Exercises

Exercises: htmlwidgets and RPubs

Overview of Exercises

The interactive web is driven by JavaScript*, the majority of interactive elements that you use on websites are written in JavaScript - from interactive maps to auto-completing pop-up menus. Like in R, there are hundreds of different JavaScript libraries dedicated to various visualisation tasks. There is a tool called `htmlwidgets` that allows R developers to easily build bindings to JavaScript libraries, allowing incredibly rich interactive content for the web to be built just with the R language.

These bindings to JavaScript libraries are typically distributed as individual R packages; an individual R package for an individual JavaScript library. The htmlwidgets.org website provides a showcase of some of the `htmlwidget` dependent bindings that are available through CRAN.

Important: There will be functions mentioned in these tutorials that you may not have used before and some descriptions may appear deliberately misleading. However, they are an attempt to get you to think about how your code is constructed. Please do ask questions at any point!

Worked examples

The tutor will work through a number of worked examples on the projector, utilising the following datasets. Note that these are also provided in the `htmlwidgets.R` file and you are advised to copy and paste them from there and NOT from this PDF. Copying code from a PDF into a script file is a recipe for disaster, there are likely hidden characters and all sorts of nastiness.

```
africa_data_points = data.frame(
  lat = rnorm(26, mean = 6.9, sd = 10),
  lng = rnorm(26, mean = 17.7, sd = 10),
  size = runif(26, 5, 10),
  label = letters
  )
```

```
## WDI data
library(WDI)
world_data <- WDI(country=c("GB","EG","SA","EE","CA"), indicator=c("EG.GDP.PUSE.KO.PP","EN.ATM.CO2E.PP.
                  start=1990, end=2000)

wdi_codes <- list(
  "EG.GDP.PUSE.KO.PP" = "GDP per unit of energy use (PPP $ per kg of oil equivalent)",
  "EN.ATM.CO2E.PP.GD.KD" = "CO2 emissions (kg per 2005 PPP $ of GDP)",
  "NY.GDP.MKTP.KD" = "GDP (constant 2000 US$)"
)
```

Exercise 0: R language checks

If you feel you would benefit from some revision of the basics of R syntax, then you are invited to complete the following exercise, otherwise please continue to Exercise 1.

Shiny is very easy to use but does expect knowledge of the basic R language - particularly an understanding of the different types of brackets and assignments. Many new users of R feel frustrated because of confusion about what brackets are for, to ensure that in later exercises you can build Shiny apps please consider the following guide:

- Round brackets () encapsulate the arguments for a function, in the case of `rep("Hello World", 2)` the round brackets encapsulate the two arguments passed to the function `rep` - arguments are therefore deliminated by commas.
- Square brackets [] are used for extracting parts (rows, columns, individual elements) from data structures - that's there only use
- Braces {} are used for containing expressions - when writing mathematical expressions by hand round brackets are usually used for controlling precedence (order of operations), but in R you should write 2*{x+1}^2.

Braces are necessary where *more than one thing* is being done in an individual argument

```
## [1] "strings" "strings" "strings" "strings" "strings"
```

With this in mind, work through the exercises in "Scripts-to-Fix.R.

Exercise 1: interactive map

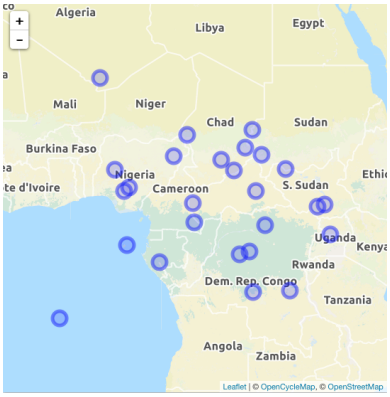
Start a new script file for this exercise with appropriate comments.

1.1 Create a basic `leaflet` map wth the following code:

```
leaflet() %>%
  addTiles()
```

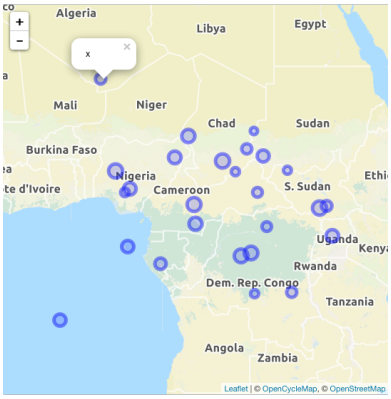
1.2 Refer to rstudio.github.io/leaflet/basemaps.html to change the map to use the attractive "Thunderforest.OpenCycleMap" tiles

1.3 Combine the code from 1.2 with the code for visualising `africa_data_points` to obtain something similar to the following visualisation



1.4 In the `plotly` worked example it was shown how to access columns from a `data.frame` within a `htmlwidget`. Use this knowledge to add two features to the map:

- scale the size of the circles by the size column of the dataset
- add a tooltip (also called popup) to the circles that shows the label of each point



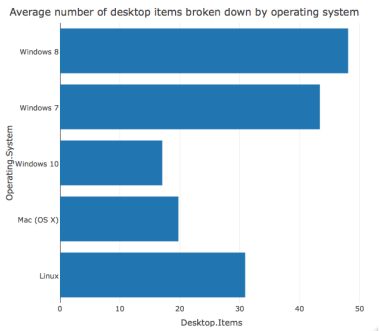
1.5 Consult the documentation for `addCircleMarkers` and find out how to cluster the circles as you zoom out.

Exercise 2: interactive chart

This exercise uses an example dataset deposited on Figshare about the number of desktop items on University member computers, dx.doi.org/10.6084/m9.figshare.3425729. Please consider adding to the dataset on your own machine - http://goo.gl/forms/IehEi6dyCEBIlbXW2.

Start a new script file for this exercises with appropriate comments.

In this exercise you will simply create a [relatively boring] bar chart with `plotly` that looks like this:



2.1 Use the function `read_csv` from `readr` (a part of the `tidyverse`) to import the data and store it against an appropriate variable.

2.2 A good feature of `read_csv` is that it preserves spaces in column names, a bad feature of `formula` is that they doen't handle these. Use the function `make.names` to sanitise the `colnames` of your variable.

3.3 Using the `dplyr` library [part of `tidyverse`] perform the following actions, note that you may want to refer to the cheatsheet under Help in the menubar:

- Select only the columns containing the number of desktop items and the operating system
- Group the data by the Operating System
- Mutate the column containing the number of desktop items to contain the mean of the number of desktop items (the previous grouping step will ensure this is a factored mean)
- Use unique() to ensure that duplicate rows are removed
- Store this subsetted dataset against an appropriate symbol

