Data Analysis for Psychology in R 1

Department of Psychology, The University of Edinburgh

Academic year 2020-2021

Contents

Course overview

Course description

Data analysis for psychology in R 1 or, in short, **DAPR1**, is your first step on the exciting road of analysing data, and using data to answer interesting questions in the ways that statisticians, researchers, data scientists do.

This year-long course is designed to work gently through conceptual content that form the basis of understanding and working with data to perform statistical testing. It will provide you with the foundations in working with data and probability theory, and build to learning about how we make inferences about our hypotheses in psychology. Concepts are demonstrated in R throughout, taking you from basic calculations to the foundations of data management, plotting and use of simple statistical tests.

The overall aim of the course is to provide you with all the necessary skills to feel confident in the basics of statistics, R and data, before we move on to discuss a broader array of statistical methods in year 2.

Team

- Dr Tom Booth, *Senior Lecturer*: Tom.Booth@ed.ac.uk
- Dr Josiah King, *Senior Teaching Coordinator*: ug.ppls.stats@ed.ac.uk
- Dr Umberto Noe, *Senior Teaching Coordinator*: ug.ppls.stats@ed.ac.uk
- And not forgetting your friendly tutoring team! Ask them anything!

Schedule

SEMESTER 1

Week	Topic
1	Collecting data

Week	Topic
2	Types of data
3	Visualising distributions
4	Describing distributions
5	Visualising and describing relationships
Break	
6	Basics of probability theory
7	Probability rules
8	Random variables
9	Sampling variability and sampling distributions
10	Bias-variance trade-off

SEMESTER 2

Week	Topic
11	Bootstrap & Confidence Intervals
12	Hypothesis testing with the p-value approach
13	Hypothesis testing with the critical values approach
14	Hypothesis testing & Confidence Intervals
15	Making decisions - Effect sizes, Power, Errors
Break	
16	Test for one mean
17	Test for two means (paired samples)
18	Test for two means (independent samples)
19	Chi-square test
20	Covariance, correlation, and looking ahead

Textbook

TBC

Chapter 1

Collecting Data

Learning Objectives

- LO1: Install R & Rstudio, and get comfortable with the layout

- LO2: Learn about how to read in and store data in R
- LO3: Produce your first Rmarkdown document

You have two options for how you use R and Rstudio:

A: Download R and Rstudio onto your computer (recommended)
B: Use R and Rstudio online via a web browser (for people using chromebooks).

Checklist for today

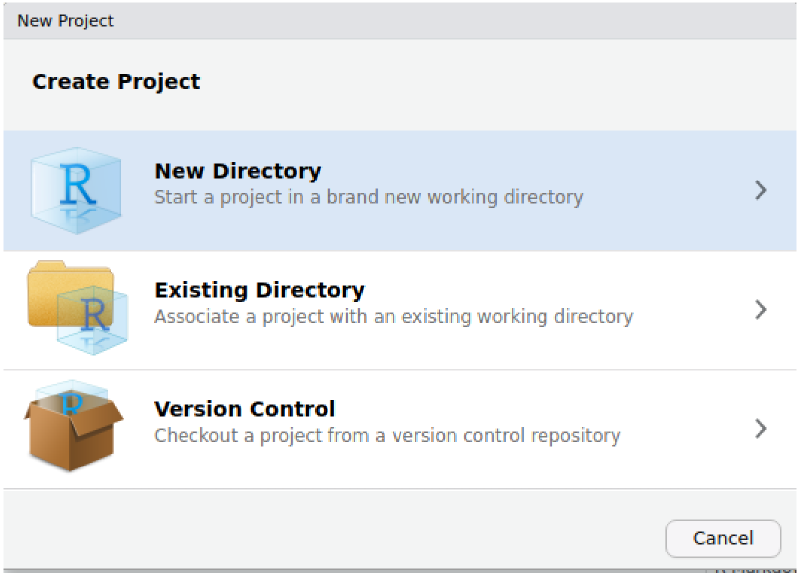
1. **EITHER:**
    - Option A: Install R and Rstudio
    - Option B: Register for RstudioCloud (free) and log in
2. Start a new project for the course
3. Change a few Rstudio settings (recommended)
4. Install some R packages (the "tidyverse")
5. Create a new Rmarkdown document
6. Complete today's tasks and exercises
7. Compile your Rmarkdown document
8. Celebrate!

1.1 Introducing R and Rstudio

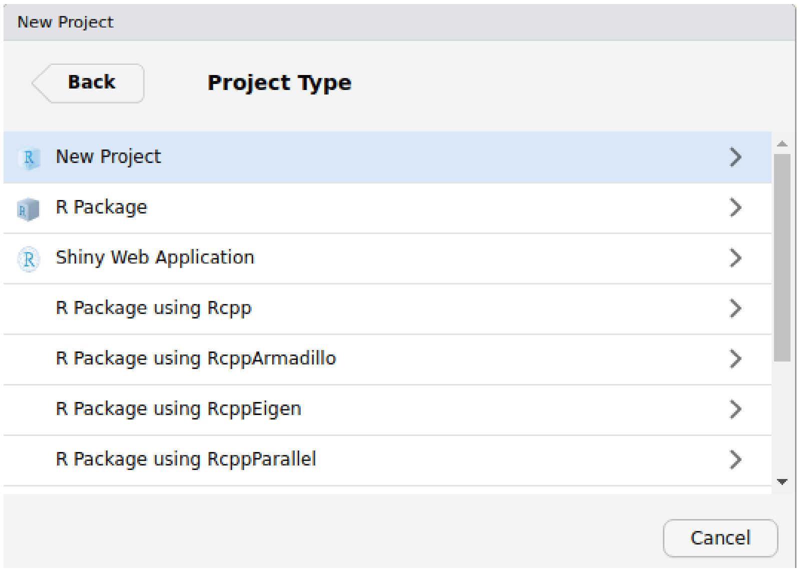
Installing R and Rstudio

Option A: Installing R and Rstudio (recommended)

