Oxford Genomics CENTRE

UNIVERSITY OF OXFORD

Introduction to R Programming

26th February 2018

Helen Lockstone and Ben Wright

Bioinformatics Core

wellcome centre human genetics

The Wellcome Trust Centre for Human Genetics

UNIVERSITY OF OXFORD

Introduction to R Programming

- Teaching day and separate workshop session to follow
- Format includes lecture slides to introduce concepts, interactively working through an online tutorial and workshop-style sessions to interact with tutors
- Course material available at:
  http://www.well.ox.ac.uk/bioinformatics/training/Introduction_to_R/
- Aims:
  - Introduce the R software through practical sessions
  - Help make R and associated resources accessible to novice programmers
  - Emphasise good programming practice

Schedule

Time	Topic
10:00 – 10:15	Welcome and introductory remarks
10:15 – 11:00	Getting started with R – RStudio, Data Types and Structures
11:00 – 11:15	*Coffee Break (refreshments provided)*
11:15 – 12:30	Guided tutorial: Handling and Analysing Data in R
12:30 – 13:30	*Lunch break*
13: 30 – 13:45	Overview of loops and functions
13:45 – 15:00	Online tutorial: Handling and Analysing Data in R (continued) and Loops in R
15:00 – 15:15	*Break*
15:15 – 16:00	Optional session to continue working through tutorial

Introductory Remarks

Why learn R?

- The ability to handle and analyse large-scale datasets is, and will continue to be, a key skill in modern biological research, as well as many other fields
- Generating data far outstrips our ability to interpret and make sense of it – and it is still hard to recruit good bioinformaticians
- R has become a key programming language for genomics due to the diverse set of packages available through BioConductor https://www.bioconductor.org/

The downsides to R

- R is not an intuitive language to learn, even for those who are familiar with programming concepts
- It can take many months to start to feel comfortable writing your own R code, so be prepared for some investment of time, hard work and a steep learning curve
- R's extensive functionality and versatility are its main attributes but also the reason it can be hard to know where to begin…

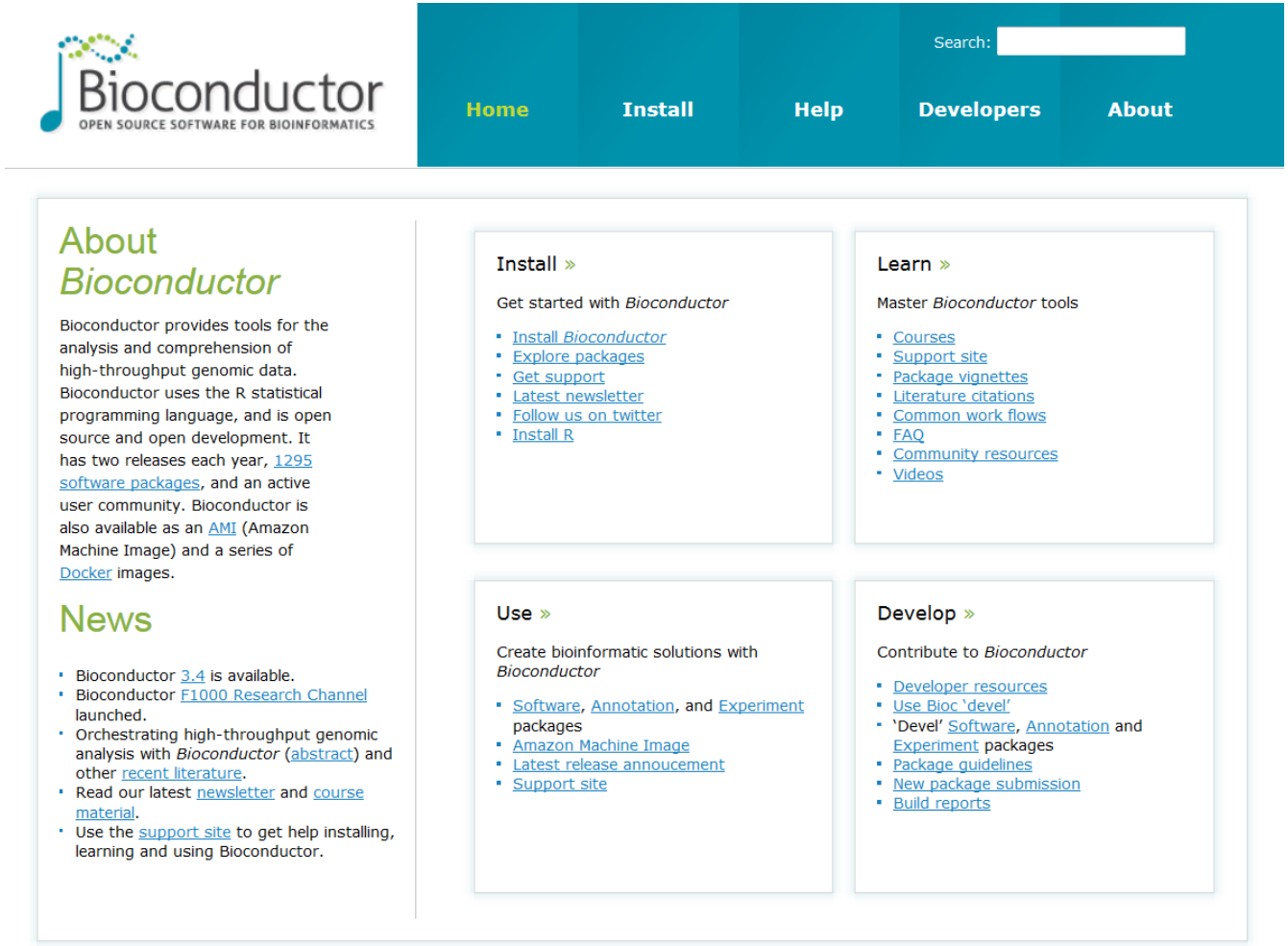
Tips for Successful Programming

- Logical thinking and problem solving skills are vital
  - Decide what steps needed to solve a task
  - Troubleshooting error messages
  - Testing code to ensure it does what it should
- Consistency, accuracy and attention to detail
  - Easy to make unintentional mistakes (which R may well execute with no warning message)
  - Check what code is doing at every step
  - Mentally predict what should happen, so you can spot potential errors
- Accept that it can be a frustrating process!

What is R?

- R is a powerful software package for statistical analysis and also a high-level programming language
- Originally written in the 1990s for teaching statistics by Ross Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland
- It's open-source, available for free for Win, Mac, Linux and regularly updated (maintained by R Core development team)
- http://www.r-project.org/index.html
- Has become a mainstream research tool
  - Users have contributed hundreds of add-on packages (CRAN)
  - Bioconductor resource is invaluable for genomic data https://www.bioconductor.org/

Bioconductor



Bioconductor packages extend functionality of R to all aspects of handling, processing and analysing genomic data

https://www.bioconductor.org/

Getting started with R

- Some aspects of R are analogous to tasks you might routinely perform in Excel:
  - sorting/filtering data
  - finding a particular occurrence of text
  - simple data summary statistics and tests
  - plotting graphs

- R provides far more complex functionality besides, particularly for statistical modelling/analysis and data visualisation. It can also perform small useful tasks with a simple line or two of code (where there may not be a quick way to do the equivalent in Excel) e.g. finding the overlap of two lists of genes

Getting started with R

- For those without a programming background, the biggest challenge might be getting used to how the R language is structured, and working with objects to store and manipulate data
- Imagine that instead of using Excel, we are giving R step-by-step instructions of what we want to do (e.g. filter a column for all values <0.05):
  - Our data could be created in Excel and read into R (like a table); it can be stored in an appropriate object (named by the user)
  - We'd need to specify the column of interest somehow (R has more than one way to do this) and set the condition 'values less than 0.05'
  - Finally we'd have to decide whether to display the output or save it in a new variable
- Although some aspects of R can be quite unintuitive and take some time to become familiar with, they provide the flexibility that makes the software so powerful.

Getting started with R

- R is extremely extensive and versatile, quite possibly no two people use it in the same way. So how best to get started?'

- Hopefully this course will help, and R itself it very well documented with a large and active user community (mailing lists, online forums like stack overflow etc) – usually googling any R problem will find relevant threads.

- Learn by doing – follow examples in the manuals or start by running scripts written by others to understand what the code does before moving on to modifying code or writing your own scripts from scratch.

- It's impossible to remember the details of all R functions (or even know about all of them!) – make use of the help pages to check syntax, arguments, examples of usage etc

Some general advice

- A computer will do exactly what you tell it to do and you need to tell it every little step, in the right order
  - Imagine writing instructions that someone can follow to produce a particular result, including every tiny detail they would need
  - Akin to instructions for an experimental protocol - important to be very precise
- Comment your code as much as possible – if you need to revisit it at a later date, it will make much more sense!

- Check and double check (and check again) that your code is doing what you ***think*** it is doing – R has some default behaviours that may not always be realised, and can lead to problems if not spotted
  - Inspect the objects created, their length, contents and so on.
  - Manually check a few elements of the output to be sure they are correct

Problem solving

- You will inevitably run into problems when you are programming - try solving them by:
  - reading the error message
  - reading the help file
  - make a list of what could possibly be wrong and check these possibilities one by one
  - break down a complex step into smaller, simpler steps (this is also a useful way to build up a more complex piece of code)
  - If you really cannot solve a problem, remember there are usually alternative ways in R to achieve the same goal (using a related function for example)
  - Google the problem – many helpful forums/lists
  - If all else fails, post a question on the relevant help list (but be sure to read the guidance first!)

R documentation and help

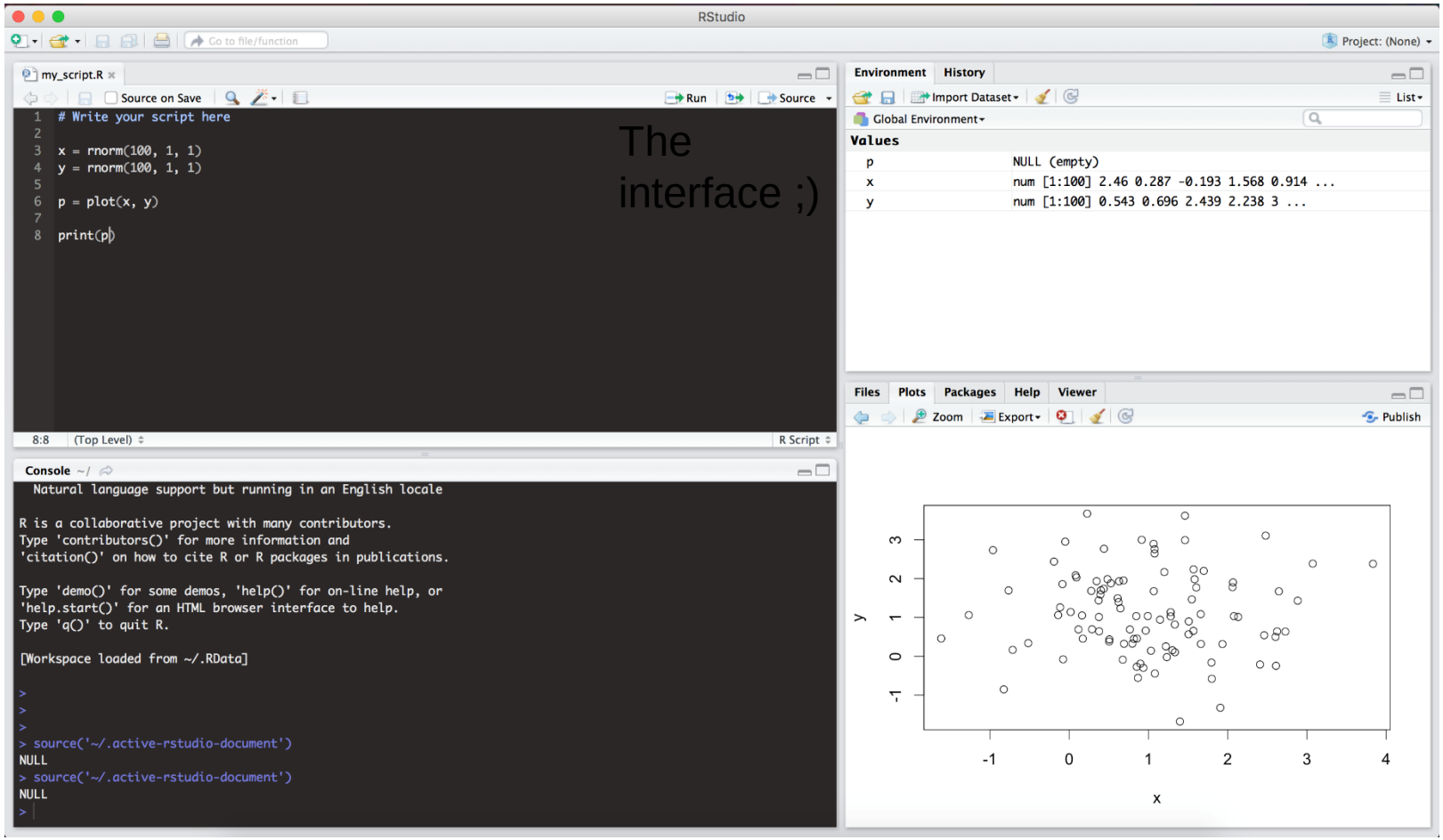
- `> help.start()`
  - Most useful if you don't know which command to use
  - Starts a Search Engine in your favourite browser
  - Can get this also from the R GUI Help Menu
- If you do know which command to use but need to check details
- `> ?plot`
- `or equivalently`
- `> help(plot) # help("plot") in RStudio`
  - Brings up the help page on the function 'plot'
- From the R GUI Help menu or the R website, you can get Manuals (in PDF)
  - An Introduction to R
  - R Reference manual

R documentation and help

- Further reading
  - Michael J Crawley (2005) Statistics: An Introduction using R. Wiley: Chichester, England
    - This is an excellent book, and well worth working through
    - Also has an associated web page with datasets, exercises etc
    - http://www.imperial.ac.uk/bio/research/crawley/statistics/
  - Tutorials and other textbooks – see for example http://www.statmethods.net/about/books.html or http://cran.cnr.berkeley.edu/other-docs.html (also includes R manuals in other languages)
  - Various material on the R web site
  - FAQ
  - R-help mailing list

https://www.rstudio.com

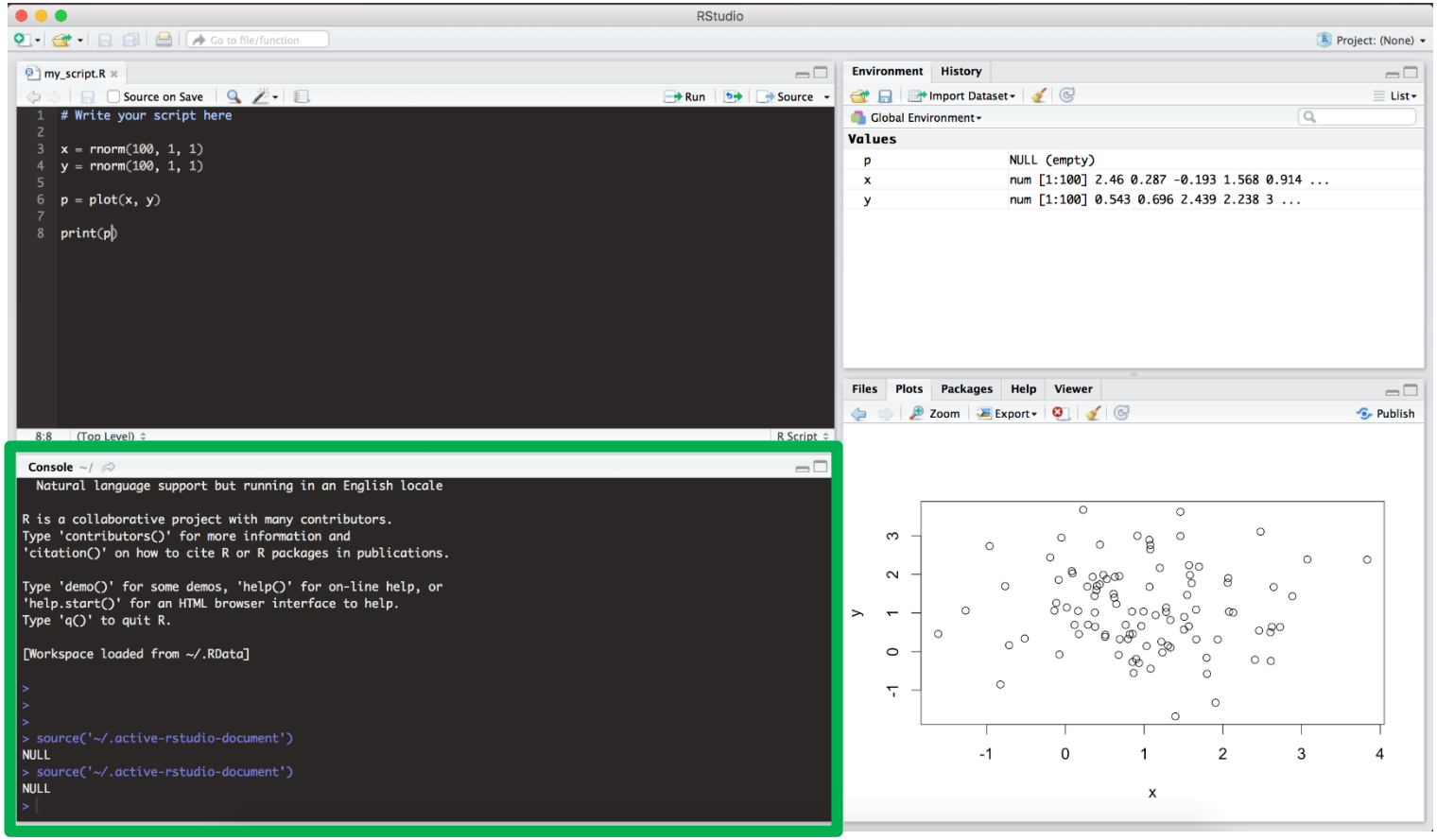
RStudio Interface



An interactive and easy-to-use interface with many features that make working with R easier

Slide courtesy of Quentin Ferry

RStudio Interface

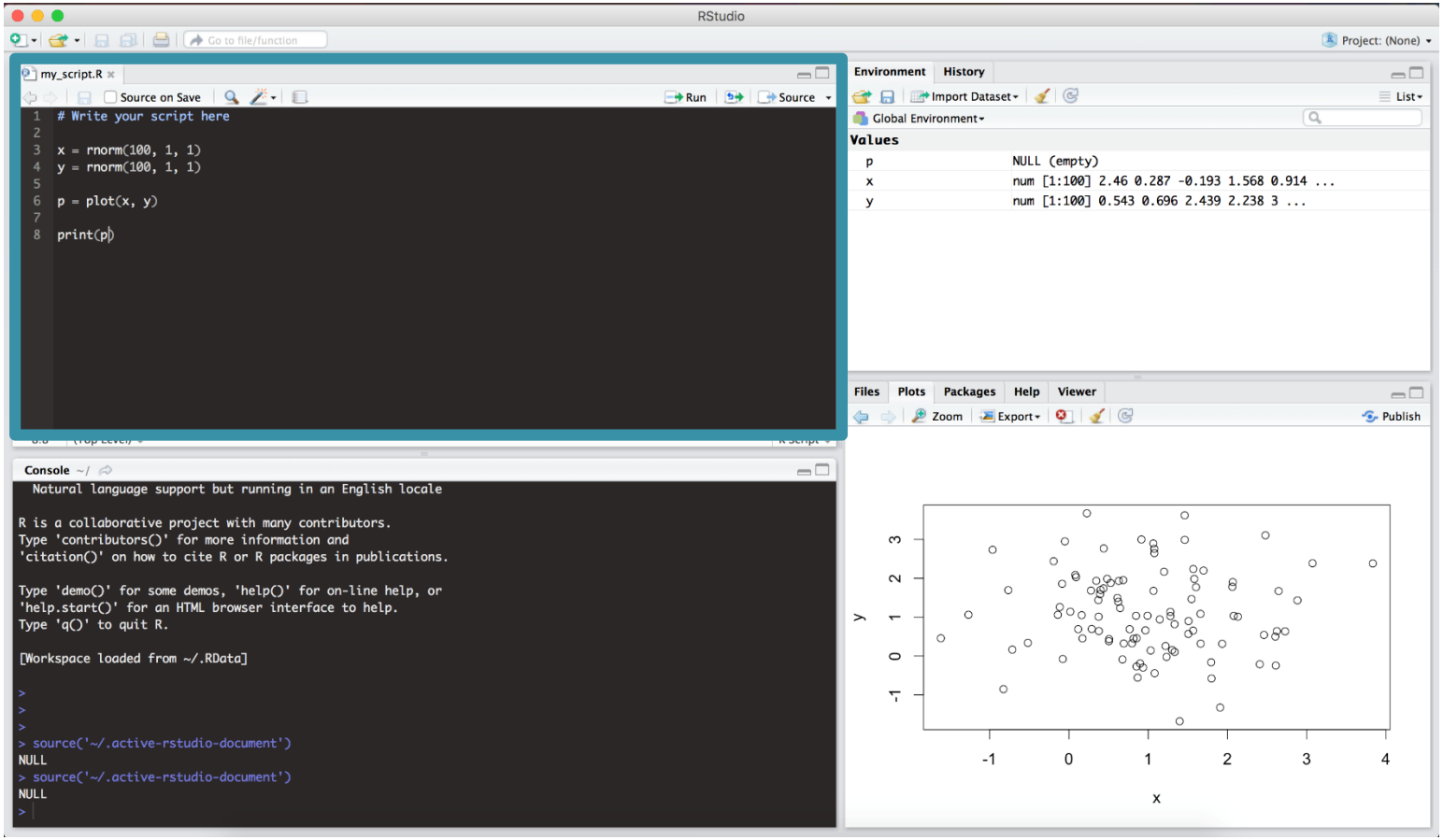


Console for entering R commands

Slide courtesy of Quentin Ferry

RStudio Interface

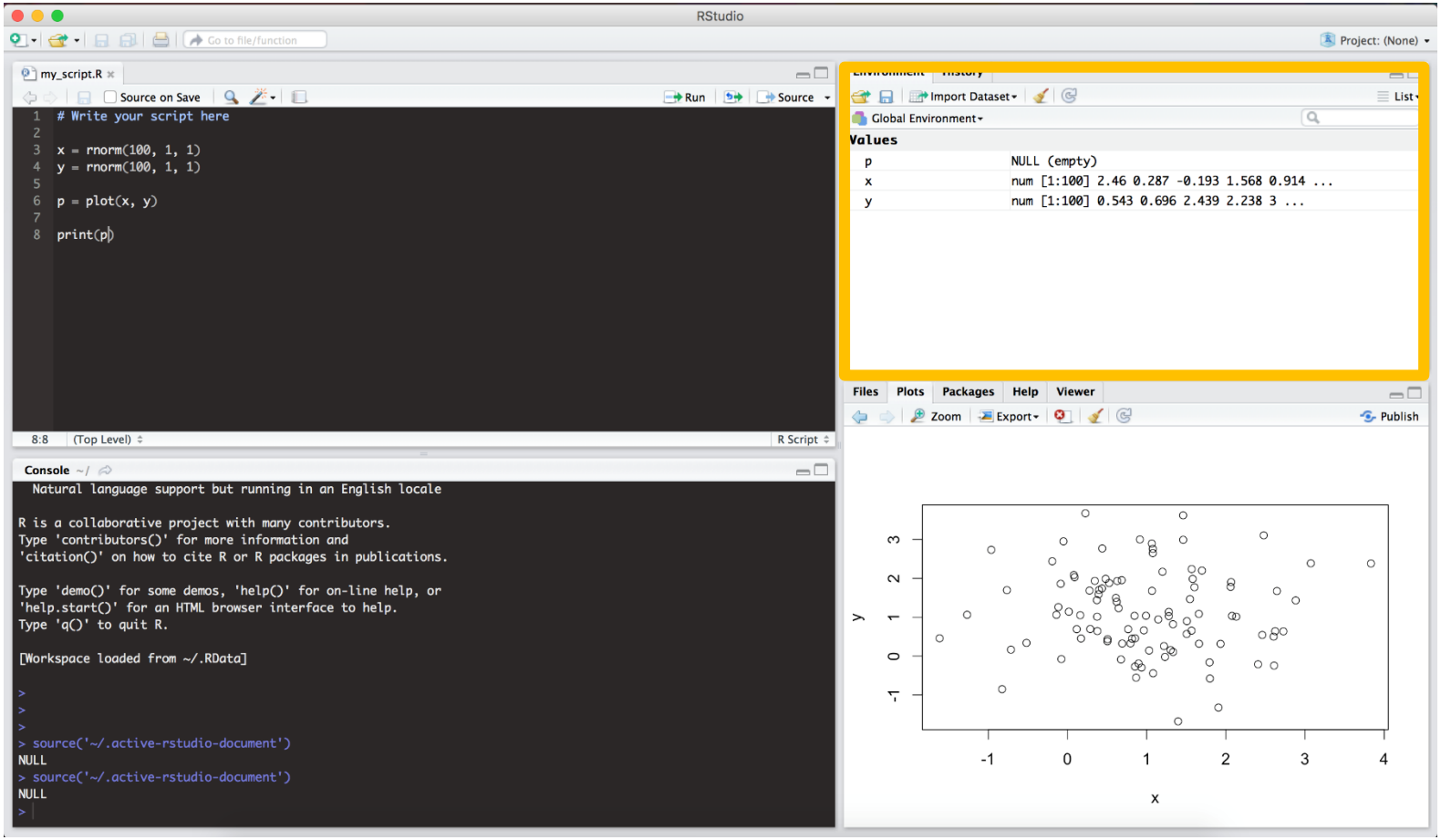
Script, data tables, etc…



Slide courtesy of Quentin Ferry

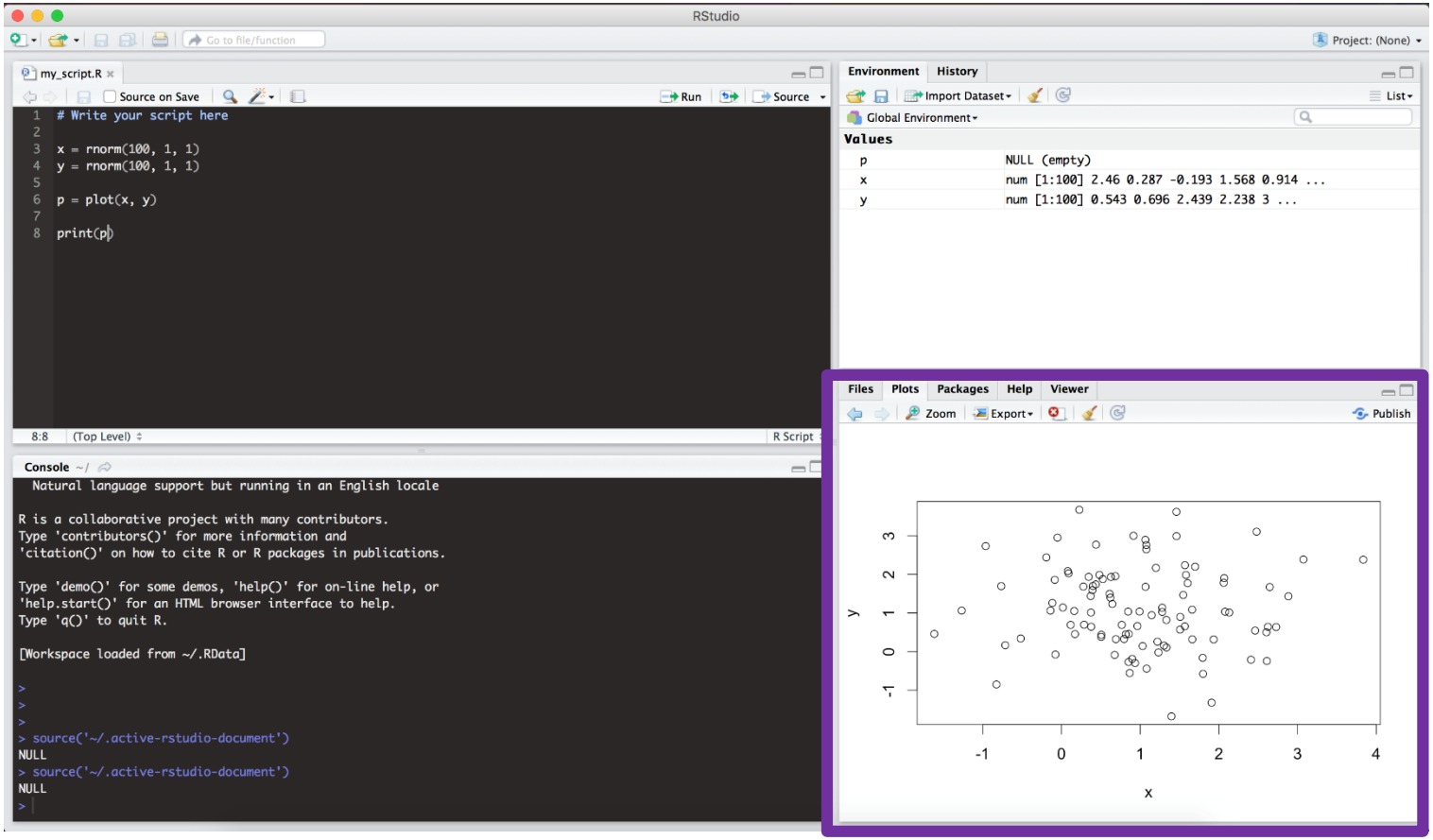
RStudio Interface

Clickable list of variables in memory



Slide courtesy of Quentin Ferry

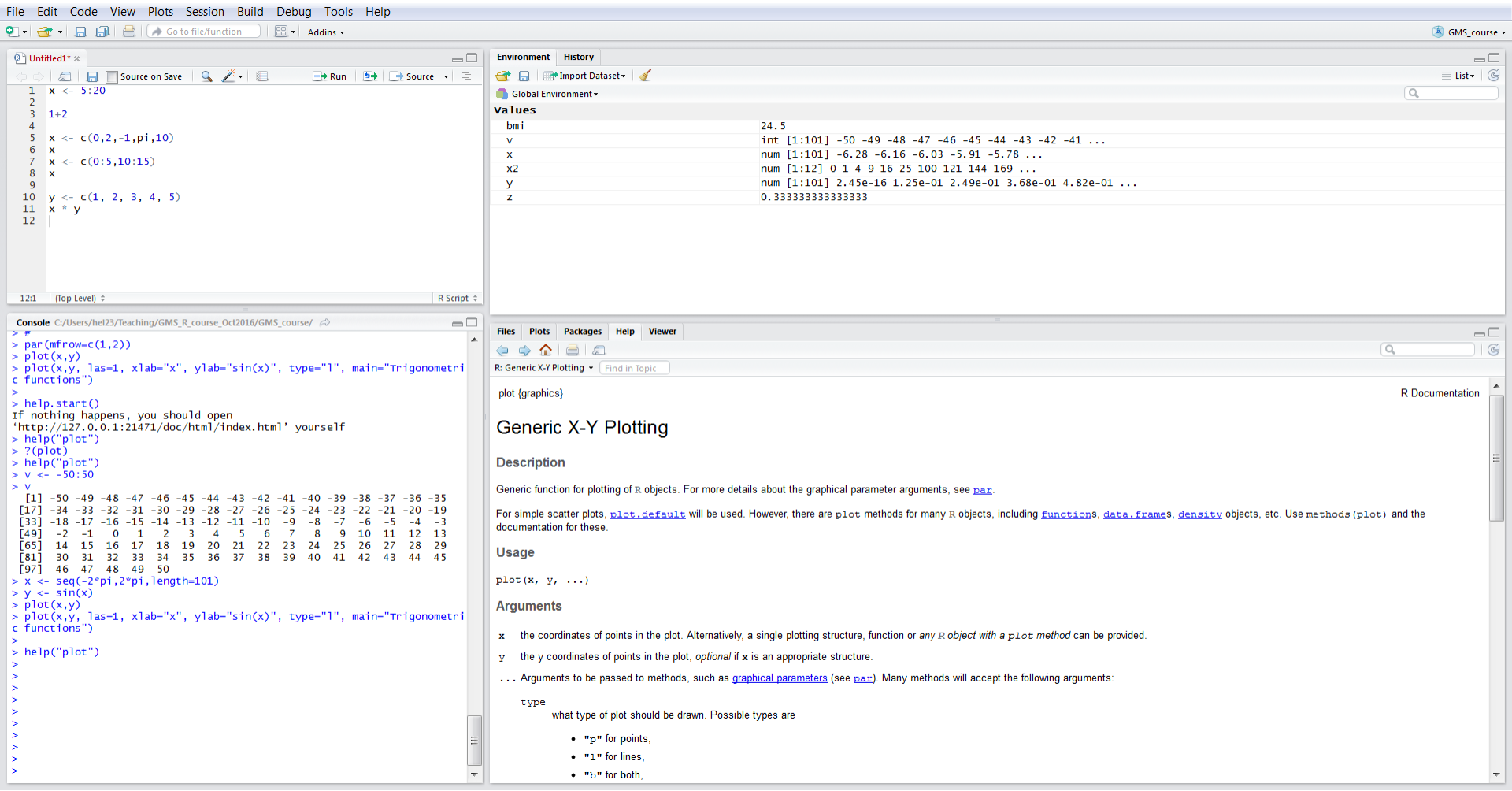
RStudio Interface



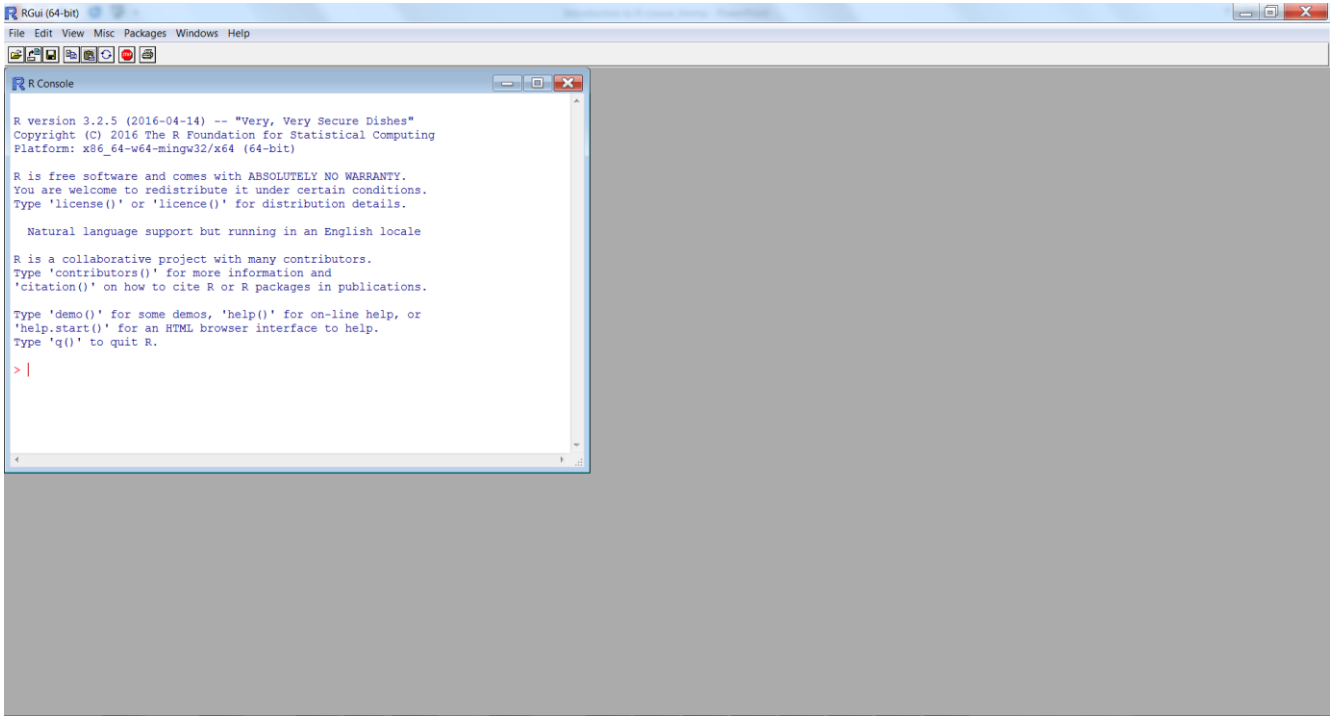
Plots, Files, Packages, help etc…

Slide courtesy of Quentin Ferry

RStudio on Windows



The R interface (Windows)



- From here we can start working in R and do thinks like:
  - Check what packages are already installed
  - Install new packages
  - Check the current working directory or change to a new one
  - Access the help options
- As always, there are several ways to do things

Getting Started with R

Notation

- Throughout these slides, text in red denotes commands that you can type in your R session (it is good to type them directly rather than copy and paste to get used to syntax of commands and expressions, at least to begin with)
- Note that commands are shown including the prompt sign > but ignore this when entering the command. For example:

```
> x <- 5
```

should be entered at the command line as:

```
x <- 5
```

- An extra > as well as the in-built R prompt will give an error:

```
> > x <- 5
Error: unexpected '>' in ">"
```

- Commands are shown in red, and output in blue (this is the convention in the standard Windows R GUI but differs in other interfaces)

R basics: command line

Getting used to the command line (in red):

Prompt [Whitespace] Command [Whitespace] Return

```
> 1+2

[1] 3
```

Results are output underneath (in blue). Elements of the output are indexed by the square brackets

- The prompt tells you that R is waiting for you to enter a command – type a valid command and press return or enter
- Some output may be printed to the console (as above) or a new variable may be created for example
- The prompt symbol will be displayed again afterwards
- Sometimes a + appears instead of the usual > prompt symbol. This means the previous line was incomplete while a command that is invalid will generate an error message

R basics: assignment

Variable name Assignment operator (gets) value

```
> x             <-          1
# Inspect the contents of the new variable named x
> x
[1] 1
# we can also write this as
> x = 1
```

Note that the contents of any variable will be overwritten if later assigned something else:

```
> x <- 4
> x
[1] 4
```

Naming variables

- R is case sensitive so x and X are different variables:

```
> X
Error: object "X" not found
```

- Variable names are chosen by the programmer and should start with a letter followed by letters and numbers – informative names are helpful for several reasons
- You can use . or _ to separate parts of a variable name but not whitespace since this is used to detect the end of a token ('word'). A variable named data.norm or d.norm or dataNorm is fine, but trying to assign a value to a variable named 'data norm' will give an error:

```
Error: unexpected symbol in "data norm"
```

- Elsewhere R ignores whitespace so the following commands are equivalent:

```
> x<-4+3
> x <- 4 + 3
```

Command syntax

```
> x <- c(1:5,8,9)
```
this creates a vector named 'x' containing some numeric values

- If we forget the closing bracket, a + sign indicates the command is incomplete:

```
> x <- c(1:5,8,9
+ )
>
```

- Some text editor programs highlight different types of syntax in different colours or automatically close brackets and quotation marks to help eliminate typing mistakes
- If we can't simply continue our command, use Esc or control-C to return to the prompt and start again
- There is also a useful command recall option – you can use the up/down arrows to scroll through previously entered commands, which can be edited or re-run to save typing again

Data Types and Structures

Common data structures (object types)

- To do anything useful in R, we need to use **objects** to hold data or information and perform various operations on them
- The terms 'object', 'variable' and 'data structure' can all refer generally to objects created in R. There are several different data structures in R including:
  - Vectors
  - Factors
  - Matrices
  - Dataframes
  - Lists

Common data structures (object types)

- Objects can be created in many different ways and hold different kinds of information
- Unlike other programming languages, there is no need to initialise a variable or object in R (define it before first use) – it can simply be created and used directly

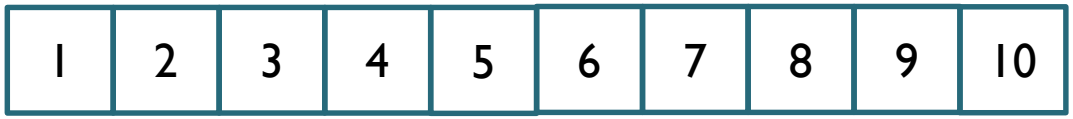
```
> x <- 1:10
> x*2
 [1]  2  4  6  8 10 12 14 16 18 20
```

- We'll work through some examples of each type and look at ways to access or manipulate the data contained within an object
- Be aware that the type (class) of an object and data it contains (numeric, character etc) can affect how it is treated by R

Vectors

- One-dimensional objects with length attribute

```
> x <- 1:10
```




```
> x <- c(1:5, 10:14)
```


```
> length(x)
[1] 10
> x[3]; x[7]; x[1:5]
[1] 3
[1] 11
[1] 1 2 3 4 5
```

Vectors

- Elements automatically added to create vector of appropriate length

```
> x <- 1:15
```

- Can contain numeric or character data

```
> class(x)
[1] "integer"


> x <- c("a", "b", "c")
```


```
> class(x)
[1] "character"
```

R basics: Factors

- Factors are useful for handling categorical data

```
> groups <- rep(c("WT", "MU"), each=3)
> groups
> class(groups) # character vector
> groups <- as.factor(groups) # coerce to factor
> class(groups) # factor
> groups
[1] WT WT WT MU MU MU
Levels: MU WT

> table(groups) # useful summary function giving the
number of entries for each level of the factor

> groups <- factor(groups, levels=c("WT", "MU")) # re-
order the levels (default alphabetical)
```

Matrices

- Two-dimensional object with row and column number attributes

```
> x <- 1:20
> y <- matrix(x, ncol=2, nrow=10)
> dim(y) # get the dimensions
[1] 10  2
> head(y) # shows the first 6 rows
> colnames(y) <- c("Col_1", "Col_2")
> y
```

- All columns of a matrix contain data of the same type. A dataframe is a similar structure but the columns can be a mixture of types – very useful for data with associated descriptive information

Accessing elements of an object

- Square brackets are used to refer to specific elements or subsets of a vector, factor, matrix or dataframe

```
> x <- 1:20 # vector again
> x[5] # print 5th element of x to screen
[1] 5
```

- For a 2 dimensional object need two indices separated by a comma [row, column]

```
> y[2,1] # specific element – second row, column 1
> y[1,] # all of first row
> y[1:5,1] # first 5 entries in column 1
```

The working directory

- There are many useful functions in R for manipulating data stored in vectors, matrices or dataframes - best introduced through practical exercises of the online tutorial
- One important concept before we get started is the **working directory**. If we want to read in data from existing files or create new ones to save any plots or analysis results, R needs to know where to find/save them.

```
> getwd() # tells us the current working directory
> setwd() # allows us to set a different working directory
```

  - The menu bar also has a 'Change working directory' option

Getting started…

- Online tutorial for R programming:
  - http://swcarpentry.github.io/r-novice-inflammation/
- To get set-up to start the tutorial, click the 'Setup' link in the menu at the top of the page or navigate to
  - http://swcarpentry.github.io/r-novice-inflammation/setup/
- Download the required data files, then return to the main tutorial page and click on the 'Analyzing Patient Data' topic
- Try typing the first command into your R session:
  - `setwd("~/Desktop/r-novice-inflammation/")`
  - You may get an error message if your actual directory structure does not match the command (likely to happen on Windows machines)
  - See guidance notes for details or ask for help