Exercises

Exercises: RStudio Interface, Projects and Basic R Syntax

Overview of Exercises

These exercises are designed to get you accustomed to the RStudio interface, creating and working on script files within the context of projects.

Projects should be used as early as possible in your use of RStudio as they provide a great way to deliminate between different projects and minimise issues with working directories.

Exercises 1: New Script, New Project

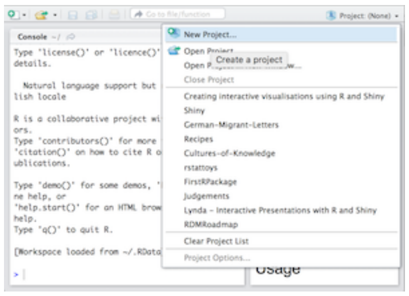
Multiple instances of RStudio can be run at once - in RStudio parlence these are called *sessions* and are how you can work on multiple *projects* at once.

- 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. Open a new RStudio session from the USB Stick (if there is not a shortcut, you will need to navigate to USB:/RStudio/bin/RStudio.exe)
2. Create a new project, using either of the following methods:

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


```
Type 'license()' or 'licence()'
details.

  Natural language support but
lish locale

R is a collaborative project wi
ors.
Type 'contributors()' for more
'citation()' on how to cite R o
ublications.

Type 'demo()' for some demos, '
ne help, or
'help.start()' for an HTML brow
help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData

>
```

Figure 1: alt text

2.1 Choose "New Directory" and "Empty Project" 2.2 Browse to the desktop 2.3 Give your project a sensible name, like "First-Project"

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

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

4.1 To evaluate *specific lines* in a script, the lines must be selected 4.2 If nothing is selected then **only** the line with the cursor in it will be evaluated

5. Save the file and close the RStudio instance you opened - say yes to the save prompts
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: Basic Syntax and Correcting Code

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

```
rep(
  "strings",
  {
    no1 <- 2
    no1 +3
  }
)
```

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

With this in mind - copy the file called "Scripts-to-Fix.R" into your project directory and complete the exercises therein.

Exercises: RMarkdown for Presentations

Overview of Exercises

These exercises will get you to go through the basic steps of creating RMarkdown slidy presentations and the necessary Markdown syntax for including code chunks and images in your presentations.

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.

Exercise 1

In the exercise folder on the desktop there is a file called First-Presentation-with-Errors.rmd

Open this file in RStudio and modify the file such that it can be knitted into a slidy presentation.

You'll need to look carefully and consult the Output and Issues section of the RMarkdown panel, as well as the preamble of the file.

Exercise 2

1. Create a slidy RMarkdown document from File -> New -> RMarkdown
2. Change the slide headings to the following:

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

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.

1. Add a new slide to your presentation with the heading "Images"
2. Create a subdirectory in the folder where your .Rmd file is saved called "images"
3. Save an image into this directory
4. Use the syntax above to embed this image into your new slide.

Exercise 4

1. Add a new slide to the presentation called “Code Chunk”
2. Use the keyboard shortcut (Ctrl+Alt+I or Cmd+Alt+I) to insert a new code
3. Write a script that will generate a vector of the squares of integers 1 through 10 (inclusive)
4. Provide an appropriate code chunk name and option so that the code AND output of the chunk is displayed in your presentation slides

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.

In these exercises you will use template `htmlwidget` visualisations to understand the breadth of what is possible, and how to publish these interactive visualisations to RPubs.

Please note that many of the packages depend on the "pipe" operator `%>%` - your instructor will provide an explanatiomn of how this works on request, but these exercises are not dependent on a deep understanding of the syntax.

Exercise 1

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 create a HTML RMarkdown file first.

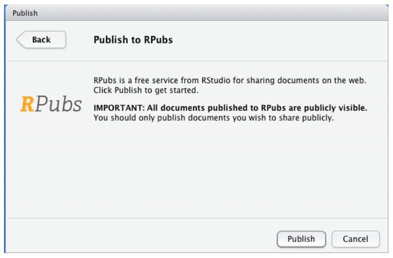
1. Presentation setup:

1.1. Create a new project for this set of exercises

1.2. Create a new Slidy presentation file in your project

1.3. Change the first slide title to "First RPubs Presentation"

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



Publish to RPubs

RPubs is a free service from RStudio for sharing documents on the web.
Click Publish to get started.

IMPORTANT: All documents published to RPubs are publicly visible.
You should only publish documents you wish to share publicly.

1.5 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

1.6 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.