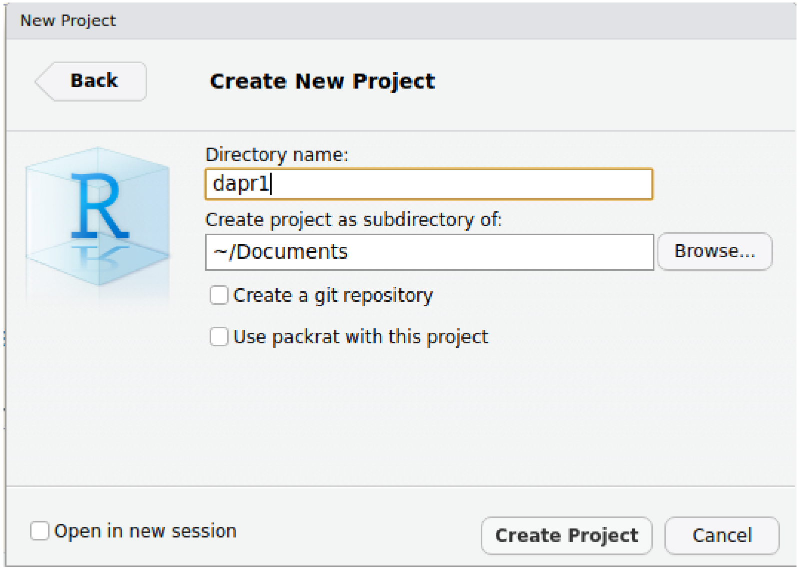
1. Download and install the most recent version of R:
    - If you are on a Mac: https://cran.r-project.org/bin/macosx/
    - If you are on Windows: https://cran.r-project.org/bin/windows/base/
2. Download and install Rstudio:
    - Choose the appropriate downloaded for your computer (e.g.., MacOS/Windows): https://www.rstudio.com/products/rstudio/download/#download
3. Open Rstudio:
4. Create a new project:
    - File > New Project..
    - Click New Directory:


New Project
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

    - Click New Project:



- Name the project, and decide where you want to save it on your computer by clicking on browse. Then click Create Project:



Directory name: dapr1

Create project as subdirectory of: ~/Documents

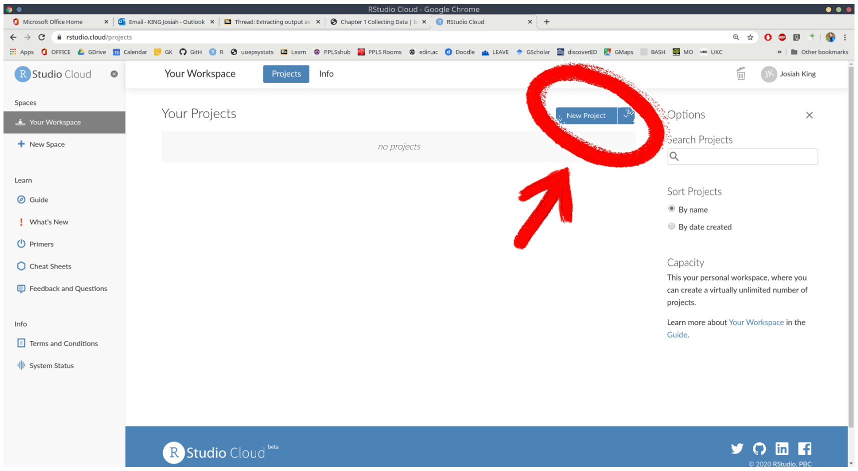
☐ Create a git repository

☐ Use packrat with this project

☐ Open in new session

Option B: Rstudio Cloud (for chromebook users)

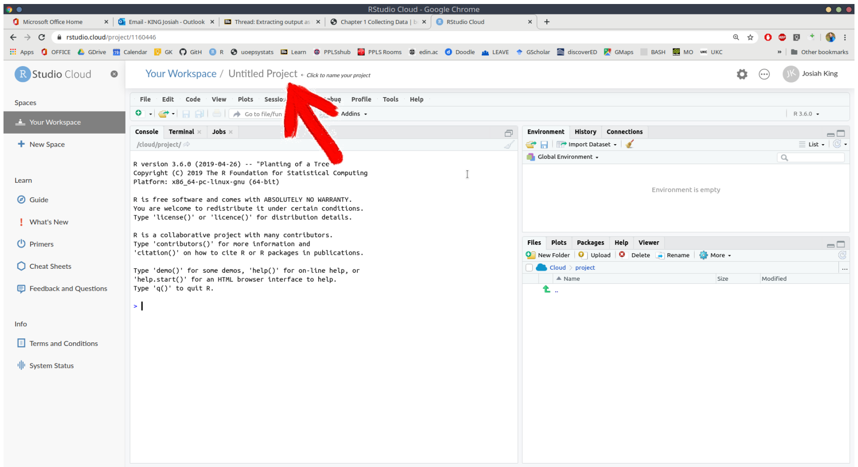
1. Register for Rstudio Cloud (https://rstudio.cloud/).
2. Log in to Rstudio Cloud



This your personal workspace, where you can create a virtually unlimited number of projects.

Learn more about Your Workspace in the Guide.

3. Create a new project

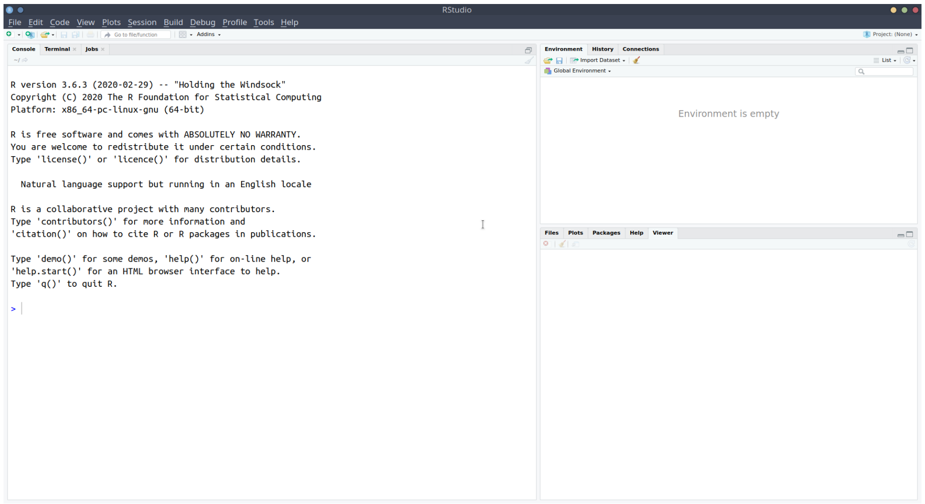


```
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

4. Rename the project

First look at Rstudio

Okay, now you should have a project open, and you should see something which looks more or less like the below, where there are several little windows.



