# Getting Started with Interactive Visualisation

—

Dr. Bruno Voisin - Irish Centre for High-End Computing

CSO Visualisation Seminar, Dublin Castle, 12/09/19

# Outline

- Data Visualisation and Data Interaction
- What, Where and Who?
- Thinking small: it's underrated!
- The elephant in the room: D3.js
- Jupyter notebooks: sharing the recipe
- R and the Shiny package: easy web applications
- Parting thoughts

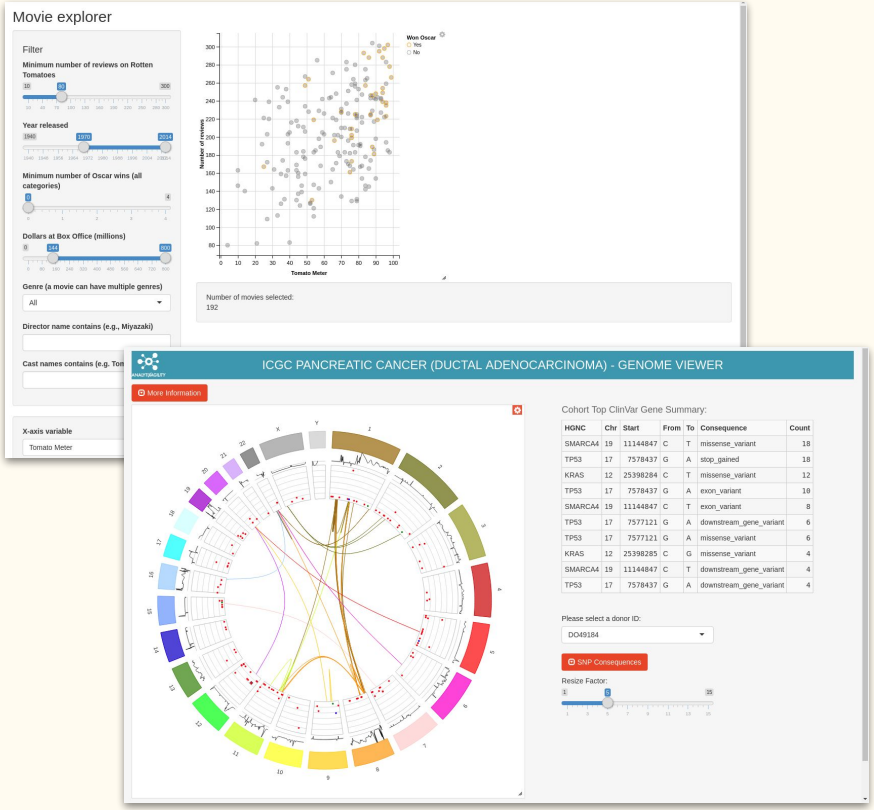
# Data Visualisation and Data Interaction

*You don't always know what you want people to see.*

- Visualisation is often about providing the reader with an intuitive understanding of specifically selected insights:
  - Scatter plot and fitted linear model.
  - Histograms showing data distribution between different categories.
  - Maps with colour scales chosen to visually separate specific ranges of numbers.
- Interaction is about providing users with the means to explore the data by themselves:
  - Axis tuning, different models of calculating the fit.
  - Modular bin width, multiple options for grouping/categories.
  - Choice of data to colour the map, colour scale tuning, map granularity.

# What, Where and Who?

- What kind of use cases?
  - Small interactive plots.
  - Larger dashboards.
  - Interactive digital reports.
- Where can they be used/published?
  - The user's desktop.
  - The Web.
- Who is the audience?
  - The data scientist.
  - The domain expert.
  - The greater public.



# Thinking small: it's underrated!

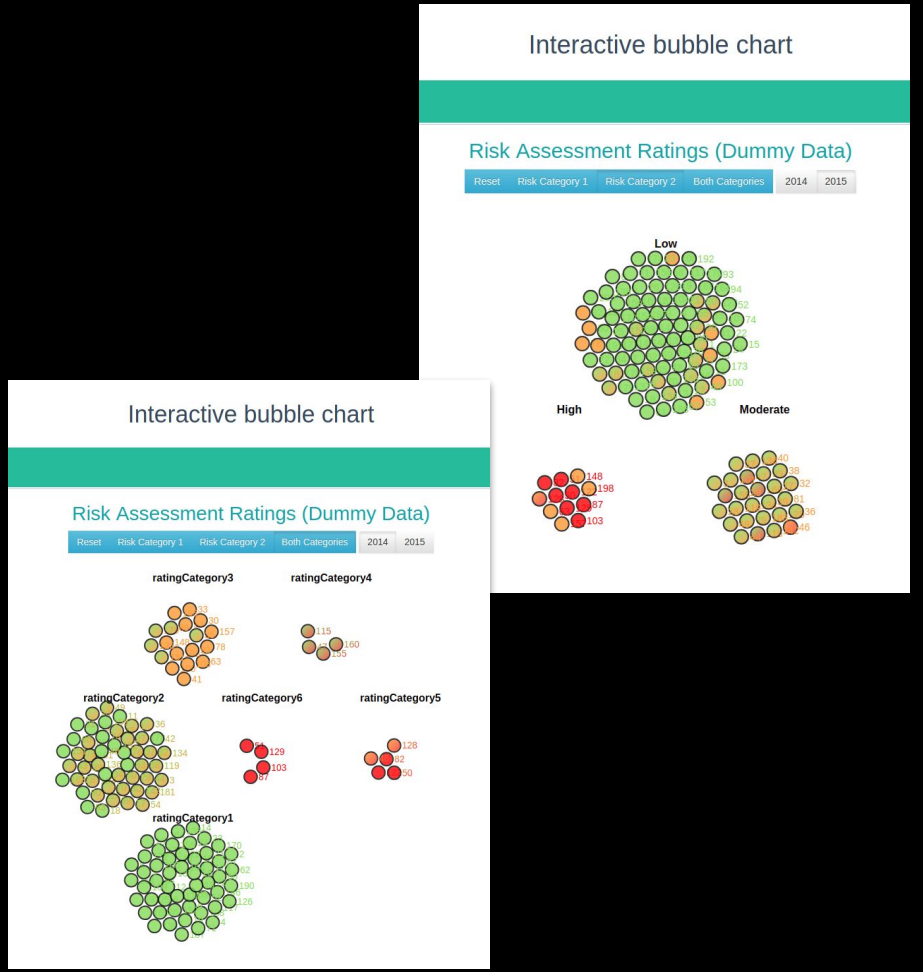
- Exploring data through interactive visualisation is handy... as long as it doesn't take too much effort to get it to work.
- While the Web is full of impressive dashboards that took a lot of software engineering efforts to design, there's a lot that can be done quickly for smaller scale use.
- Crucially, a lot of tools are now in the hands of the very people who process the data in the first place: Python and R for example have convenient packages for plot interactivity.
- Let's forget the big Web publishing, and with minimal effort we have access to a way to share data "hands-on" with our colleagues.

# The elephant in the room: D3.js

- JavaScript library for data transformation and visualisation.
- Pros:
  - A lot of very pretty visualisation options including interactive ones.
  - It's JavaScript.
- Cons:
  - A lot of low level tracing rather than plotting.
  - It's JavaScript.
- You probably won't want to use it directly but D3 these days is a fantastic building block on top of which easier solutions are built.

# Sample D3 visualisation

- Source data and visualisation process both contained on the page.
- D3 used to transform the source data then to generate self-clustering bubbles.
- A number of boxes allow for filtering and grouping options.



# D3: live examples

- The D3 Graph gallery:

  https://www.d3-graph-gallery.com/

- A user repository with a set of examples, including the Interactive Bubble Chart shown previously:

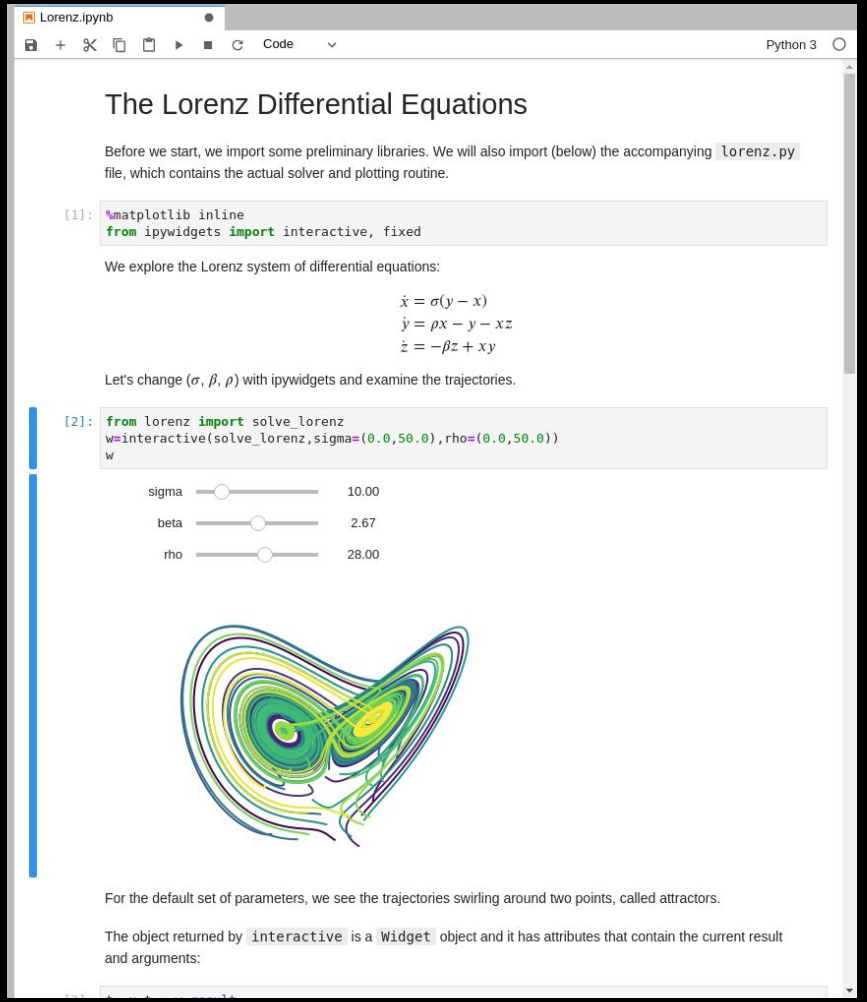
  https://bl.ocks.org/larsenmtl

# Jupyter notebooks: sharing the recipe

- Open source web application to share rich documents that include live programming code.
- You can not only provide plots, but also the code you used to plot them, allowing users to modify it and play with the visualisation details.
- Direct interaction widgets (ex: sliders) are also available for predefined interactivity.
- Supports multiple languages commonly used for data processing/visualisation (Python, R, Julia...)
- Can be installed locally on your own computer, or run remotely from a cloud server.

# Sample Jupyter Notebook

- Looks like any other online document.
- Split into a number of ‘cells’ which can be of *markdown*, *code* or *raw* type.
- All cells can be edited by the user.
- *Markdown* cells get interpreted, allowing for various formatted content such as bold/italic fonts or mathematical equations.
- *Code* cells can be executed, eventually producing additional content on the document.

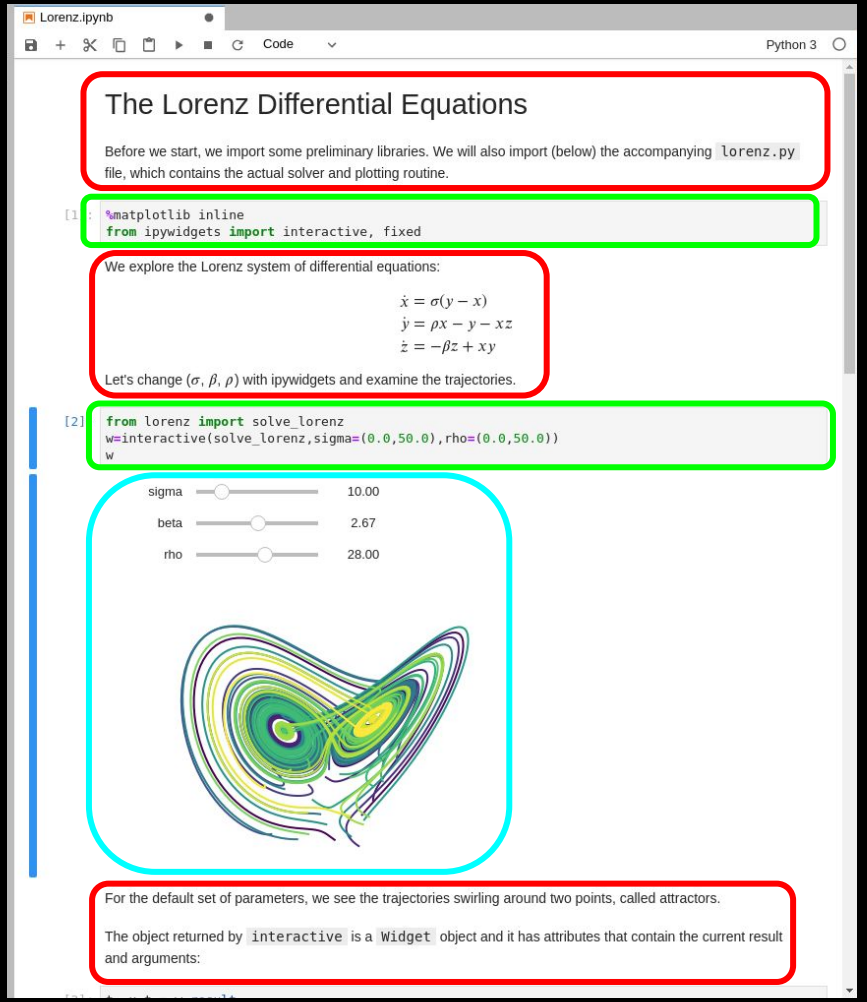


# Sample Jupyter Notebook

Text content (markdown formatting)

Code content (Python 3 here)

Results from code execution



# Jupyter Notebook: live example

- The Jupyter project web page with cloud-based examples, including the Lorenz equation notebook shown previously:

  https://jupyter.org

# R and the Shiny package: easy web applications

- A step above Jupyter, Shiny allows the creation of interactive web applications that use R for data processing and plotting.
- Shiny is an R package, and integrates well with RStudio (same authors).
- Provides a reasonably easy way to define an application layout with menus, sidebars, tabs...
- Provides all expected widgets for interactivity: text boxes, dropdowns, buttons, sliders...
- Processing/plotting can depend on such widgets and be automatically refreshed if anything it depends on changes.

# Sample Shiny App

- A Shiny application is generally split into two files (or more for complex server-logic):
  - app.R contains the graphical interface layout.
  - server.R contains the data processing and plotting logic.
- Both files refer to each other:
  - app.R defines blocks for expected server.R output.
  - server.R uses input from app.R widgets to process data as the user requires.
- The resulting application can run in a web browser.

app.R

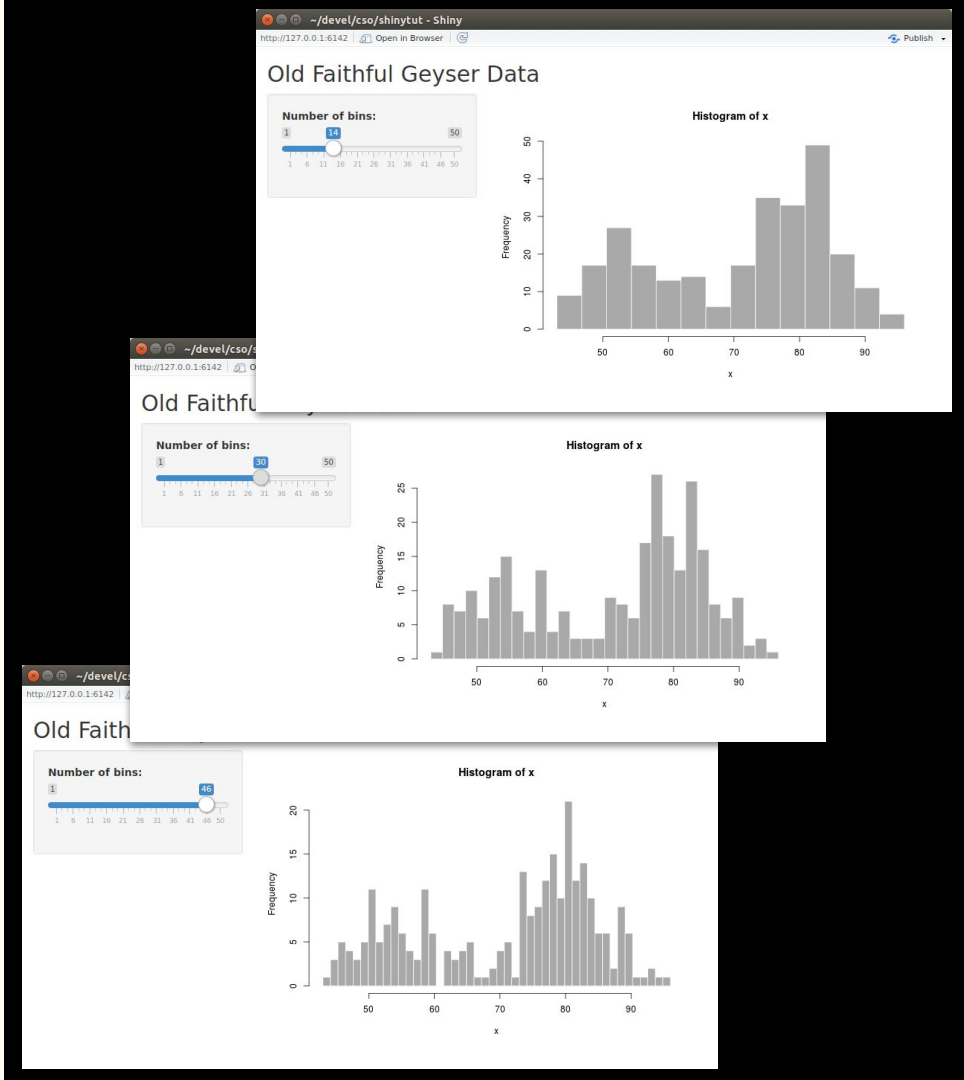
```r
# Shiny User Interface definition
library(shiny)

shinyUI(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")
    )
  )
))
```

```r
# Shiny Server-side processing and plotting
library(shiny)

shinyServer(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')

  })

})
```