Exercise 2

2.1 Add a new slide to the presentation file created in Exercise 1 with the title “Leaflet Map”

2.2 Use the `install.packages` function to install the `leaflet` package

2.3 Add a code chunk with the name “leaflet_world_map” and insert the following code:

```
m <- leaflet() %>%
  addTiles()
m
```

2.4 Evaluate the code in the script (select the code and press Ctrl+Enter), in the Viewer you’ll see a world map if the code is written correctly.

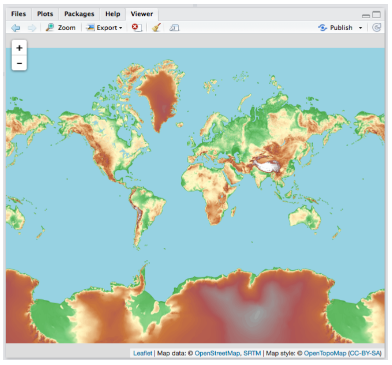
Exercise 3

The default `leaflet` map uses the OpenStreetMap basemap (the `addTiles()` argument), but there are many more available - see http://leaflet-extras.github.io/leaflet-providers/preview/index.html for a full list.

Add the following code to a code chunk in the slide created above:

```
m <- leaflet() %>%
  addTiles()
m
```

Replace the default basemap with the following code: `addProviderTiles("OpenTopoMap")`. If correct the map should appear like this:


Files
Plots
Packages
Help
Viewer
Zoom
Export
Publish
+
−
Leaflet | Map data: © OpenStreetMap, SRTM | Map style: © OpenTopoMap (CC-BY-SA)

Exercise 4

There are three basic types of cartogram one can create with the `leaflet` package:

- Scattegeo: A scatterplot overlaid on a map, with both size and/or colour available for communicating data. This is achieved with `addCircleMarkers`.
- Choropleth: Geographic areas are coloured/shaded according to some metric, this is achieved with `addPolygons`
- Weather Charts: Listed as a separate category only for the pedantic, achieved with a variety of functions including `addWSMTiles()`

This exercise covers scattergeo plots.

4.1 Add a code chunk to your slide with the name "africa_data_points" with it's chunk options set so that the output and code are not shown in the output presentation file.

4.2 Add the following code to the code chunk:

```
africa_data_points = data.frame(
  lat = rnorm(100, mean = 6.9, sd = 10),
  lng = rnorm(100, mean = 17.7, sd = 10),
  size = runif(100, 5, 10)
)
```

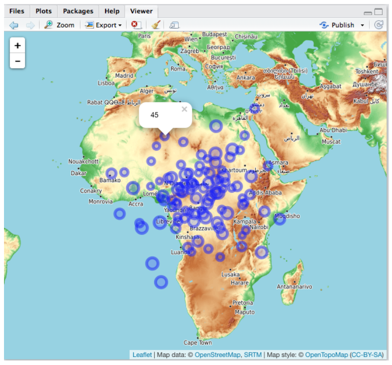
4.3 Add a new code chunk underneath called "africa_map" and add the map created in exercise 3.

4.4 Data can be made available to additional layers of a `leaflet` map through the use of `leaflet(data = my_data)`, modify your code chunk to make the `africa_data_points` data.frame available to your leaflet map.

4.5 The columns of `data.frames` available to your `leaflet` map are available by writing `property = ~column_name` - using this knowledge add the following to your code and provide the appropriate column names to the two properties:

```
m %>% addCircleMarkers(radius = , popup = )
```

If this has worked you'll have a map like this:


Files
Plots
Packages
Help
Viewer
Zoom
Export
Publish
45
Leaflet | Map data: © OpenStreetMap, SRTM | Map style: © OpenTopoMap (CC-BY-SA)

4.6 Knit together the presentation file and republish to RPubs.

Exercises: Basic shinyApp and shinyapps.io

Overview of Exercises

These exercises take you through the steps to write shinyApps within your RMarkdown presentations.

Exercise 1: Drawing Curves

1. Presentation setup:

1.1. Create a new project for these exercises - name it appropriately.

1.2. Create a new Slidy presentation file in your project

1.3. Change the first slide title to "Drawing Curves" and insert a new code chunk in which you'll write your shinyApp

1.4 Add `runtime: shiny` to the preamble of the presentation

2. Copy and paste the template for a Shiny app into your code chunk

```
library(shiny)
shinyApp(
  ui = ,
  server = function(input, output){

  }
)
```

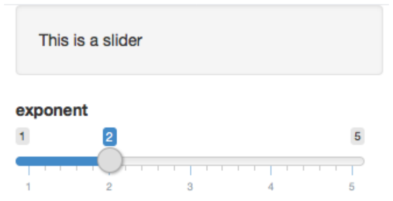
3. Add a fluidPage to the ui argument which contains the following arguments:

3.1 sliderInput with the following parameters:

- inputId = "exponent"
- label = "exponent"
- min = 1
- max = 5
- value 2

3.2 A wellPanel that contains text explaining what the slider is for

3.3 Knit your presentation together - you should see something like this

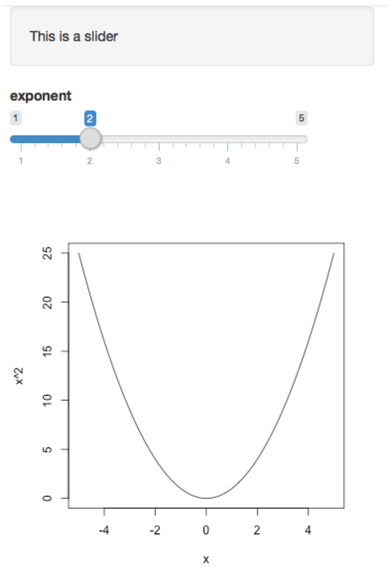


4. In the server function define a new variable for the output object (i.e. output$myThing) and assign it the following:

4.1 renderPlot

4.2 `curve(x^2, from = -5, to = 5)`

5. Add `plotOutput` to the ui argument and provide it the output object you defined above, knit together - you should have the following

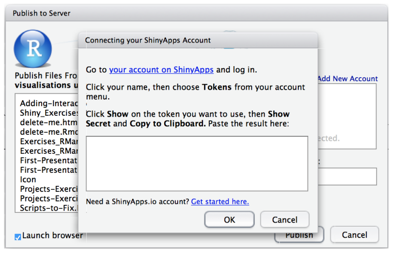


6. Modify the `curve` expression to be dependent on the input variable defined in `selectInput`
7. Knit together the presentation, the slider should update the plot now.

Exercise 2: Registering a shinyapps.io account

Publishing the shiny enabled presentation requires a shinyapps.io account - this is easy to set up.

Knit the presentation file and select "publish" in the top-right of the window, you'll be presented with this dialog - select "Get started here"



Publish to Server

Connecting your ShinyApps Account

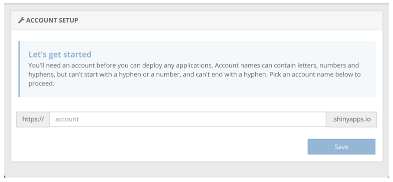
Go to your account on ShinyApps and log in.

Click your name, then choose **Tokens** from your account menu.

Click **Show** on the token you want to use, then **Show Secret** and **Copy to Clipboard.** Paste the result here:

Need a ShinyApps.io account? Get started here.

Register for an account - it's free. Choose an appropriate account name, this can be changed later but will require all your shiny apps to be re-deployed.



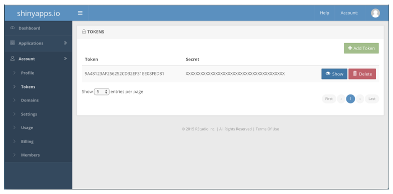
ACCOUNT SETUP

Let's get started

You'll need an account before you can deploy any applications. Account names can contain letters, numbers and hyphens, but can't start with a hyphen or a number, and can't end with a hyphen. Pick an account name below to proceed.

https:// account .shinyapps.io

Navigate to Account -> Tokens and select “Show” at shinyapps.io to expose your secret token, copy and paste this into RStudio.



Token	Secret
9A48123AF256252CD32EF31EE08FED81	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Show 5 entries per page

© 2015 RStudio Inc. | All Rights Reserved | Terms Of Use

Provide the shiny app with a name (this will be the URL for the shiny app - it therefore cannot contain spaces or anything else that is *unsafe* in URLs) and press “Publish” - your app will then be deployed to the web and your browser opened when it’s finished.

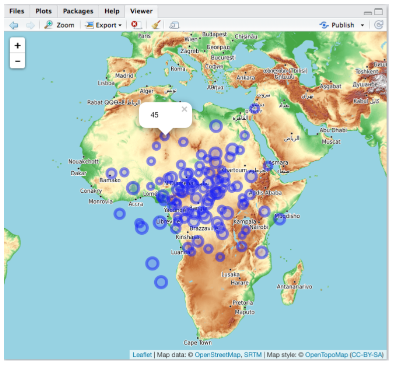
Exercises: Adding Interactivity to shinyApps and Reactive Expressions

Overview of Exercises

These exercises introduce the concept of reactive expressions - which are used to control when (and how) a Shiny app updates to user input.

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.