```
R version 3.6.3 (2020-02-29) -- "Holding the Windsock"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Environment is empty

We're going to explore what each of these little windows offer by just diving in and starting to do things.

R as a calculator

Starting in the left-hand window, you'll notice the little blue arrow >. This is where we R code gets *executed*. Type 2+2, and hit enter

It's a calculator!

Let's work through some of the basic operations (adding, subtracting etc).
Try these commands yourself:

- `2+5`
- `10-4`
- `2*5`
- `10-(2*5)`
- `(10-2)*5`
- `10/2`
- `3^2` (Hint, interpret the ^ symbol as "to the power of")

A little note

Whenever you see the blue arrow (>), it means R is ready and waiting for a command.

If you type `10+` and press enter, you'll see that instead of > you are left with +. This means that R is waiting for more. Either give it more, or cancel the command by pressing the escape key on your keyboard.

Now let's take a sidestep.
As well as performing calculations, we can *ask* R things, such as "Is 3 less than

5?":

```
3<5
```

```
[1] TRUE
```

Try the following:

- `3>5` - "is 3 greater than 5?"
- `3<=5` - "is 3 less than OR equal to 5?"
- `3>=3` - "is 3 greater than OR equal to 3?"
- `3==5` - "is 3 equal to 5?"
- `(2*5)==10` "is 2 times 5 equal to 10?"
- `(2*5)!=11` "is 2 times 5 NOT equal to 11?"

R as a calculator with a memory

We can also store things in R's memory, and to that we just need to give them a name.
Type `x <- 5` and press enter.

What has happened? We've just stored something named `x` which has the value `5`. We can now refer to the name and it will give us the value! Try typing `x` and hitting enter. It should give you the number 5.
What about `x*3`?

Storing things in R

The `<-` symbol is used to *assign* a value to a named object.

[name] <- *[value]*

Note, there are a few rules about names in R:

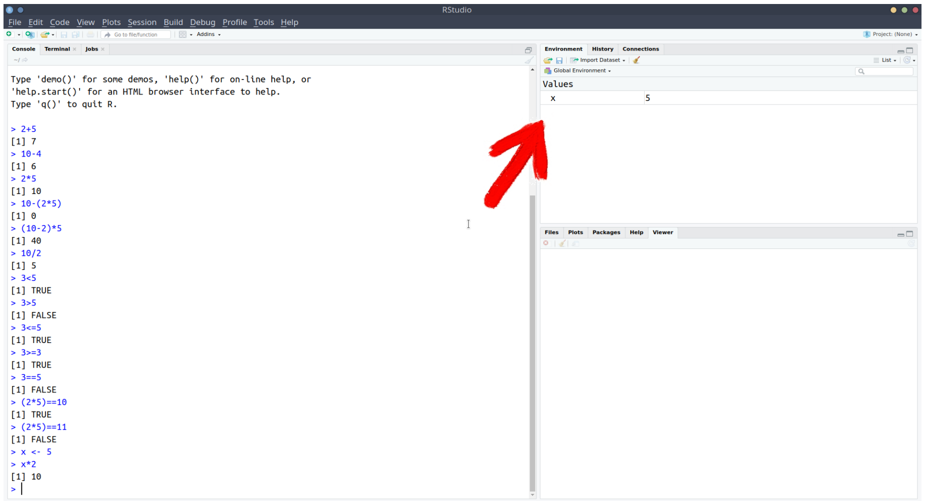
- No spaces - spaces *inside* a name are not allowed (the spaces around the `<-` don't matter):
  - `lucky_number <- 5`   `lucky number <- 5`

- Names must start with a letter:
  - `lucky_number <- 5`   `1lucky_number <- 5`

- Case sensitive:
  - `lucky_number` is different from `Lucky_Number`

- Reserved words - there is a set of words you can't use as names, including: if, else, for, in, TRUE, FALSE, NULL, NA, NaN, function
  (Don't worry about remembering these, R will tell you if you make the mistake of trying to name a variable after one of these).

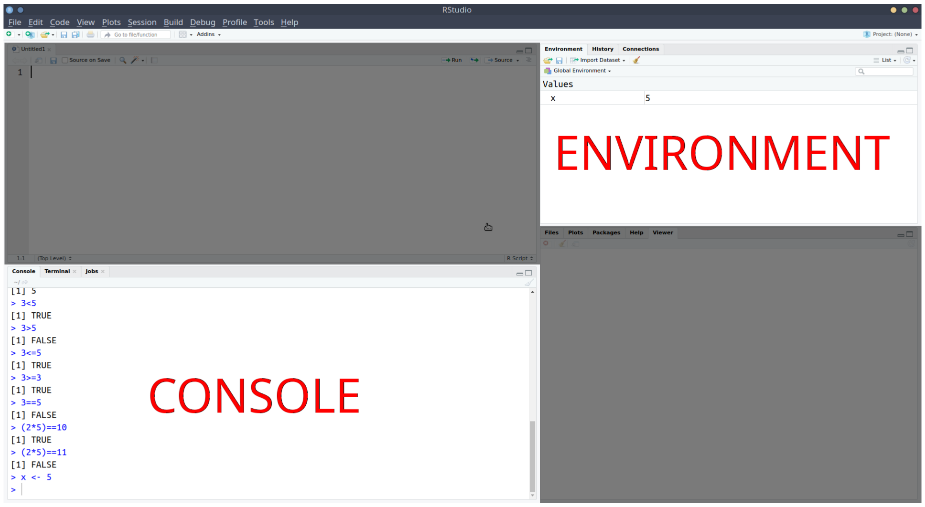
You might have noticed that something else happened when you executed the code `x<-5`. The thing we named **x** with a value of **5** suddenly appeared in the top-right window. This is known as the **environment**, and it shows everything that we store things in R:



```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 2+5
[1] 7
> 10-4
[1] 6
> 2*5
[1] 10
> 10-(2*5)
[1] 0
> (10-2)*5
[1] 40
> 10/2
[1] 5
> 3<5
[1] TRUE
> 3>5
[1] FALSE
> 3<=5
[1] TRUE
> 3>=3
[1] TRUE
> 3==5
[1] FALSE
> (2*5)==10
[1] TRUE
> (2*5)==11
[1] FALSE
> x <- 5
> x*2
[1] 10
>
```

```
Values
  x              5
```

We've now used a couple of the windows - we've been executing R code in the **console**, and learned about how we can store things in R's memory (the **environment**) by assigning a name to them:



```
Values
  x              5
```

```
[1] 5
> 3<5
[1] TRUE
> 3>5
[1] FALSE
> 3<=5
[1] TRUE
> 3>=3
[1] TRUE
> 3==5
[1] FALSE
> (2*5)==10
[1] TRUE
> (2*5)==11
[1] FALSE
> x <- 5
>
```

Notice that in the screenshot above, we have moved the **console** down to the bottom-left, and introduced a new window above it. This is the one that we're going to talk about next.

