this book.

```
# install once: install.packages('ggplot2')
library(ggplot2)
```

A Basic Plot i

```
p <- ggplot(mtcars, aes(x=wt, y=mpg))
# So far printing p would result in an empty plot.
# We need to add a geom to tell ggplot how to plot.
p <- p + geom_point()

# Now, print the plot:
p
```

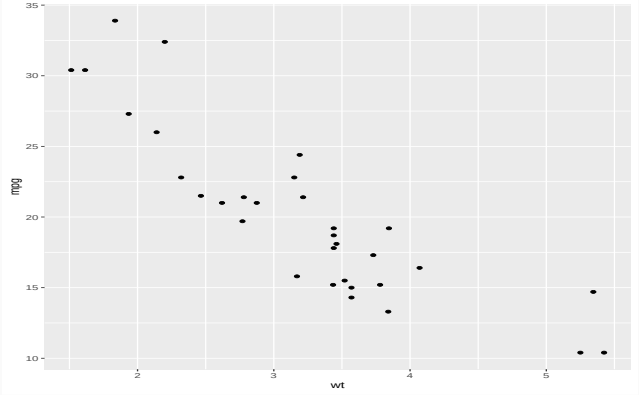

Figure 1: A basic and simple scatter-plot generated with `ggplot2`.

Adding Layers i

```
# We start from the previously generated plot p:
p <- p + aes(colour = factor(am))
p <- p + aes(size   = qsec)
p
```

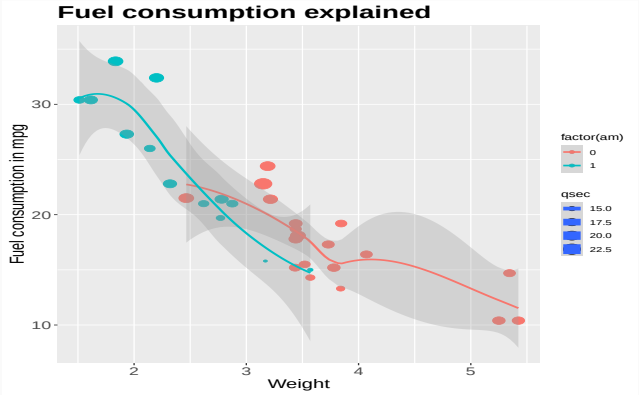

Figure 2: The same plot as in previous Figure, but now enhanced with different colours that depend on the gearbox (the parameter `am`, and the size of the dot corresponds to the time the car needs to get from standstill to the distance of 0.2 Miles (ca. 322 m).

Adding facet_wrap i

```
library(tidyverse)  # provides the pipe: %>%
mtcars %>%
  ggplot(aes(x = wt, y = mpg)) +
        geom_point() +
        geom_smooth(method = "loess", span = 0.99, alpha = 0.3) +
        xlab('Weight') +
        ggtitle('MPG in function of weight per nbr cylinders') +
        facet_wrap( ~ cyl, nrow = 2) +
        theme_classic(base_size = 14) # helps the grey to stand out
```

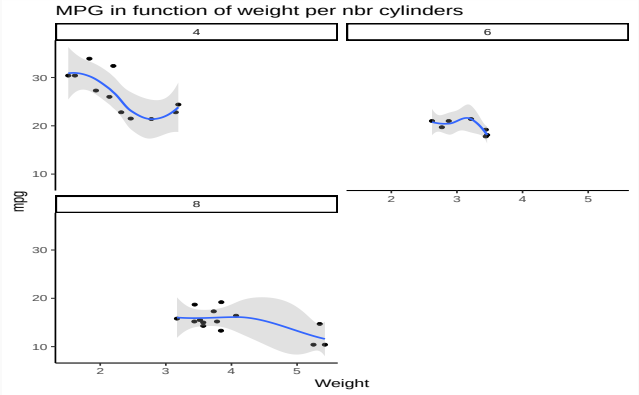

Figure 3: A facet plot will create sub-plots per discrete value of one or more variables. In this case we used the number of cylinders, so each sub-plot is for the number of cylinders that is in its title.

