R: Introduction to Basic Features

Helen Lockstone
hel23@well.ox.ac.uk

iT Centre Learning

iT services

UNIVERSITY OF OXFORD

Aims of today's workshop

- A introduction to how you interact with and use R
- A sense of R's capabilities and how it works
- Explain some programming jargon and concepts, and the R language
- Awareness of the vital importance of good programming practice
- Use R to run some typical tasks

R – Strengths and Weaknesses

- Incredibly powerful and versatile statistical programming software….but where do I start?
- Features that make life easier in many ways….but potential pitfalls as well
- Open-source, free software with a strong support and development community
- Extensive additional functionality for Genomics data through the Bioconductor project (https://www.bioconductor.org/)

Course Overview

To harness its capabilities, an R user needs to be able to do all of the following:

1. Program instructions in the R language
2. Understand R's data structures, functions and behaviour
3. Use appropriate statistical tests and models for their data
4. Interpret the output correctly

On this course we focus on the first 2 items, introducing the R programming language and starting to handle data in the R environment. The course is intended to help those new to programming get acquainted with using R and help overcome some of the barriers created by technical jargon and notation.

About you

1. I am using R for the first time today.

2. I tried using R but didn’t get very far before I ran into something I didn’t understand or a problem I couldn’t resolve.

3. I’ve used R quite a bit, but sometimes it does unexpected things and I am not sure why.

4. I’ve started using R and thought ‘Wow, this is wonderful – how easy and intuitive it is to use!’ Anyone??

About me

- Lead the Bioinformatics Core at Wellcome Centre for Human Genetics and have many years experience using R for data analysis, particularly gene expression data.

- I and other colleagues have developed and taught a range of R and Bioinformatics Data Analysis courses to DPhil students and post-doctoral researchers over the past few years.

Related R Courses

- R: Kick-off
- R: Introduction to Basic Features
- R: Data Handling
- R: Visualisation

Details at https://help.it.ox.ac.uk/courses/overview

- R: Data Analysis courses coming soon

Schedule

13:30 – 14:00	Introducing the R environment, basic commands and data types
14:00 – 14:30	Understanding variables, functions and arguments
14:30 – 14:40	Tea/coffee break
14.40 – 15:30	Setting the working directory, reading in and accessing data
15.30 – 16:30	Performing simple data analysis and plotting

Course format: informal workshop-style - please feel free to ask questions at any time.

Post-course information:
You can attend one of our related courses or use the Additional Material links to further develop your R skills

Getting Started with R

iT Centre Learning

iT services

UNIVERSITY OF OXFORD

Introductory Remarks

Understanding how to interact properly and carefully with R is vital to analyse your data correctly - mistakes can all too easily arise from oversights such as extracting the wrong portion of data or failing to spot data entry errors or inconsistencies before performing the analysis.

These can also be hard to detect in the output so a key habit to adopt is frequent checks of data objects and their contents, and making plots that will help spot issues or confirm if an output makes sense.

We will spend today's session introducing basic features of R and gaining some familiarity with using it as a programming language and to perform typical tasks associated with analysing data

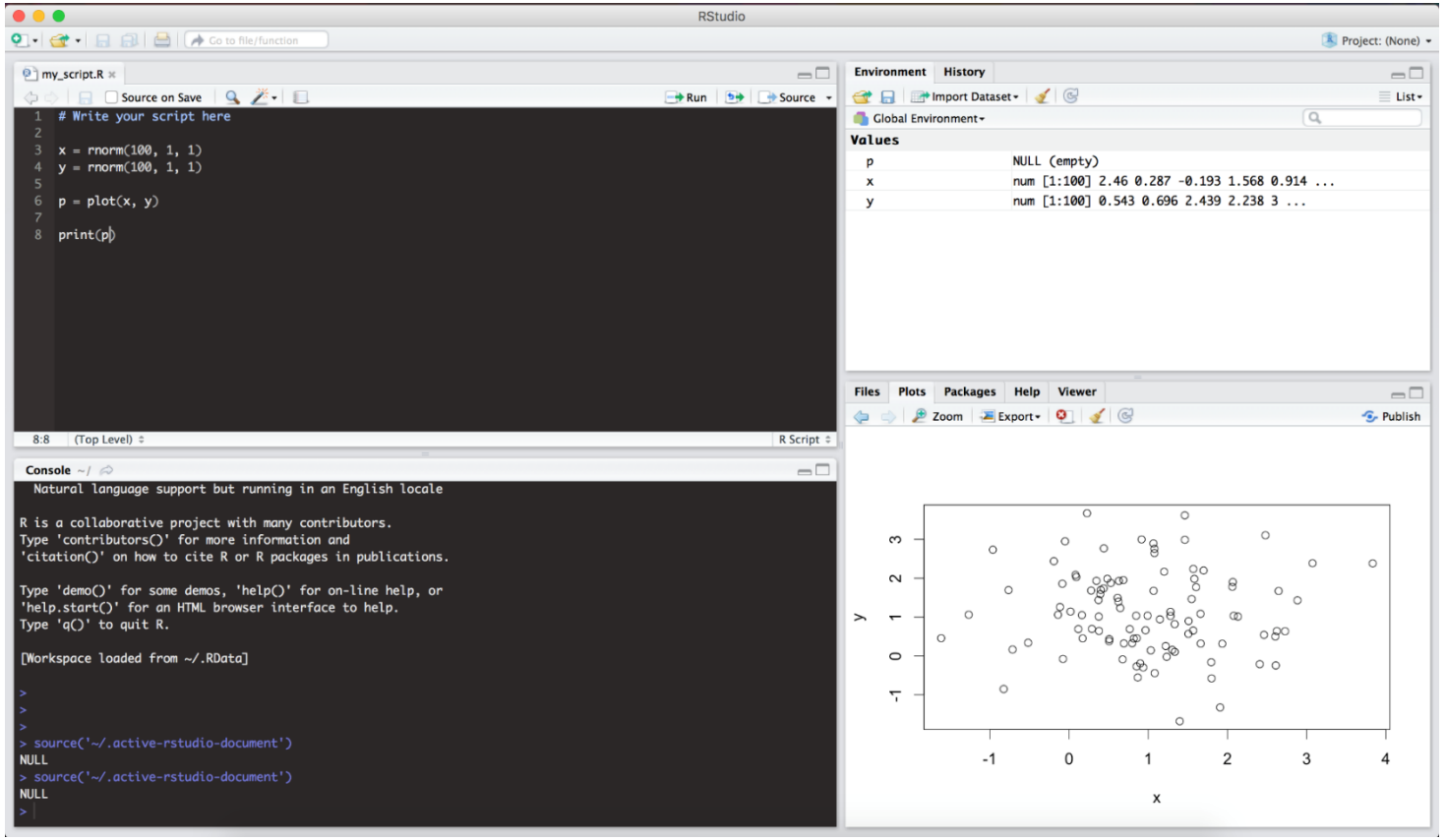
Learning R is a long process but we hope this course will help you get started.

The RStudio interface

RStudio®

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

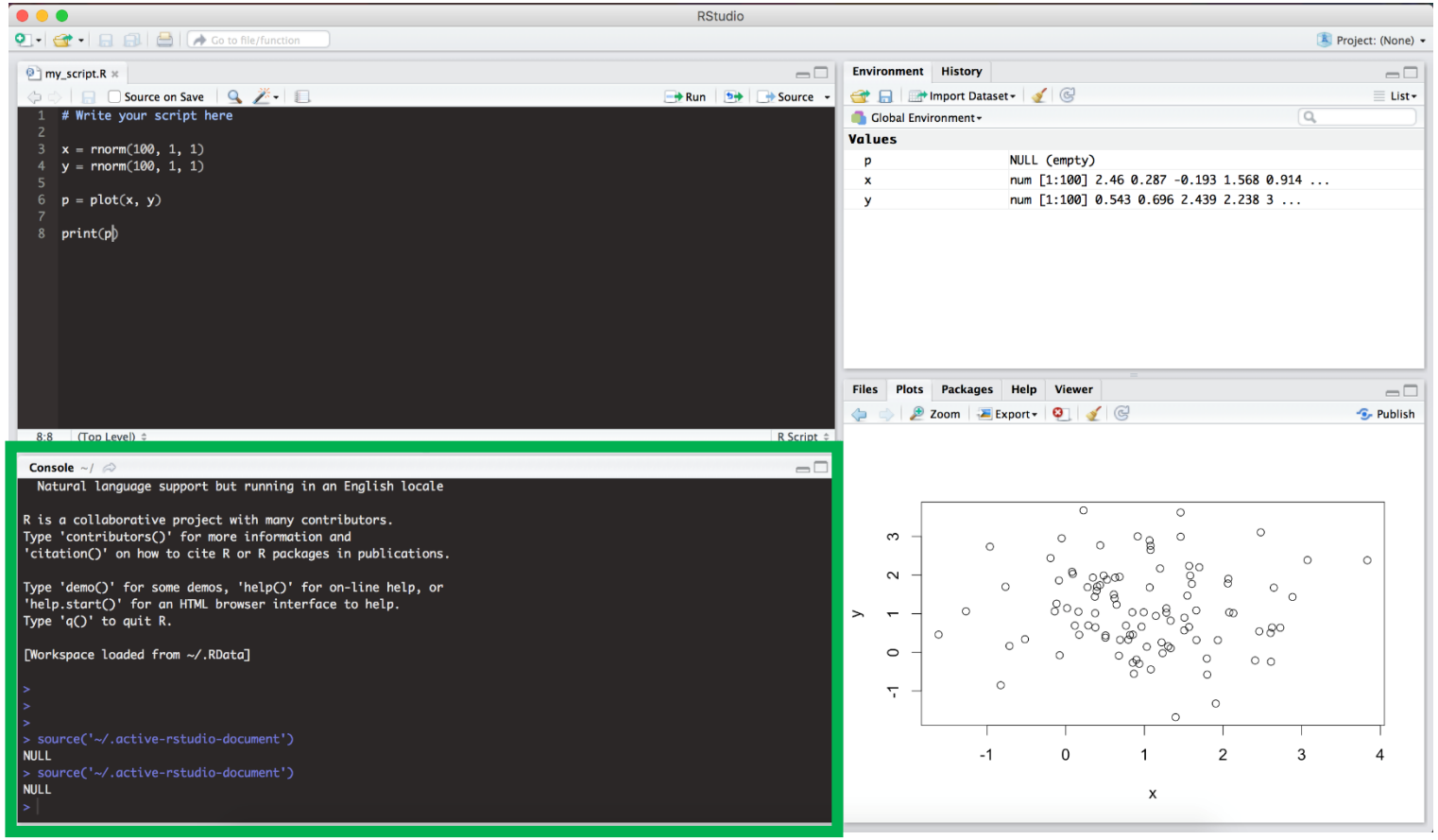
(https://www.rstudio.com)



Slide courtesy of Quentin Ferry

The RStudio interface

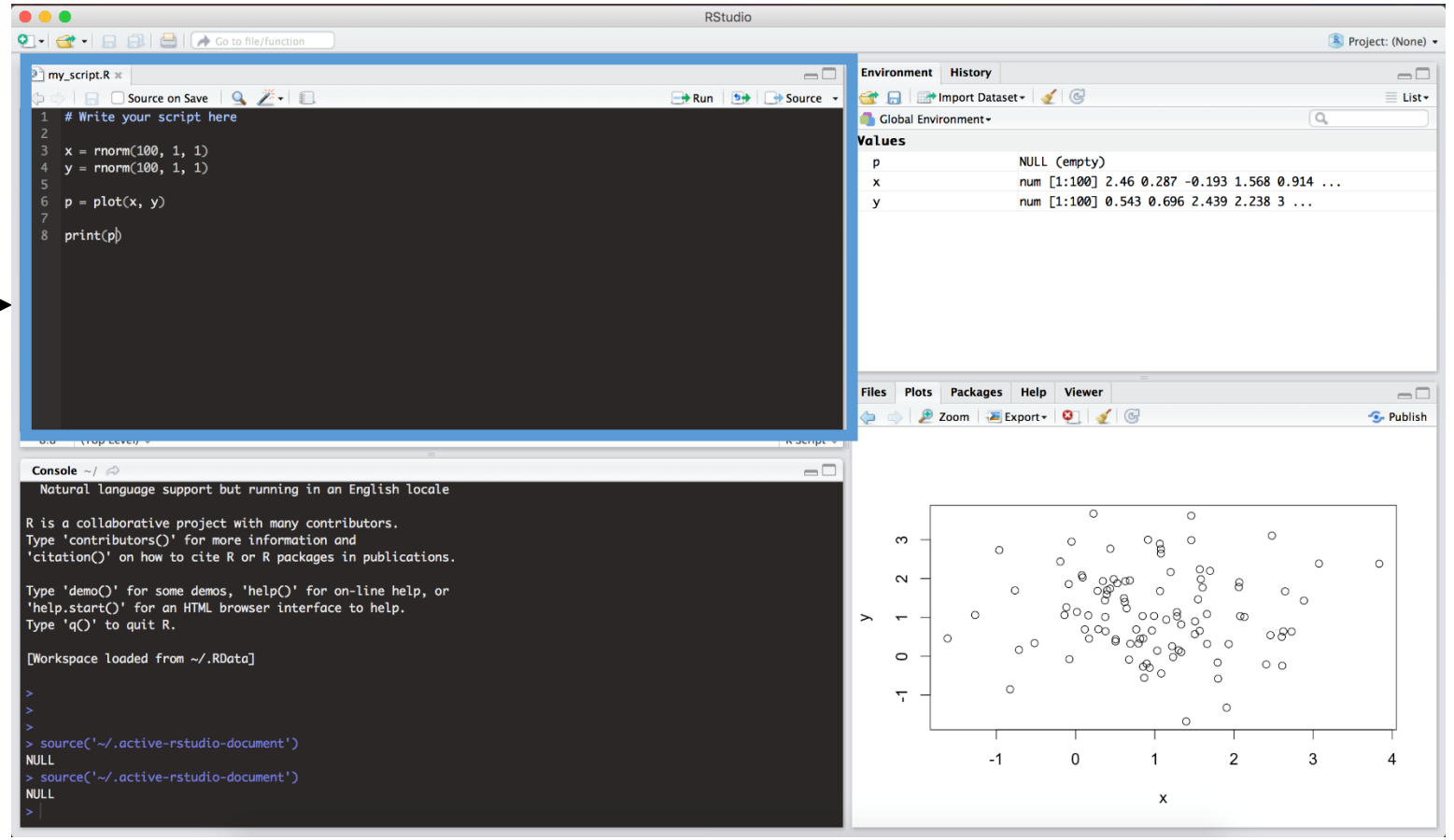
Console for entering
R commands



Slide courtesy of Quentin Ferry

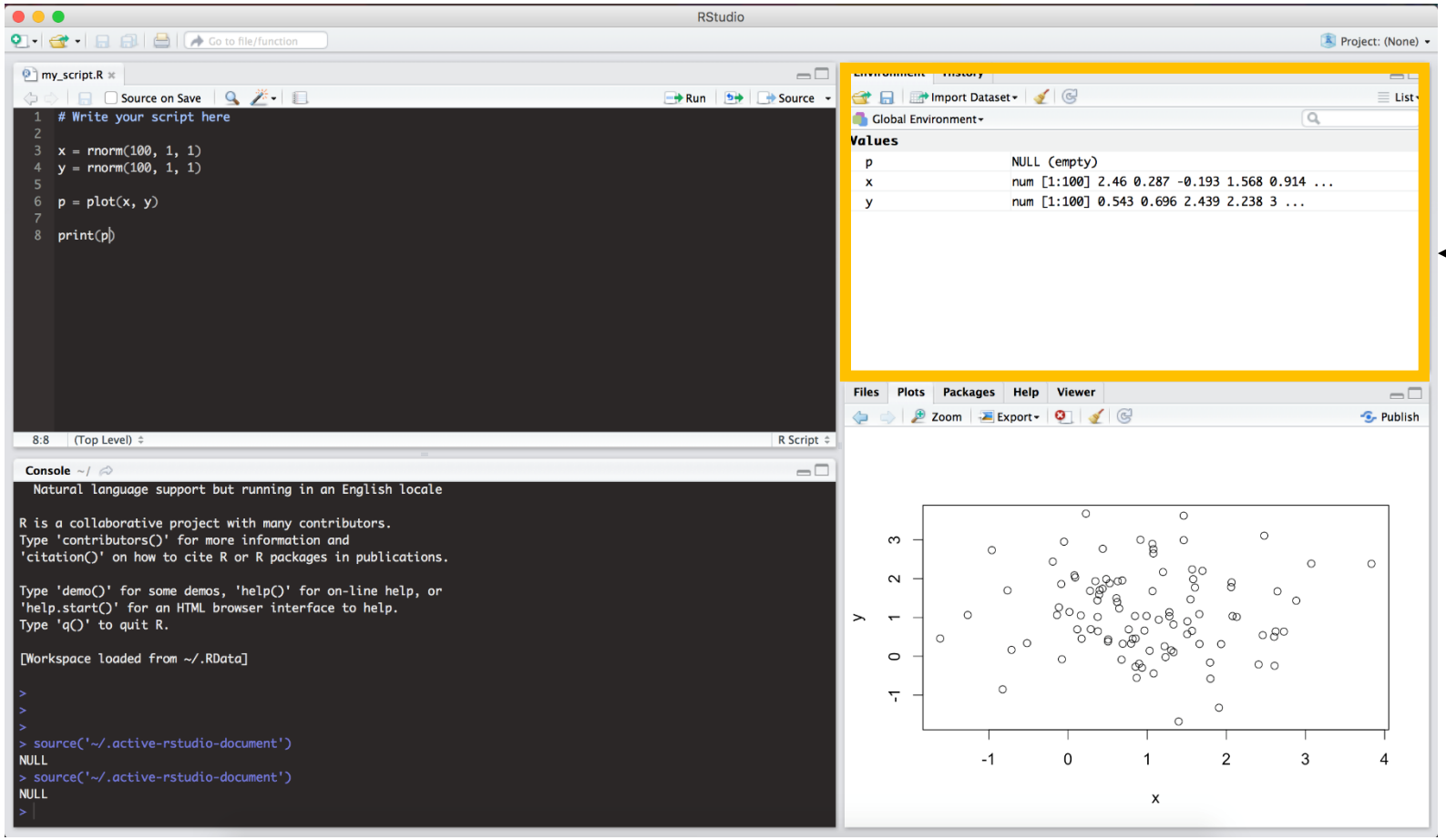
The RStudio interface

Script, data tables etc



Slide courtesy of Quentin Ferry

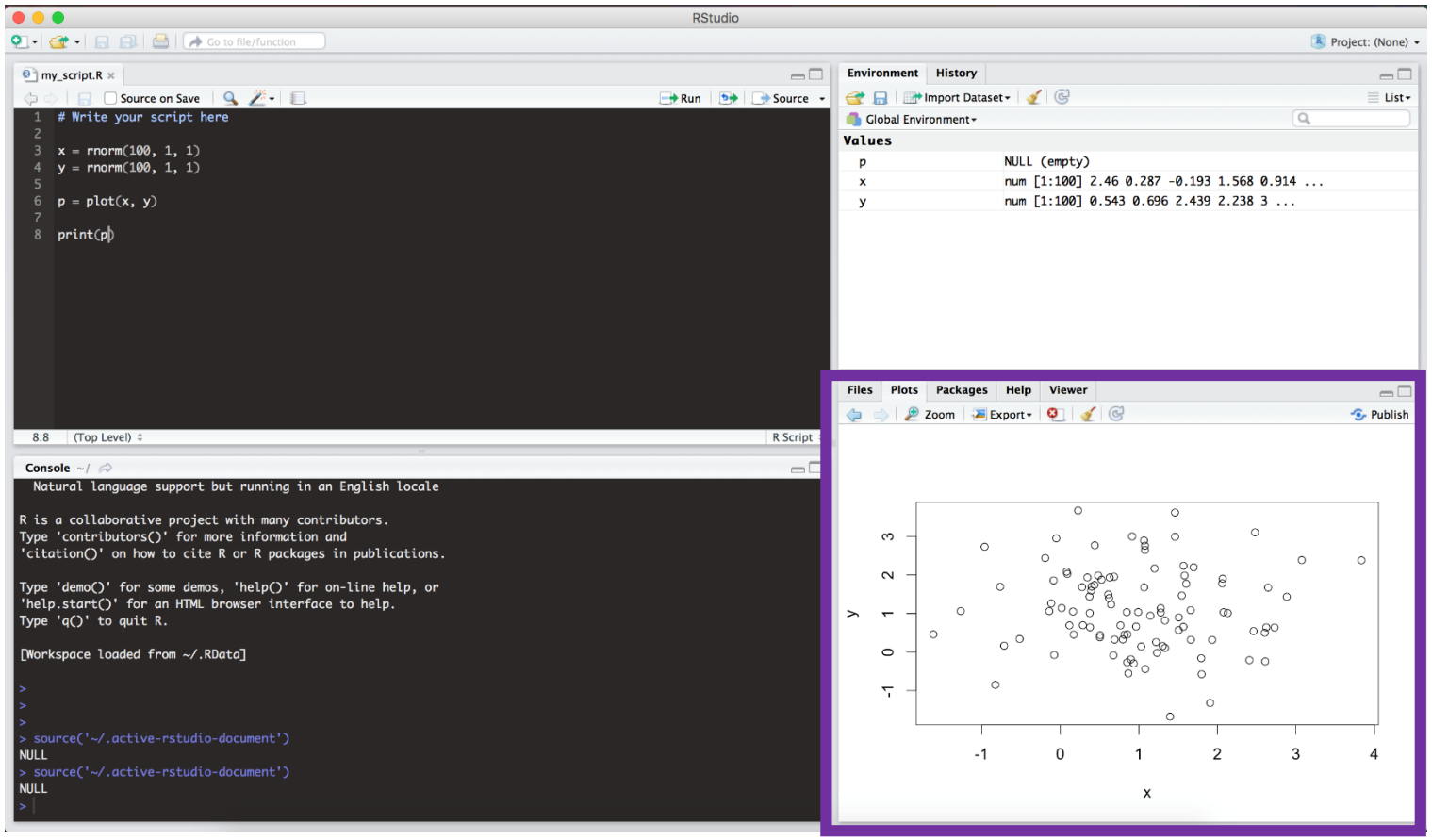
The RStudio interface



Clickable list of objects in memory

Slide courtesy of Quentin Ferry

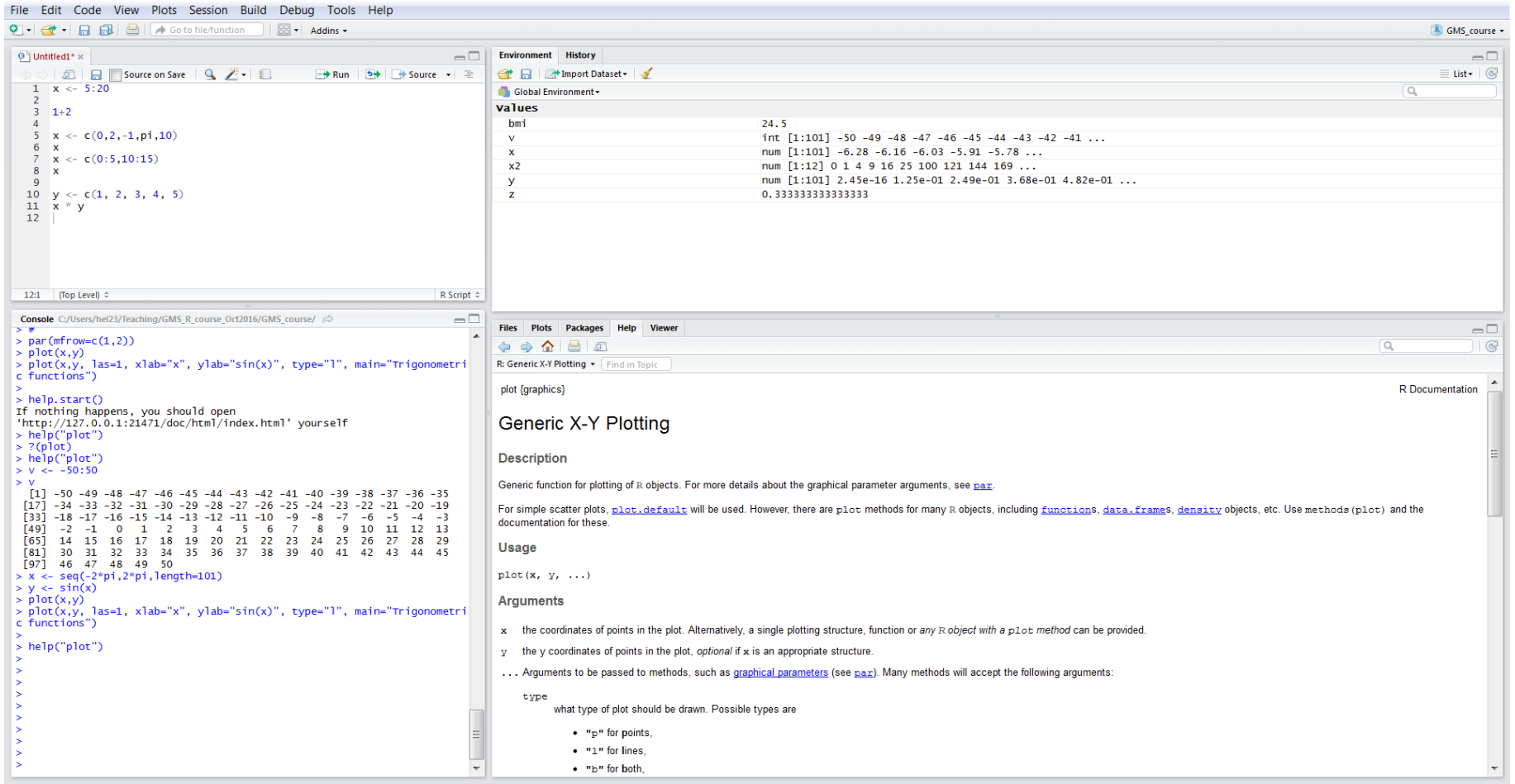
The RStudio interface



Plots, files, packages, help etc…

Slide courtesy of Quentin Ferry

RStudio on Windows



R basics: the R environment

R is a very interactive environment – commands can be entered into the console one by one, and each is interpreted and executed by R in real time. R is a high-level programming language, meaning that it is fairly human-readable.

Depending on the command, different things may happen – a new object might be created, some output displayed, computations performed, plots generated etc.

If a command is not valid in the way it is constructed (its syntax), R will print an error message to the screen. These can sometimes be hard to interpret but particularly common culprits are simple typing mistakes or quotes and brackets, whether they are in the wrong place, missing, or not in pairs (e.g. missing a closing bracket).

We will start by entering a few simple commands and discuss what is happening. For now we will work directly in the console but later we will save our code in a script.

R basics: 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 this appears by default in the R console and you don't need to type it yourself. 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 [press Return key]

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

The output of this command (shown in blue) is printed directly to the console before the command prompt appears again. Elements of the output are indexed by the square brackets (in this case there is only one but sometimes longer lists of elements are returned).

Some commands do not produce any output; after executing the line of code, the prompt immediately appears again - it may not look as though anything has happened but it is likely something has changed. It is important to know what each command has done and it was as intended. The new prompt tells you that R is ready for you to enter another command.

R basics: data structures

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.

Although *variable* is a widely used programming term and would be the preferred term in certain situations, I will use *object* as a general term throughout to refer to any of R’s data structures. These include vectors, factors, matrices, dataframes and lists. We’ll focus on vectors initially and meet dataframes later on.

R basics: assignment

The following command assigns the value 1 to a new object we create and name ‘x’.

```
> x <- 1
```

The assignment operator is <- and running this command creates a new object in R’s memory

```
## Inspect the contents of the new object
> x
[1] 1
```

Note that the contents of an object will be overwritten if later assigned something else:

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

We can also perform operations directly on the object (note the object itself does not change):

```
> x * 2
[1] 8
```

Unless we were to re-assign the output to it….

```
> x <- x * 2
```

R basics: naming objects

R is case sensitive so x and X are different:

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

Object names are chosen by the programmer – informative names are helpful for several reasons.
You can use capitalization, _ or . to separate parts of an object name but they cannot contain spaces, nor start with a number. To avoid confusion or potential issues, it is also best not to give them the same name as an R function, which have their own defined names. An object named raw_data, raw.data, rawData (or even d.raw for minimal typing!) is fine, but trying to assign a value to a variable named 'raw data' will give an error because R cannot parse it correctly:

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

Elsewhere R ignores whitespace so the following commands are equivalent:

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

R basics: command syntax

```
> x <- c(1,2,3,4,5)
```
this creates a vector named ‘x’ containing some numeric values

If we forget the closing bracket before pressing enter, a + sign indicates the command is incomplete:

```
> x <- c(1,2,3,4,5
+ )
>
```

- 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

- Rstudio and some text editor programs highlight different parts of the syntax in different colours and automatically close brackets and quotation marks to help eliminate typing mistakes

Data Types and Structures in R

R basics: objects

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. R also automatically decides which of its data structures and types are most appropriate for the data given, rather than being explicitly specified by the programmer.

We'll work through some examples 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.

R basics: creating vectors

There are many shortcuts in R to avoid tedious or error-prone steps. When we created our small example vector containing the numbers 1 to 5, we issued the command:

```
> x <- c(1,2,3,4,5)  # this tells R to concatenate these 5 numbers
```

Equivalently we could write:

```
> x <- 1:5
```

This is very handy if we wanted a much longer vector such as 1 to 100, or 1 to 1000,000

We can also put together non-consecutive strings of numbers or a mixture:

```
> x2 <- c(1,3,5,7,9)
> x3 <- c(1:5, 7, 9, 10:15)
```

If we need to create a sequence of numbers, the function ‘seq’ is very useful.

```
> seq1 <- seq(from=1, to=99, by=2)
> seq2 <- seq(from=0, to=1, by=0.01)
```

R basics: vectors

Vectors are one-dimensional objects; in the case of the object we created ‘x’, it has length 5.