Rscripts and Rmarkdown

What if we want to edit our code?
Whatever we write in the console just disappears upwards. What if we want to change things we did earlier on?

Well, we can write and edit our code in a separate place *before* sending it to the **console** to be executed!!

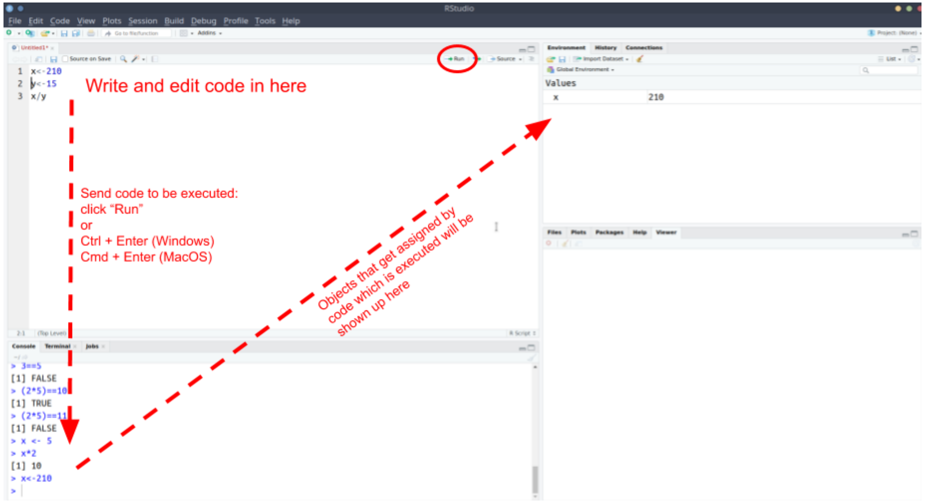
R scripts

Task 1. Open an R script + **File > New File > R script** 1. Copy and paste the following into the R script

```
x<-210
y<-15
x/y
```

1. With your text-cursor (blinking vertical line) on the top line:
    - Ctrl + enter (Windows)
    - Cmd + enter (MacOS)

Notice what has happened - it has sent the command `x<-210` to the console, where it has been executed, and **x** is now in your environment.
Additionally, it has moved the text-cursor to the next line.



```
x<-210
y<-15
x/y
```

```
> 3==5
[1] FALSE
> (2*5)==10
[1] TRUE
> (2*5)==11
[1] FALSE
> x <- 5
> x*2
[1] 10
> x<-210
>
```

```
Values
x                210
```

File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1*
Source on Save
Run
Source
Environment History Connections
Import Dataset
Global Environment
List
Files Plots Packages Help Viewer
Console Terminal Jobs
Write and edit code in here
Send code to be executed:
click "Run"
or
Ctrl + Enter (Windows)
Cmd + Enter (MacOS)
Objects that get assigned by code which is executed will be shown up here

Task Press Ctrl + enter (Windows) or Cmd + enter (MacOS) again.
Do it twice (this will run the next two lines).

Then, change **x** to some other number in your R script, and run the lines again (starting at the top).

Task Add the following line to your Rscript and execute it (send it to the console pressing Ctrl/Cmd + Enter):

```r
plot(1,5)
```

A very basic plot should have appeared in the bottom-right of Rstudio. The bottom-right window actually does some other useful things.

Task 1. Save the Rscript you have been working with: + File > Save + give it an appropriate name, and click save. 1. Check that you can now see that file in the project, by clicking on the "Files" tab of the bottom-right window.

Rmarkdown

In addition to R scripts, there is another type of document we can create, known as an "Rmarkdown".

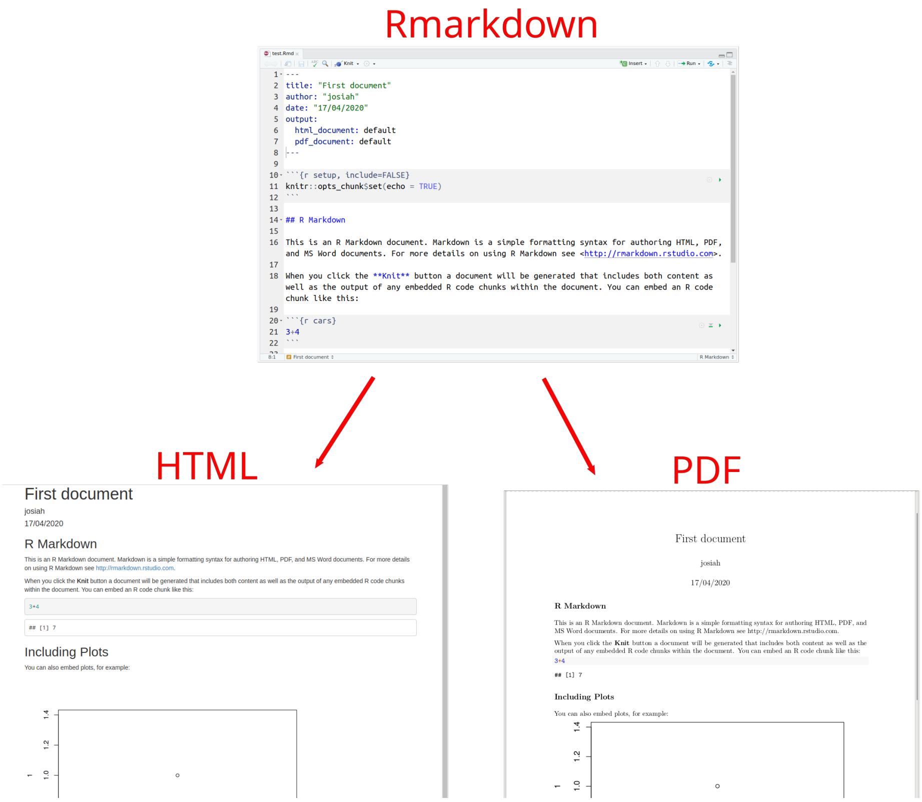
Rmarkdown documents combine the analytical power of R and the utility of a text-processor.