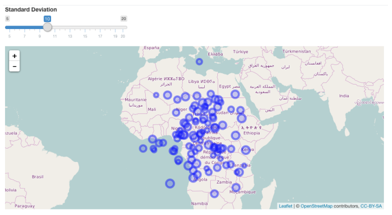
Create a Shiny app using the following template:

Add the following to the ui argument:

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

Wrap the following code in `renderLeaflet` and define a suitable output object:

Add `leafletOutput` to the ui argument - if correct then the when the Shiny app is generated a leaflet map will appear that looks like this:



Exercise 2: Non-Reactive Shiny Map

Let us add another controller to the shiny app built above, one that allows the user to select between different base maps for Leaflet. The full list of available maps is here - http://leaflet-extras.github.io/leaflet-providers/preview/index.html, but we're only interested in the following maps:

The standard basemap for leaflet is provided as follows:

To set a different base map you need to replace `addTiles()` as follows:

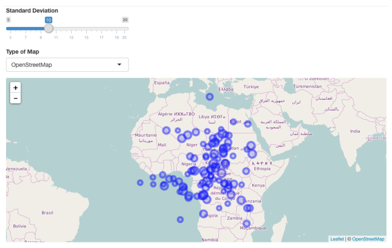
1. Modify your shiny app from the first exercise so that the base map is replaced with the `OpenTopoMap` shown above.

2. Modfiy your `renderLeaflet` expression so that changing the `sliderInput` value will change the standard deviation (the `sd` argument in `rnorm`) - if this works then the slider will increase the dispersion of points across the map.

3. Add to the ui argument a `selectInput` that can be used to choose between the three different maps listed at the top of this exercise.

4. Make the map's `addProviderTiles` argument dependent on the `selectInput` you just added.

5. At the end you'll have a visualisation that looks like this:



Exercise 3: Reactive Africa

In the shiny app above random data is generated when either control is selected - which is undesirable. The reason for this is that the `reactiveLeaflet` expression re-evaluates itself whenever the expression is updated, which at the moment is whenever a control changes.

The following application was created by the lecturer and demonstrates how interactive components of an app can be broken down into `reactive` expressions:

Modify your application so that it too is `reactive`, the following steps need to be followed:

1. Move the generation of the `data.frame` outside of `renderLeaflet` and assign it to a suitable variable (ensuring that it is `reactive`)

2. Reactive expressions must be "called" as if they are functions, i.e `my_reactive_thing()`. Ensure that your `reactive` expression is called as a function within `renderLeaflet`

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.

Exercise 1: sidebarLayout

1. Add a new slide to the presentation from before called "sidebarLayout"
2. Create a new (named) code chunk and copy paste your `curve` app into this
3. Wrap the contents of `fluidPage` in `sidebarLayout` and add both the `wellPanel` and controller to the `sidebarPanel` and the output plot to `mainPanel`

Exercise 2: tabsetPanel

1. Add a new slide to the presentation from before called "tabsetPanel"
2. Create a new (named) code chunk and copy paste your app from the above exercise into this
3. Add a new output object to the `server` argument which will visualise `curve(x^{1/exponent})` ensuring that it uses your input variable. Note the use of braces {} here to control the precedence of operations
4. Add `tabsetPanel` to the `mainPanel` of your app, and insert the following `tabPanel("Exponent", plotOutput("yourOutputObject"))`
5. Add another `tabsetPanel` and insert display the output of step 3

Exercise 3: navbarPage

Let's combine the two code chunks you've just written into a navbarPage

1. Add a new slide to the presentation from before called "navbarPanel"
2. Create a new (named) code chunk and add a template `shinyApp`, as a reminder:

```
shinyApp(
  ui = ,
  server = function(input, output){

  }
)
```

3. In the `ui` argument add a `navbarPage` to the following specification (refer to the documentation)

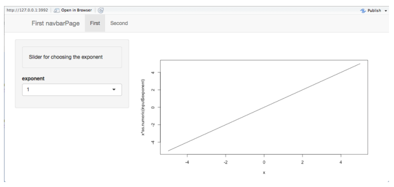
3.1 An appropriate title, for instance "First navbarPage"

3.2 Two `tabPanels`, with titles "First" and "Second"

4. Copy the `fluidPage` function (and its contents) from the exercise 1 and 2 and paste them into the "First" and "Second" `tabPanel`s, respectively.

5. Combine the body of the `server` functions of both exercise 1 and 2 into this code chunk

6. Knitting the apps together the output should look something like this:



7. Do both the controllers work? Try changing the app so that the `selectInput` controllers use different variables. If you're familiar with JavaScript, have a look at the source code for the shinyApp in a browser - why do you think the controllers might not work in the app made in step 6?