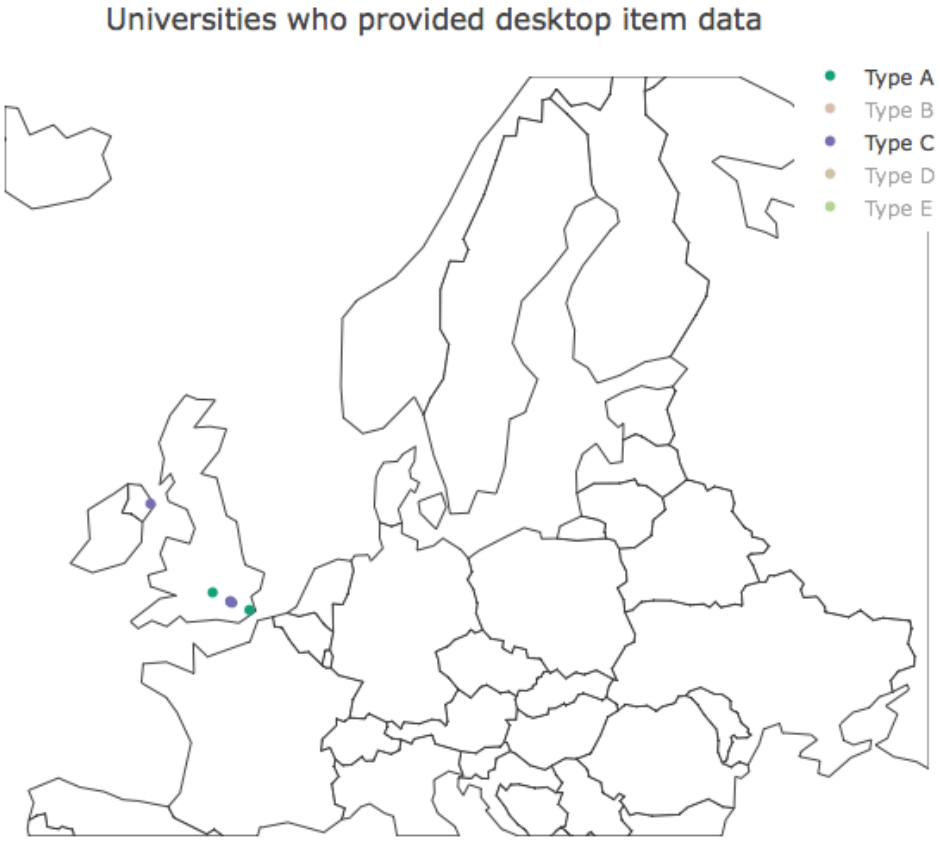
3.4 Provide this dataset to `plot_ly` with `type = "bar"` to generate a simple barchart

3.5 Pipe the chart into `layout` and specify an appropriate title for the chart.

3.6 If you wish to further modify the chart, refer to https://plot.ly/r/reference/

Exercise 3: maps with plotly

Leaflet is extremely powerful and you're advised to use that library for "geo-accurate" maps. If you're interested in "representative" maps then plotly (and highcharter) provides both choropleth and scattergeo functionality. This exercise is a basic introduction to scattergeos with plotly, you'll create the following map:



Start a new script file for this exercises with appropriate comments.

3.1 Use the function `read_csv` from `readr` (a part of the `tidyverse`) to import the data file at this url https://ndownloader.figshare.com/files/5449670 and store it against an appropriate symbol.

3.2 To specify a particular type of chart to plotly, you must use the argument `type`. Create a `scattergeo` chart from the dataset above, noting that rather than `x` and `y` you must specify the `lat` and `lon` columns.

3.3 The layout of a plotly chart is controlled with the `layout()` function, pipe the output from above into layout and provide an appropriate `title`.

3.4 There is a `geo` argument for `layout` to which a `scope` can be provided, refer to https://plot.ly/r/reference for an appropriate value for `scope` to display this data well.

3.5 The dataset doesn't include any groupings for the universities, add the following vectors as columns to your data.frame so that you can use them within the plotly map:

```
colors <- rep(c("#1b9e77", "#d95f02", "#7570b3", "#e7298a", "#66a61e"), 3)
labels <- rep(c("Type A", "Type B", "Type C", "Type D", "Type E"), 3)
```

3.6 Use `dplyr` to group the data.frame by the `label` column you just added, and pipe this into your map. Ensure that the `plot_ly` map uses this grouping by specificying two additional arguments:

- color: this is the column by which groupings should be detected
- colors: this is the column containing the colours for each group

Exercise 3: Network

Networks/Graph are used to study connections between entities, where these entities may be telephones and the connections between then SMS messages. The `htmlwidget` library called `visNetwork` is the most widely recommended tool for visualising this data, it is based on the vis.JS library and supports `igraph` objects to.

For demonstration purposes, we require a dataset that is not yet a graph but can readily be converted into one. We use the `quanteda` library to generate collocations for a given text, i.e. which words appear adjacent in a corpus.

3.1 Install and load the `quanteda` library

```
## quanteda version 0.9.8.3
```

```
##
## Attaching package: 'quanteda'
```

```
## The following object is masked from 'package:base':
##
##     sample
```

3.2 Use the following code to generate a collocation table (note: please copy and paste this from the accompanying htmlwidgets.R file)

```
nineteen_eighty_four <- c("Exactly. By making him suffer. Obedience is not enough. Unless he is sufferi

collocs_1984 <- collocations(nineteen_eighty_four, punctuation = "dontspan")
```

3.3 The object created by `collocations` is a data.table, which is a slightly different beast than a data.frame. We can pretty much ignore the differences for now, but it might be wise to convert it into one of the `tibbles` that the `tidyverse` uses:

```
collocs_1984 <- as_data_frame(collocs_1984)
```

3.4 The visNetwork library expects two data structures; a data.frame of nodes and edges. From the `collocs_1984` object create two objects with the following properties:

- Nodes: – id: Column containing unique ids for the nodes (can be strings) – label: Column containing labels for each node (displayed in networks) – title: Column containing titles for each node (displayed as tooltips in networks)
- Edges: – from: node from which an egde leaves – to: node into which an edge enters

3.5 Ensure you have the `visNetwork` library on your machine, consult the documentation for the `visNetwork` function and provide your data structures from above as the first two arguments of the function. NOTE: It may take some time to display the network with defaults, which is a shame.

3.6 The network takes a long time to display because the layout algorithm is inefficient. The `igraph` library is a fantastic tool for network analysis [written in C/C++] and provides a much faster layout alogirthm. Pipe your network from above into `visIraphLayout` to see the difference, this is typically a very good layout.

3.7 The `nodes` and `edges` objects can be provided with additional attributes to style the graph, add the following columns to your data structures and observe the change to the network when re-evaluated:

- edges: column called "width" populated with the "count" column from the `collocs_1984` collocations data.
- nodes: column called "color" populated with an appropriate vector containning named colours from this list: c("blue","red","green","purple")

3.8 Consult the documentation to modify your network accordingly (these are increasingly difficult):

- Prevent users from dragging nodes
- Change the nodes to be squares
- When a node is selected (clicked), highlight those nodes directly connected to your selection
- Make the edges curved lines
- Create a legend for your network

Exercises: RMarkdown for Presentations

Overview of Exercises

These exercises introduce you to the basic of RMarkdown and publishing to the Rpubs platform.

The process of interpreting your RMarkdown files and generating HTML/PDFs out is called "knitting" and uses (amongst other libraries), `knitr`. Knitting documents, and Shiny apps later, is done by pressing the "knit" button shown below - the exact text/image of the button will change dependent on what type of document you're working on.

From this point onwards you are asked to start each new exercise [within reason] as a new project, some sets of exercises require a different project per exercise. In this instance, however, just create one project for these exercises called "RMarkdown" or something similiar.

Exercises 1: New Script, New Project

Multiple instances of RStudio can be run at once - in RStudio parlence these are called *sessions* and are how you can work on multiple *projects* at once.

- On Windows, it is sometimes difficult to understand the difference between "instances" of applications and multiple windows within the same "instance". In general, RStudio runs in a single window with multiple files. Therefore, separate items in the task bar indicate distinct RStudio sessions.
- On OS X it is easy to differentiate between instances of an application (in general), separate dock itmes mean separate RStudio instances.

The following exercises will assist you in becomming familiar with this paradigm:

1.1. Open a new RStudio session

1.2. Create a new project, using either of the following methods:

- File -> New Project
- The Projects menu in the top-right of the screen

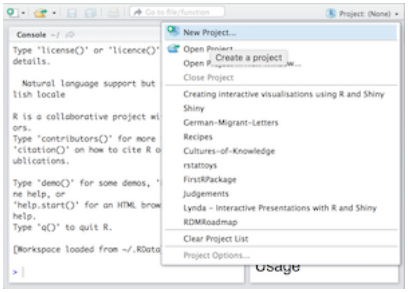


Figure 1: alt text

1.2.1 Choose "New Directory" and "Empty Project" 1.2.2 Browse to the desktop 1.2.3 Give your project a sensible name, like "RMarkdown"

1.3. Create a new script file with the shortcut CTRL+N or from the menubar select File -> New -> R Script

1.4. Type the following into your script file and evaluate it by pressing CTRL+Enter - in the "Environment" section of RStudio you should see that your variable has been assigned to the session environment

```r
myVariable <- rep("Hello World", 2)
```

1.4.1 To evaluate *specific lines* in a script, the lines must be selected 1.4.2 If nothing is selected then **only** the line with the cursor in it will be evaluated

1.5. Save the file and close the RStudio instance you opened - say yes to the save prompts

1.6. Reopen the project in a new instance of RStudio - if you have following the instructions correctly, the variable assigned previously will be in the project environment.

Warning and Sanity Check

Because projects store variables (data, functions and one off assignments) there may be examples of unexpected *old* variable definitions or data polluting your work. When working on large projects, and when attempting to diagnose bugs, it is a wise idea to check your environment and to "clear" the environment if necessary by using the broom icon in the environments panel.

Exercise 2

Ensure the project you just created is open before beginning this exercise.

2.1. Create a slidy RMarkdown document from File -> New -> RMarkdown

2.2. Change the slide headings to the following:

- Laying out presentations
- Formatting Text
- Including Code in Slides
- Obscuring Code

2.3. Change the 1st and 3rd slide titles to be a heading (#) rather than subheading (##) - how does this change the output presentation file?

Exercise 3

Images are embedded into RMarkdown files using the following syntax

`![alternative-text](image-path)`

Paths are *relative* to the RMarkdown document.

3.1. Add a new slide to your presentation with the heading "Images"

3.2. Create a subdirectory in the folder where your .Rmd file is saved called "images"

3.3. Save an image of your choice into this directory

3.4. Use the syntax above to embed this image into your new slide.

Exercise 4

4.1. Add a new slide to the presentation called "Code Chunk"

4.2. Use the keyboard shortcut (Ctrl+Alt+I or Cmd+Alt+I) to insert a new code

4.3. Write a script that will generate a vector of the squares of integers 1 through 10 (inclusive)

4.4. Provide an appropriate code chunk name and option so that the code AND output of the chunk is displayed in your presentation slides

Exercise 5

5.1. Select a `htmlwidget` example from the previous set of exercises, create a new slide explaining what the `htmlwidget` is about

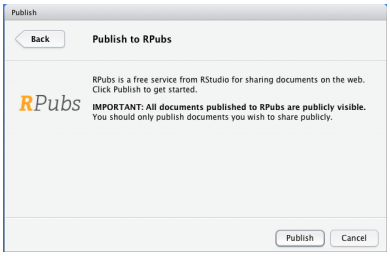
5.2. Add a code chunk that generates the `htmlwidget`

5.3. Ensure that the document knits together correctly.

Exercise 6

RPubs provides a free platform for publishing HMTL RMarkdown files, including those with embedded htmlwidget visualisations. The simplest way to connect your instance of RStudio to RPubs is to already have a HTML RMarkdown file ready - like the one you've just built.

6.1. Knit the presentation file and select "publish" in the top-right of the window, you'll be presented with this dialog - select "Publish"



6.2 Your browser will be directed to the "Publish Document - Step 1 of 2" page where you will be asked to create an account. When selecting your username be aware that URLs to your RPubs documents will have the form: rpubs.com/your-fancy-username/your-documents-name

6.3 Register your account and choose an appropriate title, description and slug for your document.

When these steps have been finished you'll have a published slidy presentation on RPubs.

Exercises: Shiny Basics

Overview of Exercises

This exercise is designed to consolidate the very fundamentals of a shiny app and the syntax required. They are minimal examples of what Shiny is capable of.

Use the same project for both exercises.

Exercise 1

1.1 Create a project (refer back to the notes in the RMarkdown section of the course if necessary) containing both a ui.R and server.R file, you may find it useful to use the template files provided to you.

1.2 If you followed along with the instructors live-coding example, use the code from there. Otherwise you will need to add the following to your files:

ui:

```
plotOutput("curvePlot")
```

server:

```
output$curvePlot <- renderPlot({
    curve(x,
          from = -5,
          to = 5)
  })
```

1.3 Add a `sliderInput` to the ui that will allow users to change a value from 1 to 5, with a default value of 3. Refer to the documentation or http://shiny.rstudio.com/reference/shiny/latest/ as necessary.

1.4 Modify the server so that the curve(x^slider) is generated

1.5 Run the app and ensure it works.

Exercise 2

2.1 Add a control to the Shiny app you built before that allows you to type your name

2.2 Use the "main" argument of `curve` to provide your name as the title of the plot

2.3 Use the paste function to combine the your name with the current value of the exponent, for instance: "Plot of x^4 (by Johanna Smith)"

```
paste("this", "that")
```

Exercises: Shiny Basics (Publishing)

Overview of Exercise

These exercises will take you through the steps necessary to connect your RStudio installation to your shinyapps.io account.

Exercise 1

1.1 Sign up for an account at www.shinyapps.io/admin/#/signup

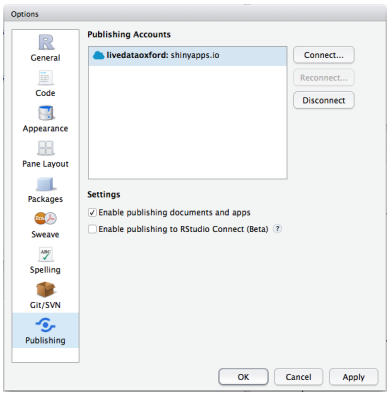
1.2 Choose an appropriate account name (this will be the "slug" for your shiny apps, i.e. http://fancyslug.shinyapps.io/myShinyApp)

1.3 Follow the steps on the website:

1.3.1 install.packages("rsconnect")

1.3.2 Copy your account info to the clipboard

1.3.3 Open RStudio's preferences, navigate to "Publishing" and click "connect"



1.3.4 Paste your account info into the dialog window.

1.4. If these steps have worked, you should be able to publish the shiny app you created in the previous exercise by clicking on the "Publish" button.

Exercise 2

Login to your shinyapps.io and investigate how you might achieve the following:

- View overall active hours across all applications
- Modify the timeout period for your shiny app
- Archive your app
- Download a copy of the shiny app from the server

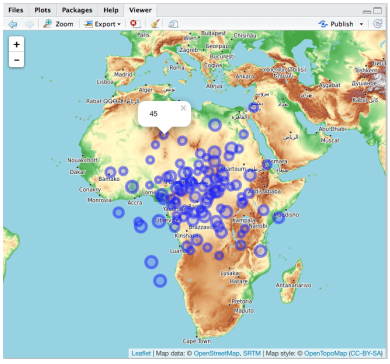
Exercises: Shiny and htmlwidgets

Overview of exercises

This collection of exercises is designed to introduce the basics of incorporating `htmlwidgets` into your shiny apps.

Exercise 1: Shiny Africa Map

Previously you created a leaflet map that looks something like the image below, we're going to create a Shiny app with an embedded Leaflet map - which means you will need to load these libraries in RStudio.



1.1 Create the shell of a shiny app from the template provided to you (a ui.R and server.R file)

1.2 Add the `leaflet` map generating code into the `server.R` file such that an output object will be created, that renders the map appropriately.

1.3 Display the map in the `ui.R` file using an appropriate function.

1.4 In the ui.R include the following controller:

- A slider that is allowed to move between 5 and 20, labelled "Standard Deviation"

1.5 Modify the `server.R` code such that changing the slider value changes the standard deviation of the `lat` and `lng` coordinates displayed, ensure that changing the slider affects the output map as you would expect.

1.6 Publish the shiny app to the web - ensure that it works!

Exercise 2: Selected marker

For this exercise you will need to refer to http://rstudio.github.io/leaflet/

2.1 In the `server.R` file create a new output UI element that will display the latitude and longitude coordinates of your selected points

2.2 Add appropriate code to the `ui.R` file such that this output object will be displayed.

2.3 Publish the shiny app to the web - ensure that it works!

Exercises: Laying out Shiny Apps

Overview of Exercises

These exercises introduce the various options available to you in laying out Shiny apps and consolidate the shiny app syntax.

You could reasonably continue within the same project as the last set of exercises.

Exercise 1: sidebarLayout

1.1 Continue with the code from the last exercise, or simply duplicate and start a new project

1.2 Add a `sidebarLayout` to the `ui.R` file such that the slider you created (and the selected map coordinates, if completed) are shown to the left of map.

Exercise 2: tabsetPanel

2.1 Add a `tabsetPanel` into the shiny app, with the leaflet map in the first tab.

2.2 Add a second `htmlwidget` (using a different library) from the earlier set of examples into the second tab of the `tabsetPanel`.

2.3 Publish the shiny app to the web - ensure that it works!

Exercise 3: navbarPage

3.1 Replace the fluidPage with a `navbarPage`, ensure the page with the `tabsetPanel` you've just created is shown when the app first loads.

3.2 Add a second tab to the `navbarPage` called "About" and add some "lorem ipsum" about the app.

3.3 You can republish this again for a sense of accomplishment, but previous requests to "publish and ensure it works!" are designed to ensre you've not missed out an important step. If you have a the `navbarPage` displaying correctly on your machine, congratulations I think you're ahead of the majority of people who have tried to use Shiny before!

Exercises: Shiny Reactivity

Overview of Exercises

These exercises are designed to introduce the concept of reactivity, to provide you the ability to control when and how an app updates.

Create a new project for this set of exercises, it's not necessary to have a different one per exercise.

Exercise 1

1.1 Duplicate the app from the last set of exercise you were able to complete, all we need is the map and slider to be working.

1.2 Add an additional control to the app that allows the user to select a few different map types from rstudio.github.io/leaflet/basemaps.html

1.3 Currently, changing the basemap will result in the data points being regenerated. Use `eventReactive` to ensure that the data is only updated when the standard deviation controller is modified.

1.4 Publish the shiny app to the web - ensure that it works!

Exercises: Shiny renderUI

Overview of Exercises

These exercises consolidat your understanding of reactivity and the overall communication of variables between the `ui.R` and `server.R` components of a shiny app

Use one project to contain all of these exercises, there are skeleton templates in the `shiny_renderUI` folder you'll find useful.

Exercise 1

1.1 Add a slider with the following specification, ensure the slider displays correctly before "plugging it in" to the `leaflet` map

- Label: Capital City population size
- Min: Minimum capital city population size
- Max: Maximum capital city population size
- Default values: 25% and 75% quartiles of the capital city population sizes

1.2 Ensure that the capital cities are now filtered according to this slider

Exercise 2

2.1 Add a slider with the following specification, ensure the slider displays correctly before "plugging it in" to the `leaflet` map. - Label: Modification date - Min: Earliest modification date - Max: Most recent modification date - Default values: 25% and 75% quartiles of the modification dates

2.2 Ensure that the capital cities are now filtered according to this slider

Exercise 3

3.1 Generate a ui element in `server.R` that displays a warning in a `wellPanel` if no capital cities match your filter conditions.

3.2 Display this warning where you wish in the app.

Exercise 4

It would be best for the "no data points!" warning to simply replace the map, as it most definitely signifies to the the user that there is no available data.

There are a few different methods you could use to conditionally show the wellPanel instead of the leaflet map, consult the documentation here http://shiny.rstudio.com/reference/shiny/latest/ for a potential method, but also consider whether you may be able to write a reactive ui element to achieve this.

–>