We can have one document which contains all of our analysis as well as our written text, and can be *compiled* into a nicely formatted report.

This saves us doing analysis in R and copying results across to Microsoft Word. It ensures our report accurately reflects our analysis.

Everything that you're reading now has all been written in Rmarkdown!

We're going to use Rmarkdown documents throughout this course. We'll get into it how to write them lower down, but Rmarkdown documents are basically written text interspersed with "code-chunks". In the example below, you can see the grey boxes indicating the R code, with text in between.
We can then compile the document into either a **.pdf** or a **.html**.



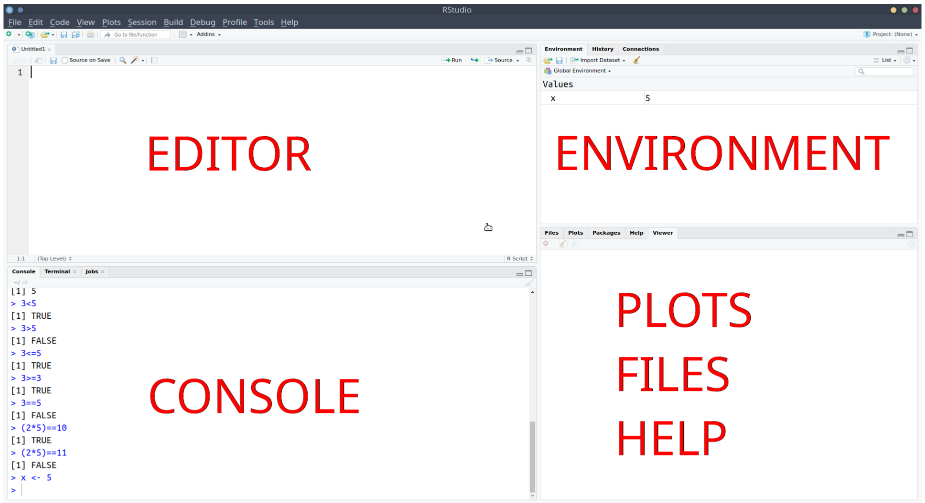
~~~
---
title: "First document"
author: "josiah"
date: "17/04/2020"
output:
  html_document: default
  pdf_document: default
---

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

R Markdown

This is an R Markdown document. 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. You can embed an R code
chunk like this:

```{r cars}
3+4
```
```

First document

josiah

17/04/2020

R Markdown

This is an R Markdown document. 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. You can embed an R code chunk like this:

```
3+4
```

```
## [1] 7
```

Including Plots

You can also embed plots, for example:

First document

josiah

17/04/2020

R Markdown

This is an R Markdown document. 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. You can embed an R code chunk like this:

```
3+4
```

```
## [1] 7
```

Including Plots

You can also embed plots, for example:

Recap

Okay, so we've now seen all of the different windows in Rstudio in action:

- The **console** is where R code gets executed
- The **environment** is R's memory, you can *assign* something a name and store it here, and then refer to it by name in your code.
- The **editor** is where you can write and edit R code and Rmarkdown documents. You can then send this to the console for it to be executed.
- The bottom-right window shows you the **plots** that you create, the **files** in your project, and some other things (we'll get to these later).
```



Values

```
x   5
```

```
[1] 5
> 3<5
[1] TRUE
> 3>5
[1] FALSE
> 3<=5
[1] TRUE
> 3>=3
[1] TRUE
> 3==5
[1] FALSE
> (2*5)==10
[1] TRUE
> (2*5)==11
[1] FALSE
> x <- 5
>
```

1.2 Take a breather

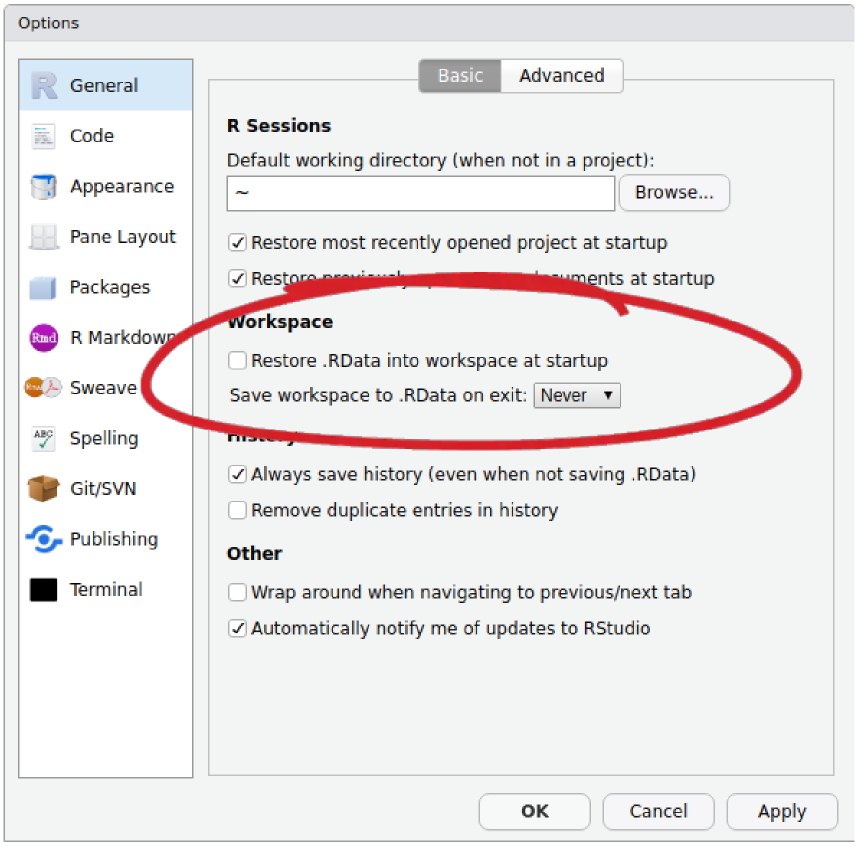
Below are a couple of our recommended settings for you to change as you begin your journey in R. After you've changed them, take a 5 minute break before moving on to learning about how we store data in R.

Useful Settings 1: Clean environments

As you use R more, you will store lots of things with different names. Throughout this course alone, you'll probably name hundreds of different things.
This could quickly get messy within our project.

We can make it so that we have a clean environment each time you open Rstudio. This will be really handy.

1. In the top menu, click
   Tools > Global Options...
2. Then, *untick* the box for "Restore .RData into workspace at startup", and change "Save workspace to .RData on exit" to "Never":



R Sessions

Default working directory (when not in a project):

~

☑ Restore most recently opened project at startup

☑ Restore [illegible] uments at startup

Workspace

☐ Restore .RData into workspace at startup

Save workspace to .RData on exit: Never ▾

☑ Always save history (even when not saving .RData)

☐ Remove duplicate entries in history

Other

☐ Wrap around when navigating to previous/next tab

☑ Automatically notify me of updates to RStudio

Useful Settings 2: Wrapping code

In the editor, you might end up with a line of code which is really long:

```
x <- 1+2+3+6+3+45+8467+356+8565+34+34+657+6756+456+456+54+3+78+3+3476+8+4+67+456+567+3
```

You can make Rstudio ‘wrap’ the line, so that you can see it all, without having to scroll.

```
x <- 1+2+3+6+3+45+8467+356+8565+34+34+657+6756+456+456+54+3+78+3+3476+8+4+67+
456+567+3+34575+45+2+6+9+5+6
```

1. In the top menu, click
   Tools > Global Options...
2. In the left menu of the box, click “Code”
3. *Tick* the box for “Soft-wrap R source files”

1.3 Data in R

Installing R packages

Alongside the basic installation of R and Rstudio, there are many add-on packages which the R community create and maintain.

The thousands of packages are part of what makes R such a powerful and useful tool - there is a package for almost everything you could want to do in R.

In order to be able to write and compile Rmarkdown documents (and do a whole load of other things which we are going to need throughout the course) we are now going to install a set of packages known collectively as the "tidyverse".

Task In the **console**, type `install.packages("tidyverse")` and hit enter.

Lots of red text will come up, and it will take a bit of time.

When it has finished, and R is ready for you to use again, you will see the little blue arrow >.

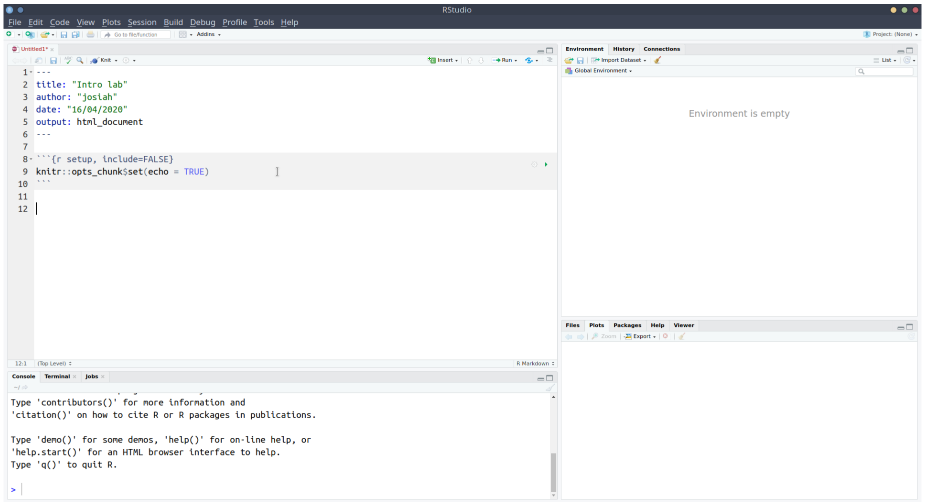
Starting a new .Rmd document

Task Open a new Rmarkdown document.

File > New File > R Markdown..

When the box pops-up, give a title of your choice ("Intro lab", maybe?) and your name as the author.

The file which opens will have some template stuff in it. Delete everything below the first code chunk to start with a fresh document:



~~~
---
title: "Intro lab"
author: "josiah"
date: "16/04/2020"
output: html_document
---

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

~~~

```
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Task Insert a new code chunk by either using the Insert button in the top right of the document, and clicking R, or typing *Ctrl + Alt + i*

Inside the chunk, type:
`print("Hello world! My name is ?")`.

To execute the code inside the chunk, you can either: + do as you did in the R script - put the text-cursor on the first line, and hit Ctrl/Cmd + Enter to run the lines sequentially + click the little green arrow at the top right of your code-chunk to run all of the code inside the chunk.

You can see that the output gets printed below.

Using R packages

We're going to use some functions which are in the **tidyverse** package, which already installed above.
However, it's not enough just to install it - to actually *use* the package, we need to load it using `library(tidyverse)`.

When writing analysis code, we want it to be **reproducible** - we want to be able to give somebody else our code and the data, and ensure that they can get the same results. To do this, we need to show what packages we use.
It is good practice to load any packages you use at the top of your code.

Task In your first code chunk, type:

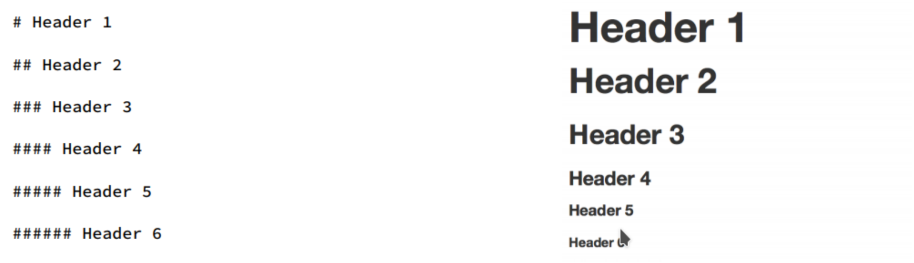
```r
#I'm going to use these packages in this document:
library(tidyverse)
```

and run the chunk.
Note, you might get various messages popping up below when you run this chunk, that is fine).
:::yellow
Comments in code
Note that using `#` in R code makes that line a comment, which basically means that R will ignore the line. Comments are useful for you to remind yourself of what your code is doing. :::

Task Below the code chunk, add a new line with the following:

# R code examples

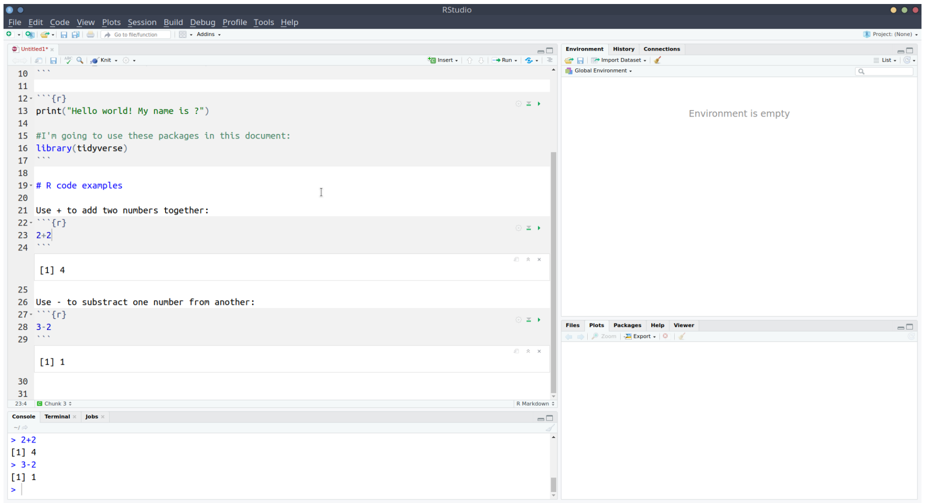
Note that when the # is used in a Rmarkdown file **outside** of a code-chunk, it will make that line a heading when we finally get to *compiling* the document. Below, what you see on the left will be compiled to look like those on the right:



```
# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6
```

Task In your Rmarkdown document, choose a few of the symbols below, and write an explanation of what it does, giving an example in a code chunk. You can see an example of the first few below.

- `+`
- `-`
- `*`
- `/`
- `()`
- `^`
- `<-`
- `<`
- `>`
- `<=`
- `>=`
- `==`
- `!=`



```
```

```{r}
print("Hello world! My name is ?")

#I'm going to use these packages in this document:
library(tidyverse)
```

R code examples

Use + to add two numbers together:
```{r}
2+2
```

 [1] 4

Use - to substract one number from another:
```{r}
3-2
```

 [1] 1
```

```
> 2+2
[1] 4
> 3-2
[1] 1
>
```
```

## Storing data in R: Sequences ("Vectors") of values

We've already seen how to assign a value to a name/symbol using `<-`. However, we've only seen how to assign a single number, e.g, `x<-5`.

To assign a sequence of values to R, we combine the values using `c()`.

```
myfirstvector <- c(1,5,3,7)
myfirstvector
```

```
[1] 1 5 3 7
```

```
myfirstvector + 5
```

```
[1]  6 10  8 12
```

Values don't have to be numbers, but note what happens when we try to add 5 to a sequence which includes some non-numbers:

```
mysecondvector <- c(1,4,"cat","dog","parrot","peppapig")
mysecondvector + 5
```

```
Error in mysecondvector + 5 : non-numeric argument to binary operator
```

Reading in data

While we can manually input data like we did above, more often, we will need to read in data which has been created elsewhere (like in excel, or by some software which is used to present participants with experiments).

Task Add a new heading by typing the following:

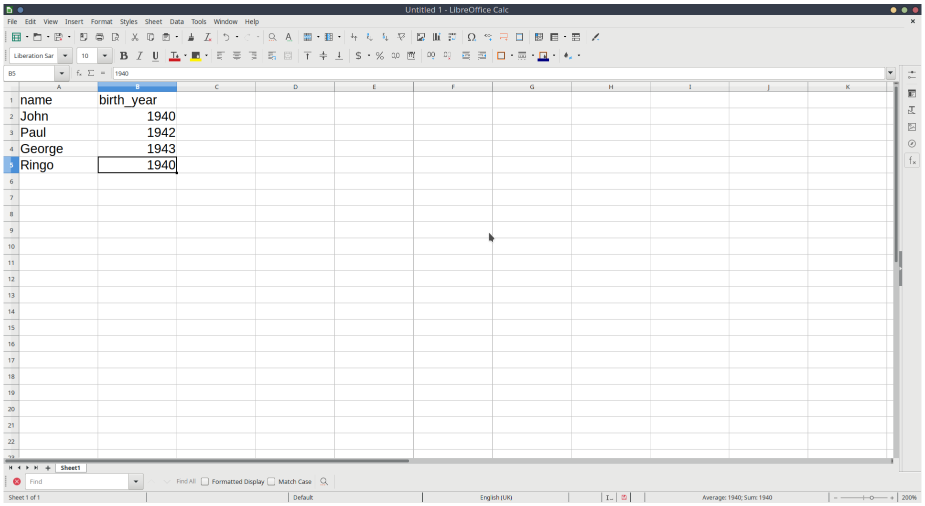
# Reading and storing data

Remember: We make headings using the # *outside* of a code chunk.

Task Open Microsoft Excel, or LibreOffice Calc, or whatever spreadsheet software you have available to you, and create some data with more than one variable.

It can be whatever you want, but we've used a very small example here for you to follow, so feel free to use it if you like.

We've got two sets of values here: the names and the birth-years of each member of the beatles. The easiest way to think of this would be to have a row for each Beatle, and a column for each of name and birth-year.



name	birth_year
John	1940
Paul	1942
George	1943
Ringo	1940

Task Save the data as a **.csv** file.

Although R can read data when it's saved in Microsoft/LibreOffice formats, the simplest, and most universal way to save data is as simple text, with the values separated by some character - **.csv** stands for **comma separated values**.

In Microsoft Excel, if you go to: **File > Save as**
In the Save as Type box, choose to save the file as **CSV (Comma delimited)**.

Important: save your data in the project folder you created at the start of this lab.

Back in Rstudio…

Next, we're going to read the data into R. We can do this by using the `read_csv()` function, and directing it to the file you just saved.

Task Create a new code-chunk in your Rmarkdown, and in the chunk, type: `read_csv("name-of-your-data.csv")`, where you replace *name-of-your-data* with whatever you just saved your data as in your spreadsheet software.

Helpful tip

If you have your text-cursor inside the quotation marks, and press the tab key on your keyboard, it will show you the files inside your project. You can then use the arrow keys to choose between them and press enter to add the code:

TODO - screencap gif

When you run the line of code you just wrote, it will print out the data, but will not store it. To do that, we need to assign it as something:

```
beatles <- read_csv("data_from_excel.csv")
```

Note that this will now turn up in the *Environment* pane of Rstudio.