1	2	3	4	5

There is an in-built R function called ‘length’ that we can use to check how long any given vector object is:

```
> length(x)
[1] 5
```

If we change what is assigned to x, the length of the vector is automatically adjusted:

```
> x <- 1:10
> length(x)
[1] 10
```

1	2	3	4	5	6	7	8	9	10

Vectors are R’s primary object type and many computations are highly efficient because they operate on the whole vector at once, rather than element by element.

R basics: vectors

Vectors can contain numeric or character data (or both). We can create a new vector, y, containing the letters 'abc':

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

a	b	c

In Rstudio's top-right panel, we see details of all the objects that have been created in the current session and are available to use. Note the differences between x and y.

We can also see how R has automatically treated them differently by checking the *class* of the objects directly:

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

Now try running the following command:

```
> y <- c(a, b, c)
```

What do you think R has tried to do and why does it result in an error message?

Accessing elements of an object

Square brackets are used to refer to specific elements or subsets of a vector, factor, matrix or dataframe. Creating a new vector as an example:

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

1	2	3	4	5	10	11	12	13	14

```
## extract 3rd element
> x[3]
[1] 3
## extract alternate elements
> x[c(1,3,5,7,9)]
[1]  1  3  5 11 13
## extract subset of elements
> x[3:6]
[1]  3  4  5 10
```

R basics: object classes

R will decide the most appropriate way to store the data it is provided with, and there are ways to convert between different object structures and classes if needed. To give more examples of how data is interpreted by R, run the following and note the results (discuss with a neighbour):

```
> x2 <- c(1:5, 6.5)
> class(x2)
> x3 <- c(1:5, 6.5, "a", "b", "c")
> class(x3)
```

This gives some idea of R's internal rules. Because the way data is being handled by R is important for both performing computations correctly and the source of many error messages, it is useful to be familiar with the common data types. Some functions, such as computing a mean for example, require numeric data objects to operate on:

```
> mean(x2)
> mean(x3)
```

R basics: object classes

The RStudio panel that shows existing objects and displays information on their contents is invaluable. Not only does it save writing separate commands to check these details, it can help you check:

- that your object has been created correctly and contains what you wanted it to
- how R will treat the object internally (when functions are applied to it)
- possible reasons for an error message
- spot any changes that happen to your object (intentionally or otherwise)
- that you do not have too many and/or poorly named objects that could lead to mistakes

TIP Sometimes you are testing things out and creating lots of objects – that's fine but it's good to start a new session when running or checking your final code to be sure previous objects do not affect it in any way. Sessions can also be cleaned up by deleting objects with the command:
rm(object_name)

R basics: functions

There are hundreds, probably thousands, of in-built functions in R. Some you will use very often and others rarely or never. There are always several ways to do the same thing in R, using closely-related functions.

Examples of the functions we have used so far include ‘length’, ‘mean’, ‘class’. The name of the function is followed by a pair of braces (), within which specific information can be provided to the function. These are known as *arguments*, and enable the function to be used in a flexible way. In the case of length, the argument supplied is the name of the object we wish to find the length of. The ‘length’ function is only applicable to vectors (or factors) and does not work on other data types such as matrices or dataframes. We can find this information and details of how to use a function via the relevant help page (these load in the lower right panel of Rstudio for convenience).

```
> help(length)
```

We can check the length of another vector object simply by changing the argument:

```
>length(seq1)
[1] 50
```

R basics: the working directory

There are many useful functions in R for manipulating data stored in vectors, matrices or dataframes - best introduced through practical exercises we will do after the break.

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
```

For the latter, a path to the directory needs to be provided in quotes within the brackets. The path can be full or relative:

```
> setwd("C:/Users/jbloggs/data")
> setwd("./data") # assuming the current working directory is
C:/Users/jbloggs (a level above the 'data' directory)
```

R basics: the working directory

The working directory can also be set and changed via the menu bar:

- Using the R console you find ‘Change dir…’ under the File menu
- In RStudio, the Session menu has an option ‘Set working directory’.

From these you can navigate to the desired directory and select OK to set it as the working directory. Note that this happens behind the scenes and nothing will appear to happen on your screen - you can use getwd() to confirm if you have set it as intended.