PART 07: REPORT

↓

CHAPTER 31: A GRAMMAR OF GRAPHICS WITH GGPLOT2

↓

SECTION 2:

Over-plotting

The Example: Building the Data i

```r
# Set the seed to allow results to be replicated:
set.seed(1868)

# Generate the data:
LTI <- runif(10000, min = 0, max = 1)
DPD <- abs(rnorm(10000,
                 mean = 70 * ifelse(LTI < 0.5, LTI, 0.5),
                 sd = 30 * sqrt(LTI) + 5))

# Plot the newly generated data and try to make it not cluttered:
plot(LTI, DPD,
     pch=19,          # small dot
     ylim =c(0, 100)) # not show outliers, hence zooming in
```

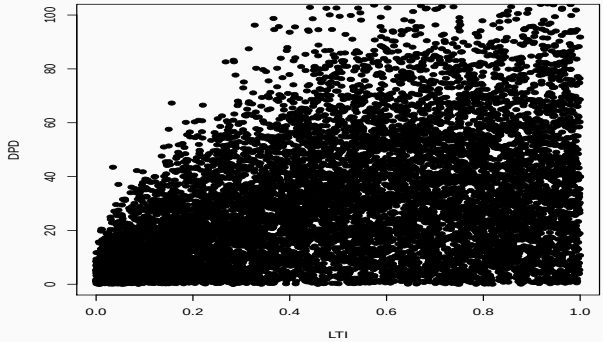

Figure 4: The standard functionality for scatterplots is not optimal for large datasets. In this case, it is not clear what the relation is between LTI and DPD. ggplot2 will provide us some more tools to handle this situation.

Tools in `ggplot2` to Deal with Overplotting

The package `ggplot2` has all the tools of the traditional `plot()` function, but it has a few more possibilities. We can choose one or more of the following:

- add additional information with `geom_smooth()`, `geom_quantile()`, or `geom_density_2d()`;
- use boxplots via `geom_boxplot()`;
- use violin plots.
- it is possible to summarise the density of points at each location and display that in some way, using `geom_count()`, `geom_hex()`, `geom_bin2d()` or `geom_density2d()`. The latter will first make a 2D kernel density estimation using `MASS::kde2d()` and display the results with contours;
- add transparency to the points by adding `geom_point(alpha = 0.05)`; and
- use small dots: `geom_point(shape = ".")`.

A fist solution that ggplot2 offers is a contour plot. Using the contour plot on the same dataset, we can work as follows and plot the results in Figure 5 on slide 17:

```
# We add also the colour schemes of viridisLite:
library(viridisLite)

d <- data.frame(LTI = LTI, DPD = DPD)
p <- ggplot(d, aes(x = LTI, y = DPD)) +
     stat_density_2d(geom = "raster", aes(fill = ..density..),
                     contour = FALSE) +
     geom_density_2d() +
     scale_fill_gradientn(colours = viridis(256, option = "D")) +
     ylim(0,100)
p
```

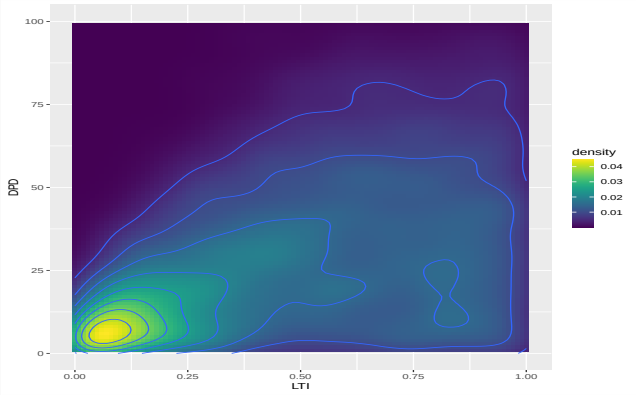

Figure 5: The contour plot is able to show where the density of points is highest by a visually attractive gradient in colour.

```
# Note that ggplot will warn us about the data that is not shown
# due to the cut-off on the y-axis -- via the function ylim(0, 100).
```

PART 07: REPORT

↓

CHAPTER 31: A GRAMMAR OF GRAPHICS WITH GGPLOT2

↓

SECTION 3:

Case Study for ggplot2

Case Study: Diamonds i

```
library(ggplot2)
library(viridisLite)

# take a subset of the dataset diamonds
set.seed(1867)
d <- diamonds[sample(nrow(diamonds), 1500),]
p <- ggplot(d, aes(x, price)) +
     stat_density_2d(geom = "raster", aes(fill = ..density..),
                     contour = FALSE) +
#     geom_density_2d() +
     facet_grid(. ~ cut) +
     scale_fill_gradientn(colours = viridis(256, option = "D")) +
     ggtitle('Diamonds per cut type')
p
```

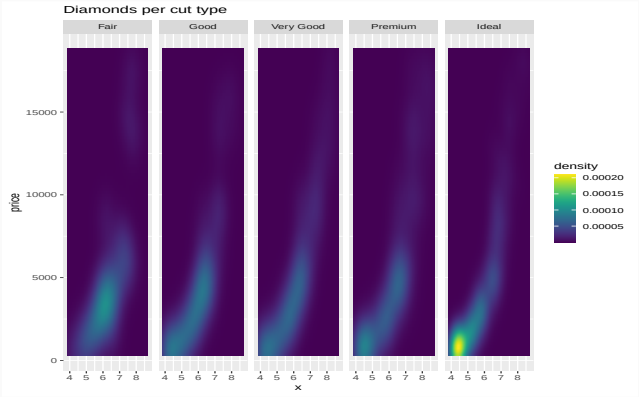

Figure 6: This plot shows a facet plot of a contour plot with customised colour scheme.

part 07: Report

↓

chapter 32:

R Markdown

R-Markdown File Structure

````
---
title: "R Markdown"
author: "Philippe De Brouwer"
date: "January 1, 2020"
output: beamer_presentation
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
```

## R Markdown
This is an R Markdown presentation. Markdown is a simple formatting
syntax for authoring HTML, PDF, and MS Word documents. For more
details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated
that includes both content as well as the output of any embedded R
code chunks within the document.

## Slide with Bullets
- Bullet 1
- Bullet 2
- Bullet 3

## Slide with R Output
```{r cars, echo = TRUE}
summary(cars)
```

## Slide with Plot
```{r pressure}
plot(pressure)
```
````

R-Markdown: Header Levels

```
# Level 1: title slide

## level 2: title or new slide
This line goes on slide the slide with title 'level 2: title
or new slide'

### level 3: box on slide
This line goes in the body of the box that will have the title
'level 3: box on slide'
```

R-Markdown: Executable R-Code

Executable code is placed between the three backtics marker (```` ``` ````), we tell that it is R-code by putting the letter “r” between curly brackets. That is also the place where we can name the code chunk and eventually override the default options `knitr::opts_chunk$set(echo = FALSE)` .

For example, the following code will create a histogram on the slide with title “50 random numbers”.

````
## 50 random numbers
```{r showPlot}
hist(runif(50))
```
````

part 07: Report

↓

chapter 33:

knitr and LaTeX

```
\documentclass[a4paper,12pt]{article}
\usepackage[utf8]{inputenc}

\title{About \LaTeX and knitr}
\author{Philippe J.S. De Brouwer}
\date{2019-02-21}

\begin{document}
<<echo=FALSE,include=FALSE>>=
library(knitr)            # load knitr
opts_chunk$set(echo=TRUE,
               warning=TRUE,
         message=TRUE,
         out.width='0.5\\textwidth',
               fig.align='center',
               fig.pos='h'
        )
@

\maketitle

\begin{abstract}
A story about FOSS that resulted in software that rivals any --even very expensive-- commercial software
     \ldots and has the best user support possible: a community of people that help each other.
\end{abstract}

\section{This is how it works}
The next code will include the R-code and its output.
<<fig.cap='a plot in LaTeX via knitr'>>=
N <- 50
hist(runif(N), col = 'deepskyblue3', border = 'white')
@
In this plot we used \Sexpr{N} random numbers and then plotted the histogram of the result.
\end{document}
```

If not already done, save this in a text-file and give it the name `latex_article.Rnw`. Now, we are ready to compile this article. Once the text-file exists, it can be compiled from the R-console as follows:

```
library(knitr);                      # load the knitr package
setpwd('/path/to/my/file')           # move to the appropriate directory
knit("latex_article.Rnw")            # creat a .tex file by executing all R-code
system('pdflatex latex_article.tex') # compile the .tex file to a .pdf
```

Since at this point you are working in a text editor and not in the R-prompt, it might make sense to compile the document from the CLI. This can be done by the following two lines in the Linux CLI:

```
R -e 'library(knitr);knit("latex_article.Rnw")'
pdflatex latex_article.tex
```

About LaTeXand knitr

Philippe J.S. De Brouwer

2019-02-21

Abstract

A story about FOSS that resulted in software that rivals any –even very expensive– commercial software . . , and has the best user support possible: a community of people that help each other.

1 This is how it works

The next code will include the R-code and its output.

```
N <- 50
hist(runif(N), col = 'deepskyblue3', border = 'white')
```

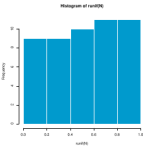

Figure 1: a plot in LaTeX via knitr

In this plot we used 50 random numbers and then plotted the histogram of the result.

Figure 7: The LaTeX article looks like this. Note that this is a cropped image so it would be readable in this book. The result will be centred on an A4 page.

part 07: Report

↓

chapter 34:

An Automated Development Cycle

- The function `source()`
- R can be invoked from the command line and from the command line we can ask R to execute some commands.
- Add those commands to the `crontab` file of your computer to automate running.
- Access those database servers from R.
- Build objects that hold everything together.
- It Write one's own packages for R.
- Produce professional documents with `knitr`, R-Markdown, or specialised dahsboards in PDF of on a dynamic webite.

part 07: Report

↓

chapter 35:

Writing and Communication Skills

Check Before we Start

1. Identify the main question or task at hand. Usually, there is a situation and a complication in some form that leads directly to one main question. Focus that single question.
2. Formulate the questions so that it addresses the concerns of the audience.
3. Break down the issue into smaller (solvable) problems.
4. Then for each of the small problems follow the logic of this book: get the relevant data, wrangle and explore, model, gather evidence, and draw conclusions.
5. Convince others by a written report and/or a convincing presentation.

Golden Rules for a Presentation

1. The ideas – at any level in the structure – are summaries of the ideas below – in other words, start with the conclusion on the title slide, the title of the next slide is the summary of the things on the slide, etc. On a smaller scale, an enumeration groups things of the same type (e.g. does not mix actions and findings) and the title of that slide will be a summary of that list.
2. Ideas in each grouping should be of the same kind – never mix things like arguments and conclusions, observations and deductions, etc.
3. Ideas in each grouping should be logically ordered – ask someone else to challenge your presentation before sending it to the boss.[1]

part 07: Report

↓

chapter 36:

Interactive Apps

part 07: Report

↓

chapter 36: Interactive Apps

↓

section 1:

Shiny

```
# install.packages('shiny') # if necessary of course
library(shiny)
runExample("01_hello")
```

Pressing enter after the line `runExample("01_hello")` will produce the output `Listening on http://127.0.0.1:7352` and open a web-page with the content in Figure 8 on slide 37. In the meanwhile, the R-terminal is not accepting any further commands. `[CTRL]+C` (or `[ESC]` in RStudio) will allow you to stop the webserver and make the command prompt responsive again.

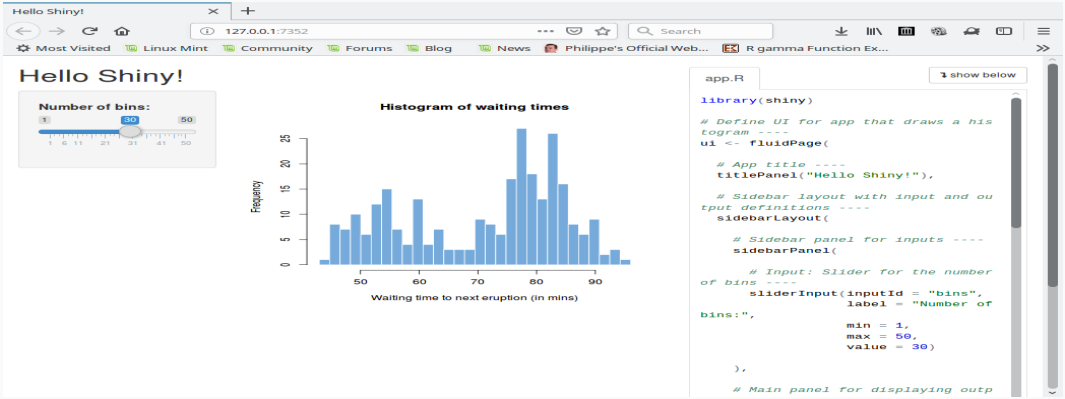

Figure 8: The output of one of the examples supplied by the Shiny package. Note the slider bar on the left and the code on the right.

An Example for shiny

```
# The filename must be app.R to execute it on a server.

# The name of the server function must match the argument in the
# shinyApp function also it must take the arguments input and output.
server <- function(input, output) {
  output$distPlot <- renderPlot({
    # any plot must be passed through renderPlot()
    hist(rnorm(input$nbr_obs), col = 'khaki3', border = 'white',
         breaks = input$breaks,
         main   = input$title,
         xlab   = "random observations")
    })
 output$nbr_txt <- renderText({
   paste("You have selected", input$nbr_obs, "observations.")
 })  # note the use of brackets ({ })
}

# The name of the ui object must match the argument in the shinyApp
# function and we must provide an object ui (that holds the html
# code for our page).
ui <- fluidPage(
  titlePanel("Our random simulator"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("nbr_obs", "Number of observations:",
                  min = 10, max = 500, value = 100),
      sliderInput("breaks", "Number of bins:",
                  min = 4, max = 50, value = 10),
      textInput("title", "Title of the plot",
                value = "title goes here")
    ),
    mainPanel(plotOutput("distPlot"),
              h4("Conclusion"),
                  p("Small sample sizes combined with a high number of bins
                 might provide a visual image of the distribution that does
                 not resemble the underlying dynamics."),
              "Note that we can provide text, but not  <b>html code</b> directly.",
              textOutput("nbr_txt")  # object name in quotes
               )
  )
)