Now that we've got our data in R, we can print it out by simply using its name:

```
beatles
```

```
# A tibble: 4 x 2
  name   birth_year
  <chr>       <dbl>
1 John         1940
2 Paul         1942
3 George       1943
4 Ringo        1940
```

And we can do things such as ask R how many rows and columns there are:

```
dim(beatles)
```

```
[1] 4 2
```

```
str(beatles)
```

```
Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame':    4 obs. of  2 variables:
 $ name      : chr  "John" "Paul" "George" "Ringo"
 $ birth_year: num  1940 1942 1943 1940
 - attr(*, "spec")=
  .. cols(
  ..   name = col_character(),
  ..   birth_year = col_double()
  .. )
```

Task Use `dim()` to confirm how many rows and columns are in your data.

Use `str()` to take a look at the structure of the data.
Don't worry about the output of `str()` right now, we'll pick up with this in the next lab.

Task By now, you should have an Rmardkown document (**.Rmd**) with your answers to the tasks we've been through today.

Compile the document by clicking on the **Knit** button at the top. The little arrow to the right allows you to compile to either **.pdf** or **.html**.

Checklist for today

1. **EITHER:**
   - Option A: Install R and Rstudio
   - Option B: Register for RstudioCloud (free) and log in
2. Start a new project for the course
3. Change a few Rstudio settings (recommended)
4. Install some R packages (the "tidyverse")
5. Create a new Rmarkdown document

6. Complete today's tasks and exercises
7. Compile your Rmarkdown document
8. Celebrate!

Glossary

- console
- environment
- editor
- r script
- rmarkdown

Symbol	Description	Example
`+`	Adds two numbers together	`2+2` - two plus two
`-`	Subtract one number from another	`3-1` - three minus one
`*`	Multiply two numbers together	`3*3` - three times three
`/`	Divide one number by another	`9/3` - nine divided by three
`()`	group operations together	`(2+2)/4` is different from `2+2/4`
`^`	to the power of..	`4^2` - four to the power of two, or four squared
`<-`	stores an object in R with the left hand side (LHS) as the name, and the RHS as the value	`x<-10`
`=`	stores an object in R with the left hand side (LHS) as the name, and the RHS as the value	`x = 10`
`<`	is less than?	`2<3`
`>`	is greater than?	`2>3`
`<=`	is less than or equal to?	`2<=3`
`>=`	is greater than or equal to?	`2>=2`
`==`	is equal to?	`(5+5) == 10`
`!=`	is not equal to?	`(2+3) != 4`
`c()`	combines values into a vector (a sequence of values)	`c(1,2,3,4)`

Chapter 2

Types of data

Learning Objectives

- LO1: Learn how to access variables in a dataframe in R
- LO2: Understand the distinction between types of variables
- LO3: Learn how to code data as different types (classes) in R

indexing, accessors

factor

Task In a new code chunk, do the following:

1. store the following numbers as an object in R:
   4,7,3,1,8,9,5,2,2,6,9,9,5,20
2. Try using the function `sum()`, with the name of your object inside the brackets.

Solution Because this is just a set of numbers, we store it as a **vector**, using `c()`.
We have named it `myvec`, but you can call yours whatever you like.

The `sum()` function will add all of the numbers together!

```
myvec <- c(4,7,3,1,8,9,5,2,2,6,9,9,5,20)
sum(myvec)
```

```
[1] 90
```

Task Using the square brackets - `[]` - pull out the 2nd, 4th and 6th values in the object you just created.

Hint: You will need to put inside the square brackets a *sequence of* numbers. How do we combine numbers in to a sequence in R? using `c()`!

Solution

```
myvec[c(2,4,6)]
```

```
[1] 7 1 9
```

Task Using the square brackets, show the 167th row, with all columns.

Remember: When you are using `[]` with a dataframe, you specify `data[rows, columns]`. If you leave either rows or columns blank it will give all of them - for instance, `data[ , columns]` will give you all rows for some specified columns.

Solution

```
mydata[167,]
```

Matrices

We will often have several vectors in which the 1st value of each vector corresponds to the same observation, for instance:

```
age<-c(1,2,3,4,5)
height<-c(20,70,80,90,95)
```

But these would be better stored as a "Matrix"

```
matrix(c(1,2,3,4,5,20,70,80,90,95), nrow=5)
```

```
     [,1] [,2]
[1,]    1   20
[2,]    2   70
[3,]    3   80
[4,]    4   90
[5,]    5   95
```

Symbol	Description	Example
`[]`	used to extract the 1st, 2nd, … i^{th} elements in a set of numbers	`myvector[3]`
`$`	used to extract a named column from a dataframe	`mydata$age_variable`

Not | `!` | !(1==1) | FALSE Or | `|` | (1==1) | (1==2) | TRUE And | `&` | (1==1) & (1==2) | FALSE

Chapter 3

Visualising distributions

Learning Objectives

- LO1:
- LO2:
- LO3:

Chapter 4

Describing distributions

Learning Objectives

- LO1:
- LO2:
- LO3:

Chapter 5

Visualising and describing relationships

Learning Objectives

- LO1:
- LO2:
- LO3:

Chapter 6

Basics of probability theory

Learning Objectives

- LO1:
- LO2:
- LO3:

thus far - we've talked about summarising a visualising datasets - "descriptive statistics".

this is only a small part of statistics, and we will now start to look at a much more powerful tool - how we can make statistical inferences.

example - sample estimate, assumes representative would we be surprised to discover true param is different? how different?

all based on probability theory

6.1 probability

p(heads)=0.5

toss coin 6 times. how many am i likely to get heads on?

probability is a known model of the world. we can use it to make predictions about events/data

6.2 statistics

in statistics we have only events/data, and we are interested in learning the truth about the world.

Chapter 7

Probability rules

Learning Objectives

- LO1:
- LO2:
- LO3:

Chapter 8

Random variables

Learning Objectives

- LO1:
- LO2:
- LO3:

Chapter 9

Sampling variability and sampling distributions

Learning Objectives

- LO1:
- LO2:
- LO3:

Chapter 10

Bias-Variance Trade-off

Learning Objectives

- LO1:
- LO2:
- LO3: