Creating Interactive Visualisations Using R and Shiny

Course Overview

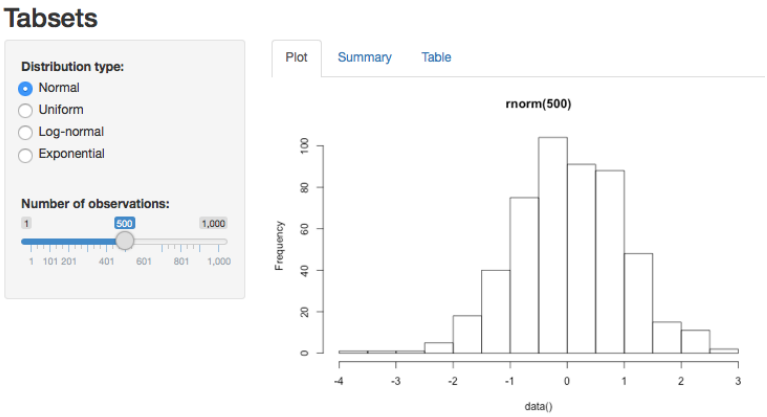
This course is designed as a broad overview of how presentations and interactive content can be built using the R language and R Markdown.

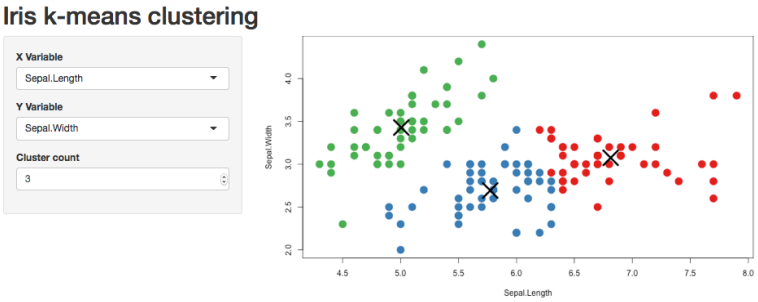
No prior knowledge of R is expected, however this course will **NOT** provide a sufficient overview of the R language to start analysing data and doing useful stuff.

- This course shows you how to make Shiny stuff.

Scope of Course

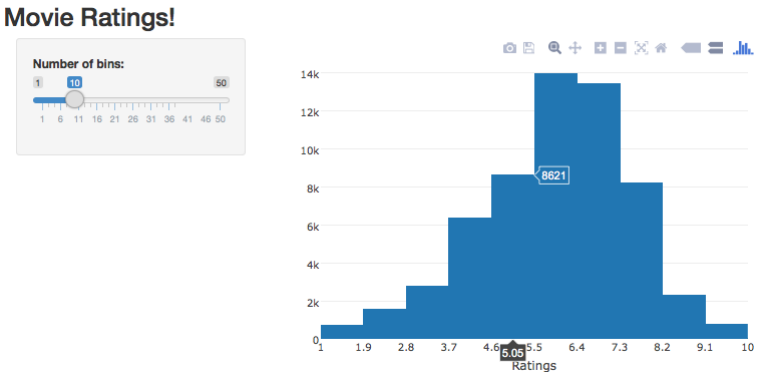
This course will introduce the necessary skills to build interactive elements similar to the following examples from the shiny.rstudio.com gallery.



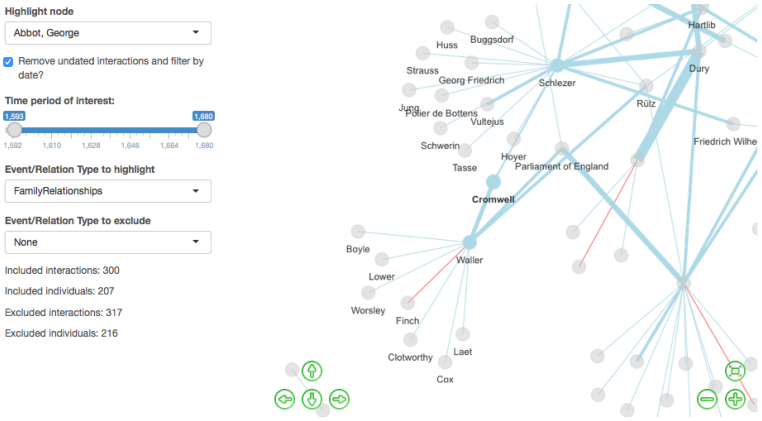


Scope of Course

The very basics of using supplementary packages for creating Shiny apps with highly interactive charts (ie. pan, zoom and tooltips) will be introduced.



The course does not cover the development of this type of applications:



Course Materials and Structure

The lecturer notes for this course have been written in R Markdown and will be provided to you at the end of the course.

There are a number of exercise files provided for you on the desktop in the folder “R and Shiny”.

The course will cover the following topics:

- The R Language
- RStudio
- R Markdoown
- RPub
- Shiny

The R Language

R is a scripting language and a very powerful tool for data analysis and presentation, primarily due to the huge user base and their dedication to developing free and open source libraries/packages covering a vast range of different knowledge domains:

- Regression
- Machine Learning
- Image Analysis

- Network Analysis

The Comprehensive R Archive Network (CRAN) is the canonical repository for R packages, note that almost all* packages hosted on CRAN may be used within a Shiny app.

**Packages dependent on parallal or distributed computing are unlikely to be supported, contact shinyapps-support@rstudio.com with any questions*

Learning R

Oxford provides a variety of courses on R in the ITLP Catalog, if there are additional courses you would like to see run please do ask!

Oxford also provides access to lynda.com for all staff and students at the University, Lynda provides training material on a whole host of programming languages and analytical techniques.

Datacamp.com is a tool for learning the R language and analytical techniques directly in the browser - note that a subscription is required to access some content.

Getting Help

Stackoverflow is the Q&A community for programming and scripting, this is a good place to start when trying to solve a problem. Note that the community may see abrasive to new users, the following advice is useful to prevent feeling like your fingers have been snapped off:

- Search for a solution before asking a question
- Provide a reproducible example of your problem

The R Console

R is the name of the programming language and *console* within which many users of R write and evaluate their code.

To use R on your local machine you must download and install the R Console, it's available on Windows, OS X and Linux.

Like all consoles, this application provides (only) the following functionality:

- Write code and script files
- Evaluate code and script files

RStudio is a free, open-source IDE that provides an extremely powerful literate programming environment for using the R language.

Literate programming environments gives programmers the freedom to write their code in a human understandable manner - allowing the programmer's approach to the problem to be embedded into the code.

In modern environments like RStudio, a literate programmer is able to include explanatory text, input and output code, images and even interactive elements.

Reproducability

Reproducability is a hot topic in research today, though the buzzwords Open Access and Open Data are usually used in its place.

Literate programming environments like RStudio are a boon to reproducable scripting practices.

When writing code always consider whether the operation you're encoding might be performed again in future, or could be generalised. The following steps will help:

- Always write comments
- Try to always convert scripts into functions
- Always test code
- Don't code if you don't need to

Projects

RStudio has a great "projects" feature that makes it easy to contain code, data and output together in a structured manner.

Projects are very useful when working on multiple or long-term projects. In the exercises after this you will create your own projects and become more familiar with the RStudio interface, but the following features of the projects functionality should be highlighted before you start:

- RStudio attempts to save session info when you quit
- Projects provide a sensible location for these files to be saved and are reloaded whenever a project is opened

Scripting environments depend on the user specifying file locations for import/export, to make this easier most user will specify a "working directory" allowing only file names to be specified.

- Projects save the last specified working directory in .RHistory
- When re-opened, projects will update the working directory of the session to what was last used in the project

Note that in our training course there will be some additional complexities as RStudio is being run from a USB.

Exercises (20mins)

These exercises will assist in you in becomming familiar with RStudio Projects and ensure you have the necessary understanding of the R language to build Shiny apps later in the course.

What is Markdown?

What is Markdown?

Markdown is a very simple and widely used markup language - it allows documents to be described and then generated through an interpreter.

R Markdown allows the specification of documents (reports, presentations, etc) which include both code and evaluated code output - let's discuss the types of RMarkdown document before introducing the basic syntax and starting some exercises.

Types of R Markdown

There are three types of RMarkdown output

- PDF files are useful for when writing academic reports, lecture notes or other material that must be printed
- HTML files are useful when wanting to share your content online, perhaps via RPubs which will be discussed later
- Generating word files from RMarkdown is beyond the scope of this course

Presentations with RMarkdown

This course focuses on presentations rather than reports, as reports are traditionally printed materials that wouldn't be suitable for interactive Shiny apps.