# finally we call the shinyApp function
shinyApp(ui = ui, server = server)
```

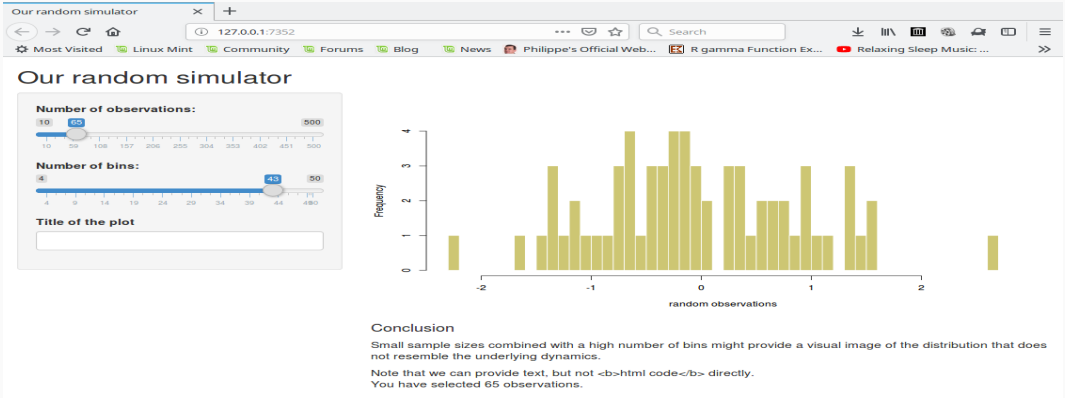

Figure 9: The web-page produced by the code fragment above.

Deploy the App on ShinyApps.io i

```r
# Load the library:
library(rsconnect)

# Upload the app (default filename is app.R)
rsconnect::deployApp('path/to/your/app',
                     server="shinyapps.io", account="xxx")
```

Then it becomes available via an url like `https://xxx.shinyapps.io/normsimulator`, where the letters “xxx” are your account name on the platform. Now, any html-page can be configured to display the app via an iframe.

```html
<iframe src="https://xxx.shinyapps.io/normsimulator/"
      frameborder="0"
      allowfullscreen
      style="width:100%;height:800px;"
      >
      </iframe>
```

Listing 1: *The html code to include our Shiny app on a live web-page.*

part 07: Report

↓

chapter 36: Interactive Apps

↓

section 2:

Browser born data visualization

- `ggvis`: Interactive plots;
- `dygraphs`: Interactive time series visualization – see `http://dygraphs.com`;
- `leaflet`: Interactive maps from all over the world – see `https://rstudio.github.io/leaflet`;
- `DiagrammeR`: A variety of diagrams (org-chars, flowcharts, etc.) – see `http://rich-iannone.github.io/DiagrammeR`;
- `3dheadmap`: Heat-maps – see `https://github.com/rstudio/d3heatmap`
- `DT`: interactive tables – see `https://rstudio.github.io/DT`;
- `network3D`: Network graphs – see `https://christophergandrud.github.io/networkD3`;
- `threeJS`: 3D scatterplots and globes – see `https://bwlewis.github.io/rthreejs`

Further information – Stunning visualisations

Need even more eye-candy? An ever growing list of widgets can be found online at `http://www.htmlwidgets.org`.

```
library(leaflet)
content <- paste(sep = "<br/>",
  "<b><a href='http://www.de-brouwer.com/honcon/'>Honorary Consulate of Belgium</a></b>",
  "ul. Marii Grzegorzewskiej 33A",
  "30-394 Krakow"
  )
map <- leaflet()                                    %>%
  addProviderTiles(providers$OpenStreetMap)         %>%
  addMarkers(lng = 19.870188, lat = 50.009159)      %>%
  addPopups(lng = 19.870188, lat = 50.009159, content,
    options = popupOptions(closeButton = TRUE))     %>%
  setView(lat = 50.009159,lng = 19.870188, zoom = 12)
map
```

When the aforementioned code is run, it will display an interactive map in a browser that looks like the figure in Figure 10 on slide 44.

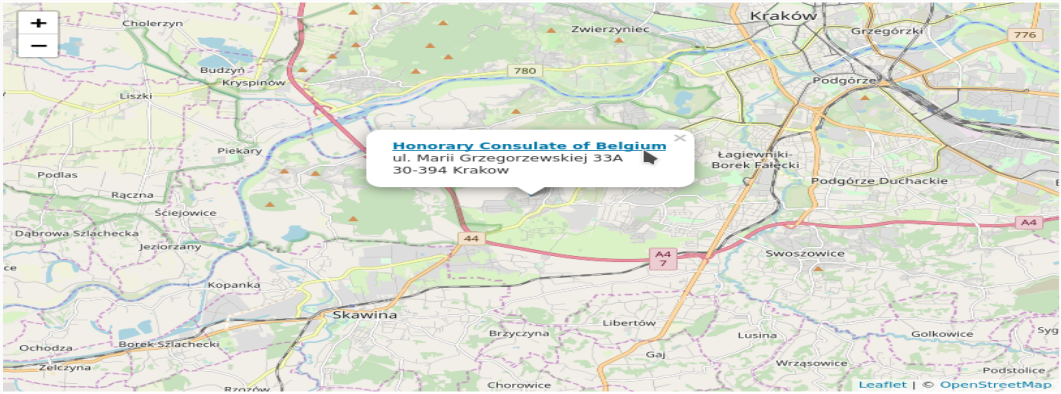

Figure 10: A map created by `leaflet` based on the famous `OpenStreetMap` maps. The ability to zoom the map is standard, the marker and popup (with clicking-enabled link) are added in the code.

Below a simple example.

```
library(titanic)    # for the data
library(tidyverse)  # for the tibble
library(ggvis)      # for the plot

titanic_train$Age     %>%
    as_tibble         %>%
    na.omit           %>%
    ggvis(x = ~value) %>%
    layer_densities(
      adjust = input_slider(.1, 2, value = 1, step = .1,
                            label = "Bandwidth"),
      kernel = input_select(
        c("Gaussian"     = "gaussian",
          "Epanechnikov" = "epanechnikov",
          "Rectangular"  = "rectangular",
          "Triangular"   = "triangular",
          "Biweight"     = "biweight",
          "Cosine"       = "cosine",
          "Optcosine"    = "optcosine"),
        label = "Kernel")
    )
```

Executing this code fragment will open a browser with the content shown in Figure 11 on slide 46.

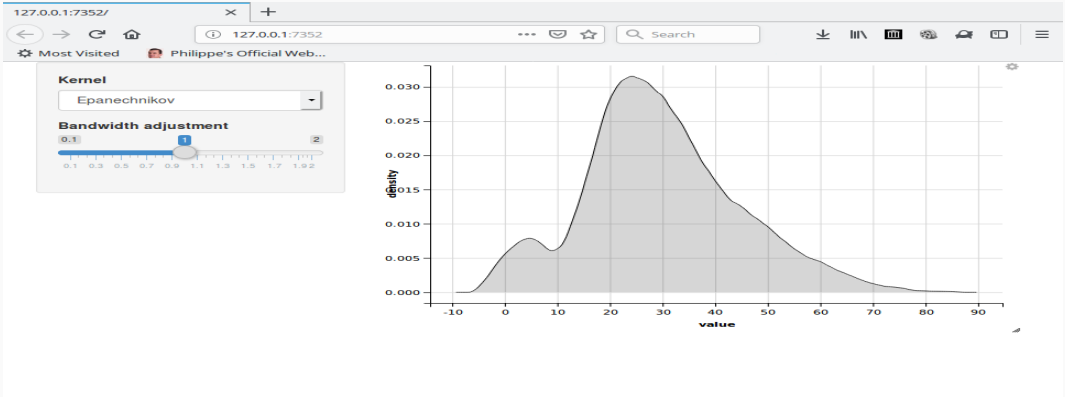

Figure 11: A useful tool to explore new data and/or get an intuitive understanding of what the different kernels and bandwidth actually do for a kernel density estimation .

Using both ggvis and shiny i

```
library(ggvis)

function(input, output, session) {
  # A reactive subset of mtcars:
  reacCars <- reactive({ mtcars[1:input$n, ] })

  # Register observers and place the controls:
  reacCars %>%
     ggvis(~wt, ~mpg, fill=(~cyl)) %>%
     layer_points() %>%
     layer_smooths(span = input_slider(0.5, 1, value = 1,
        label = 'smoothing span:')) %>%
     bind_shiny("plot1", "plot_ui_div")

  output$carsData <- renderTable({ reacCars()[, c("wt", "mpg")] })
}
```

Next, we need the ui.R-file that provides the code regulating how the items are placed on the screen:

```
library(ggvis)

fluidPage(sidebarLayout(
  sidebarPanel(
    # Explicit code for a slider-bar:
    sliderInput("n", "Number of points", min = 1, max = nrow(mtcars),
                value = 10, step = 1),
    # No code needed for the smoothing span, ggvis does this:
    uiOutput("plot_ui_div") # produces a <div> with id corresponding
                            # to argument in bind_shiny
  ),
  mainPanel(
    # Place the plot "plot1" here:
    ggvisOutput("plot1"),    # matches argument to bind_shiny()
    # under this the table of selected card models:
    tableOutput("carsData")  # parses the result of renderTable()
  )
))
```

The application can now be placed on a server – such as ShinyApps.io for example – or opened in a web-browser directly via the following command.

```
shiny::runApp("/path/to/my/app")
```

Executing this line will open a browser with the content shown in Figure 12 on slide 49. This allows us to inspect how the app works before publishing it.

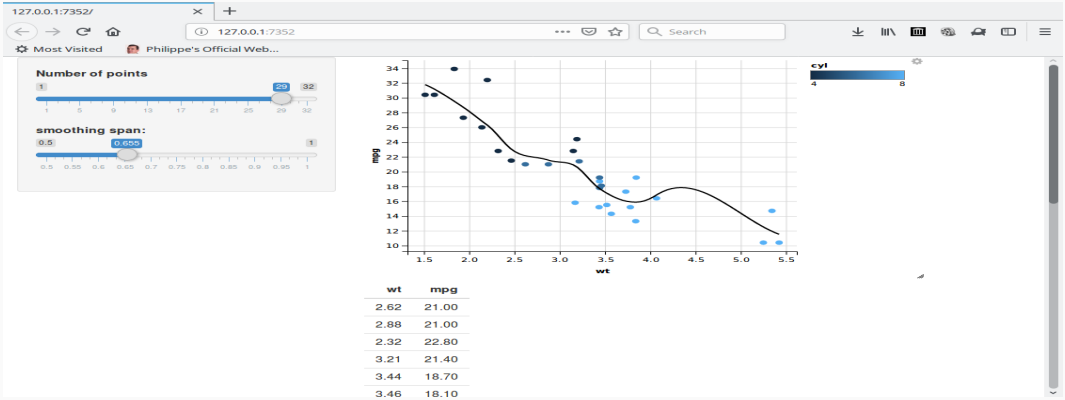

Figure 12: An interactive app with `ggvis`. This particular example uses the data from the dataset mtcars, to experiment with two parameters: the size of the sample and the span of the Loess smoothing. This is most useful to get a better intuitive understanding of the impact of sample size and span.

part 07: Report

↓

chapter 36: Interactive Apps

↓

section 3:

Dashboards

For Dashboards We Have Two Great Choices

Aspect	Flexdashboard	Shinydashboard
Learning curve	Easy	Harder
Coding	R Markdown	Shiny UI code
Objective	Static or dynamic	Dynamic
Layout via	`flexbox` CSS	Bootstrap grid
Homepage	`https://rmarkdown.rstudio.com/flexdashboard`	`http://rstudio.github.io/shinydashboard`

Table 1: A comparision of `flexdashboard` and `shinydashboard`.

```
# If not done yet, install the package:
# install.packages('flexdashboard')

# Then load the package:
library(flexdashboard)
```

Then in RStudio, chose “new file ” in the file menu and select then “R Markdown,” then “from template” and finally select “Flex Dashboard” from the list. The screen will look like Figure **??** on slide **??**, and the document will now look similar the following code. Although, note that this is only the framework (sections without the actual code).

````
---
title: "Untitled"
output:
  flexdashboard::flex_dashboard:
    orientation: columns
    vertical_layout: fill
---
```{r setup, include=FALSE}
library(flexdashboard)
```

Column {data-width=650}
-----------------------------------------------------
### Chart A
````

````
Column {data-width=350}
-----------------------------------------------------
### Chart B
```{r}

```

### Chart C
```{r}

```
````

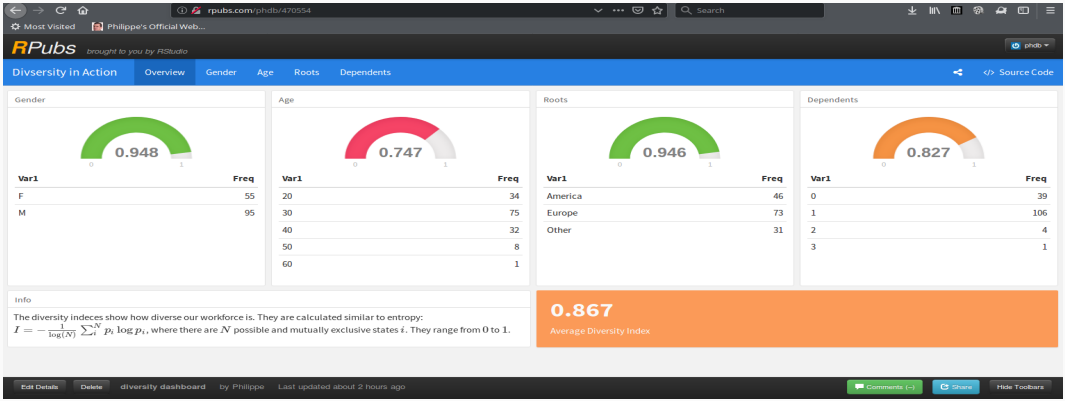

Figure 13: The welcome page of the dashboard provides the overview (menu) at the top and in the body an overview of some diversity indices.

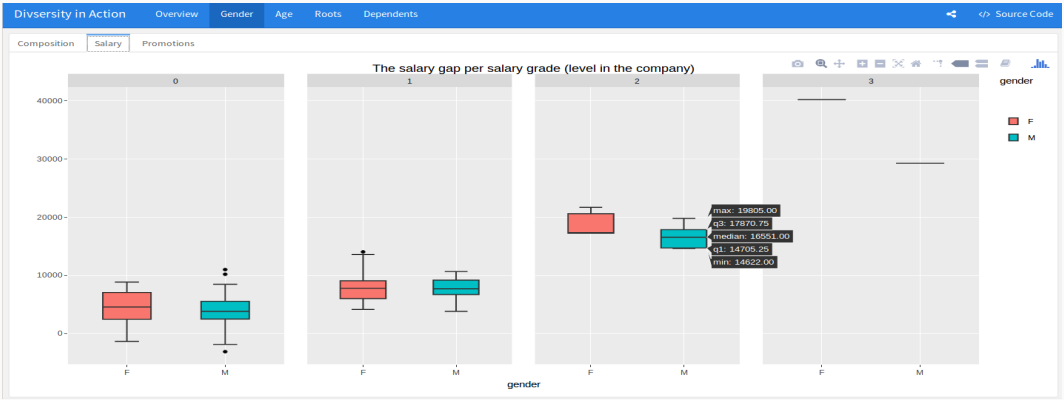

Figure 14: The page “Gender” of the dashboard provides multiple views of the data that are each made visible by clicking on the relevant tab. The effect of `ggplotly()` is visible by the toolbar that appears just right above the plot and the mouse-over effect on the boxplot for the salary grade 2/male population.

Example of a Dashboard with `ShinyDashBoard` i

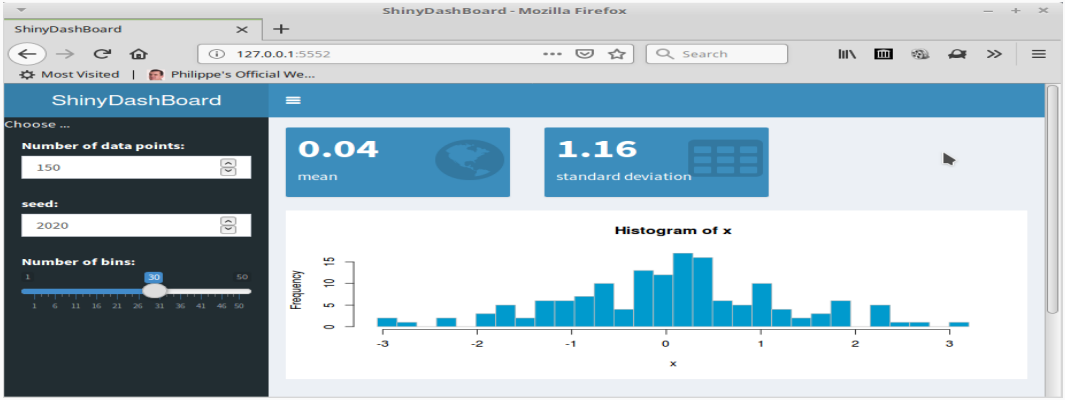

Figure 15: A simple dashboard with `shinydashboard`.

The code to obtain this dashboard is as follows:

Example of a Dashboard with ShinyDashBoard ii

```
# this file should be called app.R
library(shinydashboard)

# general code (not part of server or ui function)
my_seed <- 1865
set.seed(my_seed)
N <- 150

# user interface ui
ui <- dashboardPage(
  dashboardHeader(title = "ShinyDashBoard"),
  dashboardSidebar(
      title = "Choose ...",
      numericInput('N', 'Number of data points:', N),
      numericInput('my_seed', 'seed:', my_seed),
      sliderInput("bins", "Number of bins:",
                  min = 1, max = 50, value = 30)
  ),
  dashboardBody(
    fluidRow(
      valueBoxOutput("box1"),
      valueBoxOutput("box2")
      ),
    plotOutput("p1", height = 250)
    )
  )
```

Example of a Dashboard with ShinyDashBoard iii

```
# server function
server <- function(input, output) {
  d <- reactive({
    set.seed(input$my_seed)
    rnorm(input$N)
    })
  output$p1 <- renderPlot({
    x    <- d()
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = 'deepskyblue3', border = 'gray')
    })
  output$box1 <- renderValueBox({
    valueBox(
      value    = formatC(mean(d()), digits = 2, format = "f"),
      subtitle = "mean", icon     = icon("globe"),
      color    = "light-blue")
    })
  output$box2 <- renderValueBox({
      valueBox(
        value    = formatC(sd(d()), digits = 2, format = "f"),
        subtitle = "standard deviation",  icon     = icon("table"),
        color    = "light-blue")
      })

}

# load the app
shinyApp(ui, server)
```