server.R

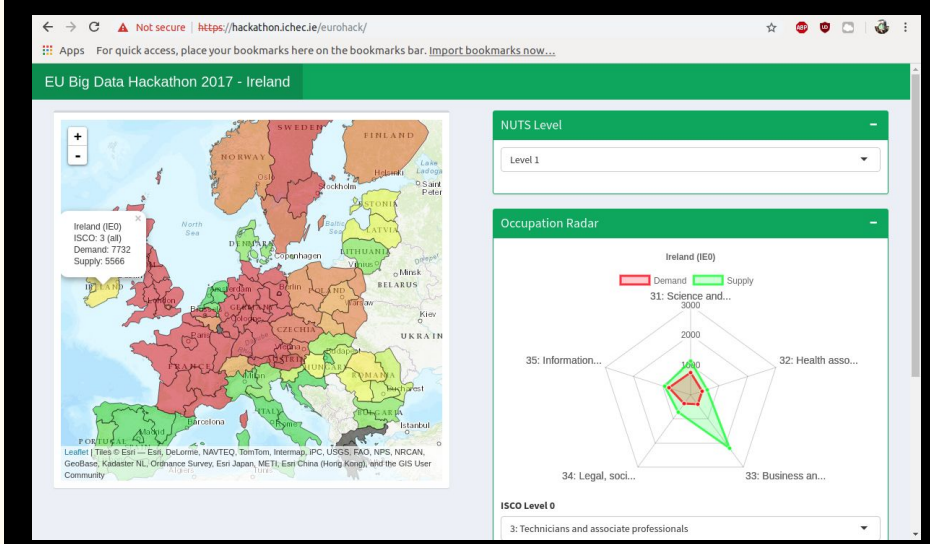
# Sample Shiny App

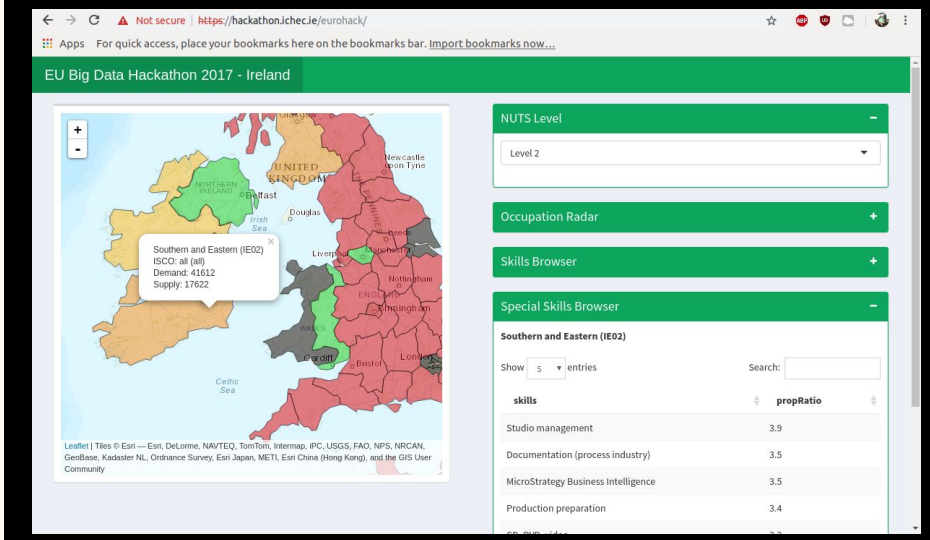
- Since the histogram depends on the value of the slider, each time the user moves it, the histogram gets redrawn.
- Number of bins, colours, axis variables, axis labels... Everything you can use for a plot in R can be tied to Shiny widgets.
- Not limited to R basic plotting: you'll likely want to use ggplot2 to draw your figures.



# Sample Shiny App: EU Big Data Hackathon 2017

- ggplot2 used for plots, using the various widgets seen around the application to decide which shapefile to use or how to colour the regions.
- This is a typical case of not knowing what to show, hence letting the user decide.
- 3 days effort by a team of 3: Shiny is really easy to use for R users.



# R/Shiny: live examples

- Shiny Gallery at R Studio:

  https://shiny.rstudio.com/gallery/

- EU Big Data Hackathon 2017 (Ireland's entry):

  https://hackathon.ichec.ie

(Yes, it's safe to ignore the security warning due to an old invalid SSL certificate) :)

# Parting thoughts

- We have tools that go well beyond the capabilities of printed media.
- Those tools are not yet for everyone but are already easy enough to be used by your average statistical programmer. No need for web design experience to do simple things.
- Wider adoption could lead to a culture change in how we present data: prepare predefined views but allow further exploration/experimentation.
- If a paper/report/article presents results from open data processed with R/Python, why *not* provide the Jupyter notebook which contains the commented code and results? Think reproducibility, user experimentation, learning resource...