There are three types of presentations you can create:

Beamer outputs a PDF file with buttons that can be used in most PDF viewers for navigating through slides.

ioslides outputs a highly stylised HTML file with a transition effect between slides and a "letter boxing" effect around your content. Contents can be shown by pressing the C key

slidy outputs a HTML file with significantly more flexibility than ioslides. Contents can be shown by pressing the C key.

RPubs

RPubs.com is a free and open platform for publishing and sharing HTML RMarkdown documents with others.

RPubs and Shiny are both maintained by the team behind RStudio and are fully integrated into the IDE.

Note that while documents hosted on RPubs are made public, the underlying code/.rmd files cannot be downloaded by others.

Markdown Syntax

There are many different flavours of Markdown but they all share the same basic syntax, which can be spit into the following types of actions:

- Laying out documents
- Formatting Text
- Including images and hyperlinks
- Include code and code output

Laying out documents

The preamble of RMarkdown documents specifies the overall layout of the document (i.e. slidy or ioslides), the only other layout directly controllable with Markdown is *headings*

Heading

Subheading (New Slide)

Subsubheading

Formatting Text

Basic text formatting is achieved using syntax that you may have used in other text editors, or in comment sections on websites:

italics

bold

Bullet point and enumerated lists are also added easily:

- Bullet point 1

- Bullet point 2

1. First Numbered Item
2. Second Numbered Item

It's important to remember that as your text is first interpreted as code, rather than text to display, it's necessary to "escape" characters that would otherwise be interpreted - for instance *this isn't italic*

URLS and Images

URL links are simply written as

```
[text to show](http://google.com)
```

Images can be embedded into RMarkdown documents from the web or from your local machine - the exercises will introduce you to the relative filepaths necessary to include local files

```
!(image)[image-link]
```

HTML for Power Users

Markdown is great for quickly specifying your document/presentation layout, but it is not designed to be flexible or extensible - it's a minimal set of instructions for styling your content.

If you're generating HTML from RMarkdown it's possible to simply write HTML directly in your .rmd files.

Slidy and ioslides both utilise the Bootstrap CSS so it's easy to include responsive content in your presentations.

Code Chunks

Code can be inserted into RMarkdown files in one of two ways: inline or as code chunks.

Code to be shown (but not evaluated) is written as ‘ 2+2 ‘ - to evaluate code use ‘ r 2+2 ‘

Code chunks are deliminated as follows:

Naming Chunks and Chunk Options

Naming code chunks make it easier to diagnose issues with the knitting of your RMarkdown documents into HTML or PDF files.

Chunk names are given as follows:

Exercises (10 mins)

Shiny and shinyapps.io

Shiny is an R package that allows interactive HTML elements to be generated (and *rendered*) from interpreted R code.

To display Shiny content in a webpage the R code must be interpreted by a Shiny-enabled server - shinyapps.io provides this as a service.

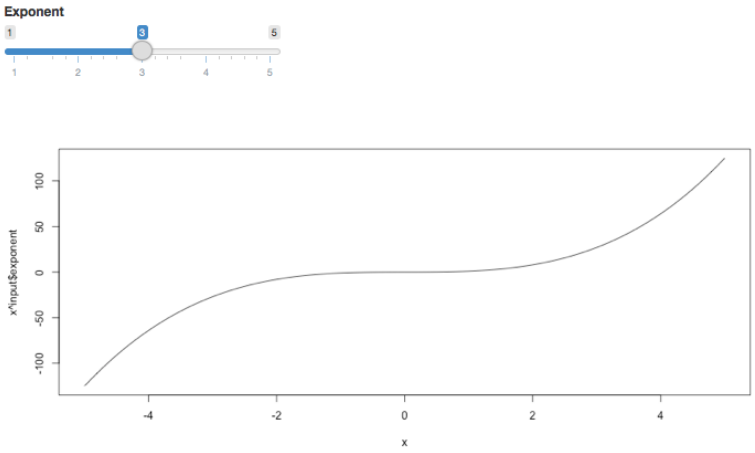
However it is very simply to interact with Shiny apps on your local machine without relying on such services. In the exercises you'll create your own shinyapps.io account and upload a Shiny app to the web, but for the next few slides we'll consider only what you need to do to build an interactive Shiny app on your local machine.

First Shiny App

To use shiny it's necessary to install the shiny library as follows, and make sure that it is "loaded" using the `library` function

```
install.packages("shiny")
library(shiny)
```

The following interactive app will now be built by your lecturer and the mechanics of it explained.



Why UI and Server?

Shiny apps are split into two sections - the UI and server side code.

It's important to understand what Shiny does - it provides a way to write a HTML GUI (i.e. HTML and JavaScript) in R code and provides a framework for a browser displaying a Shiny app to exchange information with a Shiny server.

The app is therefore split into client-side and server-side code - or shinyUI and shinyServer. Fundamentally, the UI is only aware of "values" (or data) from the server assigned to the output object.

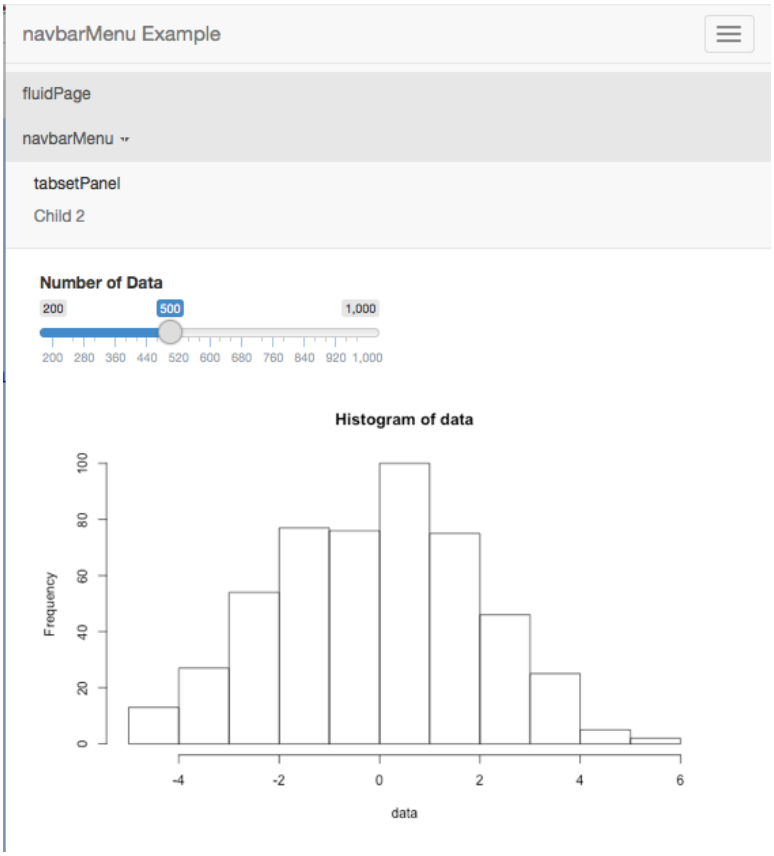
Exercises (15mins)

Bootstrap and Shiny apps

Bootstrap is an incredibly powerful and flexible framework for making "responsive web elements" - content that resizes (or transforms) dependent on the window size of your browser.

The following interface is built entirely within Shiny using the following layout tools:

- navbarPage
- navbarMenu
- tabsetPanel



Exercises (10mins)

Warnings about Shiny

- Controller variables can only be used once in a shiny app - `sliderInput("thisVar", ...)` only one slider can be creater in the app that uses `input$thisVar`
- Duplicate output cannot be used in shinyApps, it will create errors. So you cannot have multiple instances of `plotOutput("myFancyPlot")` in your app

Interactive Data Visualisations

So far we've built a Shiny app with interactive elements, but not interactive charts.

These can be easily created with a variety of additional libraries, let's consider the following packages:

- plotly
- visNetwork

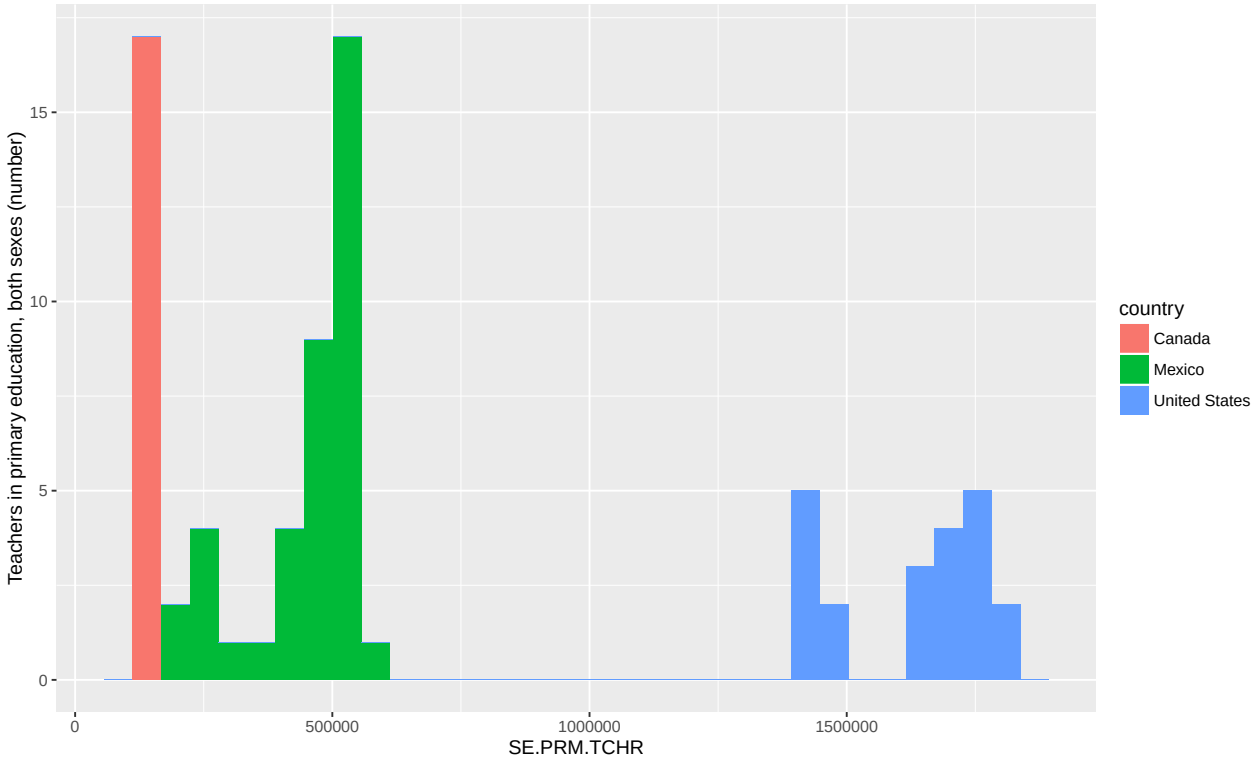
Interactive Charts

One of the really great things about shinyapps.io is that it's possible for your shiny app to pull data from external (including live) sources via APIs, CURL and other mechanisms. It's even possible to pull data from secure sources using the `ROAuth` library.

As an example, and in the exercises, we'll use the `WDI` library for accessing data from the World Bank.

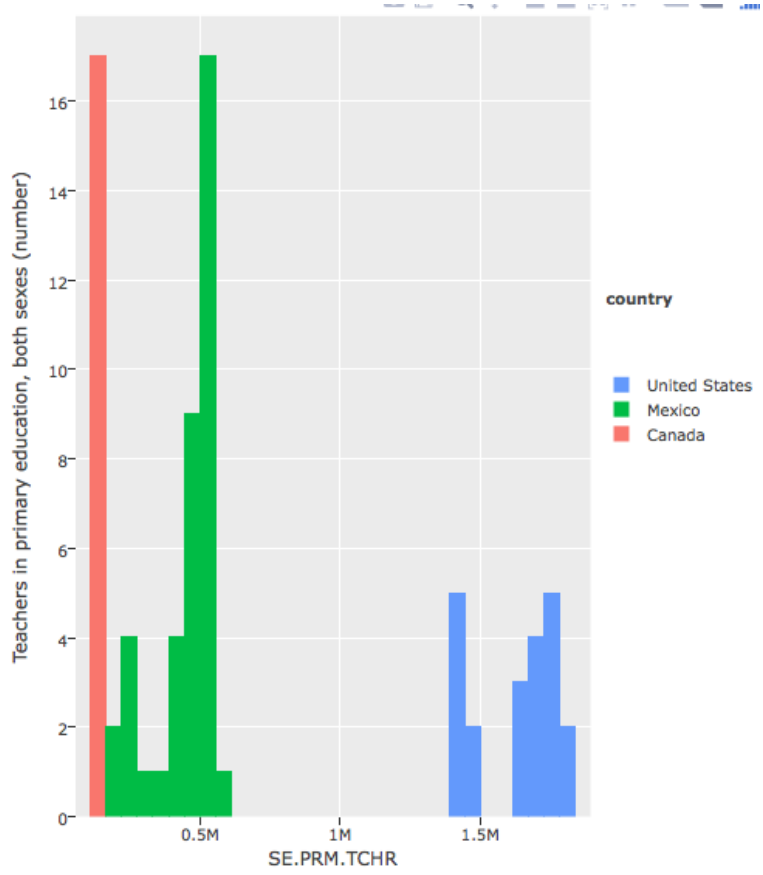
`ggplot` is a great graphics language but entirely beyond the scope of the course - we're simply using it to display data:

```
library(WDI)
dat <- WDI(indicator='SE.PRM.TCHR', country=c('MX','CA','US'), start=1960, end=2012)

library(ggplot2)
ggplot(data = dat, aes(x = SE.PRM.TCHR, fill = country)) + geom_histogram() +
  ylab("Teachers in primary education, both sexes (number)")
```



While this is a fairly attractive plot - it's not very interactive! The `plotly` library allows *many* `ggplot` visualisations to be converted directly into interactive charts which could be used in a Shiny App

```
library(plotly)
ggplotly(ggplot(data = dat, aes(x = SE.PRM.TCHR, fill = country)) + geom_histogram() +
  ylab("Teachers in primary education, both sexes (number)"))
```

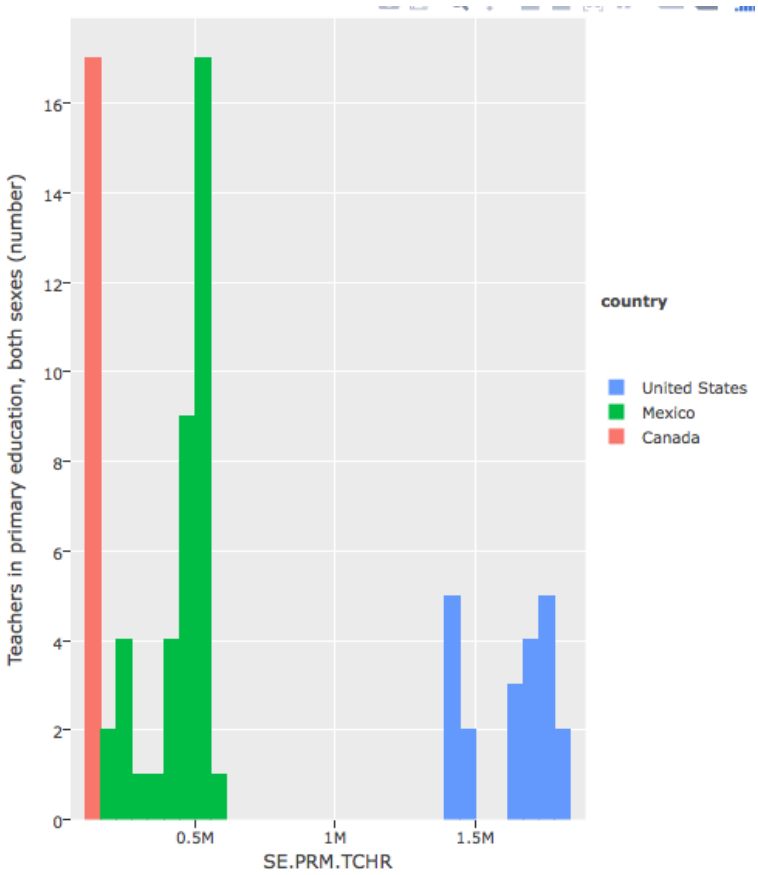


Reactive Expressions in Shiny

So far our shiny apps have simply had interactive controls and simple curves, to support more complexity in your shinyApp you must add reactive expressions.

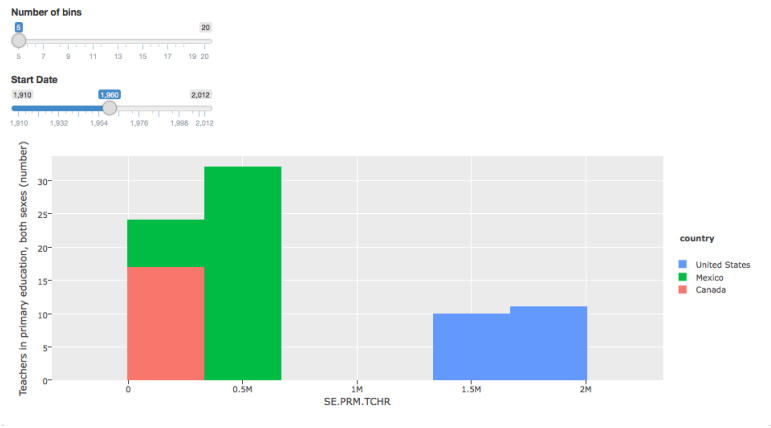
In the visualisation we just made it would be useful to specify the binwidth or number of bins for the histogram:

```
ggplotly(ggplot(data = dat, aes(x = SE.PRM.TCHR, fill = country)) + geom_histogram(bins = 10) +
  ylab("Teachers in primary education, both sexes (number)"))
```



A shiny app for this could be written as follows but it is very inefficient as the data is reloaded from the World Bank everytime the input$start.date variable is updated.

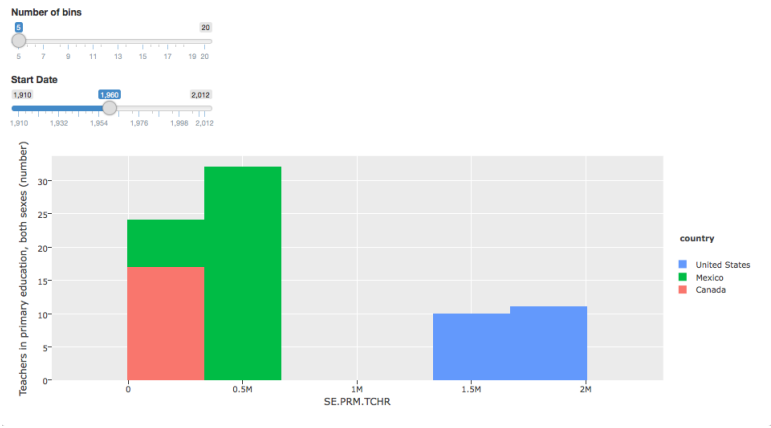
```
shinyApp(
  ui = fluidPage(
    sliderInput("no.of.bins", label = "Number of bins", min = 5, max = 20, value = 5),
    sliderInput("start.date", label = "Start Date", min = 1910, max = 2012, value = 1960),
    plotlyOutput("plotly.plot")
  ),
  server = function(input, output){

    output$plotly.plot <- renderPlotly({

      dat <- WDI(indicator='SE.PRM.TCHR', country=c('MX','CA','US'), start=input$start.date, end=2012)

      ggplotly(ggplot(data = dat, aes(x = SE.PRM.TCHR, fill = country)) + geom_histogram(bins = input$n
                ylab("Teachers in primary education, both sexes (number)"))
    })
  }, options = list(height = "600px")
)
```



Reactive Expressions in Shiny

`reactive` expressions only update when input variables contained within them are updated, for instance data will only be reloaded from WDI here when the `input$start.date` is changed

```
shinyApp(
  ui = fluidPage(
    sliderInput("no.of.bins", label = "Number of bins", min = 5, max = 20, value = 5),
    sliderInput("start.date", label = "Start Date", min = 1910, max = 2012, value = 1960),
    plotlyOutput("plotly.plot")
  ),
  server = function(input, output){

    data.for.plot <- reactive({
      WDI(indicator='SE.PRM.TCHR', country=c('MX','CA','US'), start=input$start.date, end=2012)
    })

    output$plotly.plot <- renderPlotly({

      dat <- data.for.plot()

      ggplotly(ggplot(data = dat, aes(x = SE.PRM.TCHR, fill = country)) + geom_histogram(bins = input$n
                ylab("Teachers in primary education, both sexes (number)"))

    })

  }, options = list(height = "600px")
)
```

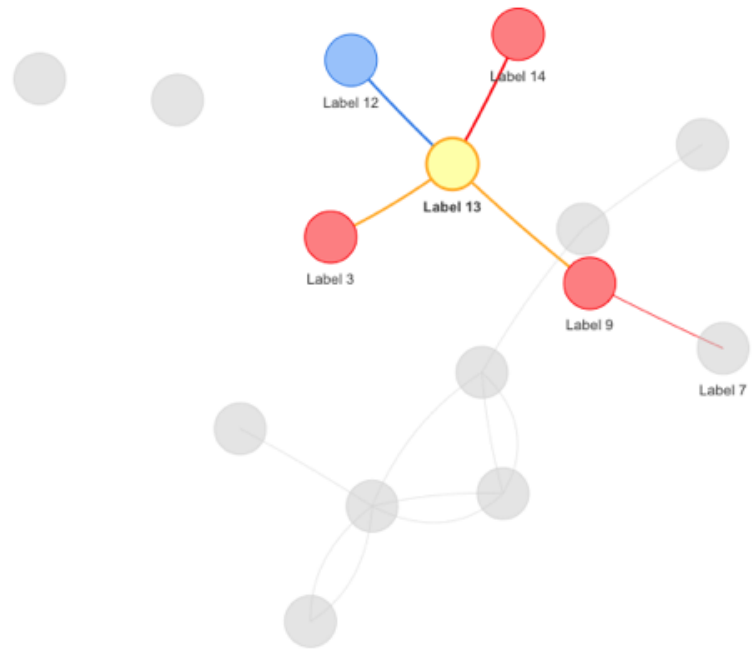


Going further with plotly and more

Plotly allows for the creation of a wide variety of interactive charts, as well as cartograms. An example template app using plotly cartograms is available here

`visNetwork` is a great library for creating interactive networks, for instance:

```
library(visNetwork)
nodes <- data.frame(id = 1:15, label = paste("Label", 1:15),
 group = sample(LETTERS[1:3], 15, replace = TRUE))

edges <- data.frame(from = trunc(runif(15)*(15-1))+1,
 to = trunc(runif(15)*(15-1))+1)

visNetwork(nodes, edges) %>% visOptions(highlightNearest = TRUE)
```



Exercises (20mins)

Split File Shiny Apps

In this course we have only considered how to build Shiny apps directly within an RMarkdown chunk, however this is only really applicable for small-scale Shiny apps.

These apps we've been building could be considered "self-contained" as they are written in a single call to the `shinyApp` function, large scale apps are developed across at least two files:

- ui.R
- server.R

This "split-file" Shiny app setup allows for much greater "separation of concerns" and make development (and reuse) of an application easier.

Embedding "Split File" Shiny Apps into Presentations

There are two methods of emebedding split file apps into RMarkdown presentations:

- `shinyAppDir`: requires the entire RMarkdown document to be in the shiny runtime, i.e. shinyapps.io active hours are consumed while anywhere in the presentation
- iframes: allows a shinyapps.io hosted Shiny app to be embedded into a single (or many) slides without the rest of the app requiring Shiny active hours to display

Layout of Split Files

The ui.R and server.R files for a split file Shiny App must be contained within the same directory, and have the following structure

```
## ui.R
library(plotly) # You must load libraries into the ui.R file when they are used in client-side interact
shinyUI(
  fluidPage(
    ...,
  )
)
```

```
## server.R
library(plotly) # You must load libraries into the server.R file when they are used in server-side inte
shinyServer(function(input, output){
  ...,
  })
```

Loading local files into split file apps

This is largely beyond the scope of this course, however it is useful to have some advisory rules:

- CSS and JavaScript should be kept in the `www` sub-directory of the app

- Images should be kept in the `images` sub-directory of the app
- All paths must be given relative to the ui.R and server.R folder (the root directory of the app)

Note that ALL files in the Shiny app directory will be uploaded to shinyapps.io if you choose to deploy your app there.