INST0065
Data Visualization and GIS

Week 6: Revision of work so far

Dr. Oliver Duke-Williams

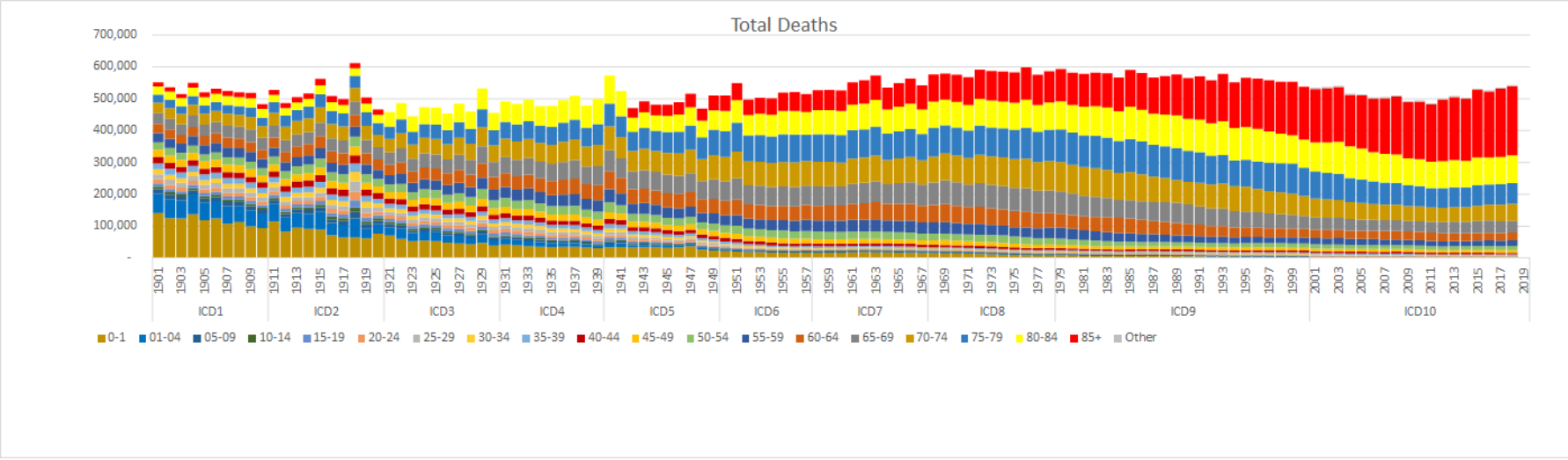
o.duke-williams@ucl.ac.uk
(Please use Moodle forums for messages about this module)

Twitter: @oliver_dw

UCL INFORMATION STUDIES
Celebrating 100 years

Contents

- Thinking about data visualisations
- Review: data types and data structures
- Review: Rmarkdown files
- Review: ggplot options



Source: https://twitter.com/Chains1945/status/1361634345405083654

Review

- We started by looking at what we mean by data and by data visualization
- We need to learn enough R to develop visualizations
  - The basic language
  - The plotting commands etc
- We have moved towards using Rmarkdown files

R – the language

- Everything in R is an *object*
  - You may see reference to different object-oriented programming methods in R ('S3','S4') which we won't worry about
- Objects have values, class, metadata etc

Data types and data structures

- R has the following basic **data types**
  - character
  - numeric
  - integer
  - logical
- Also (which we won't worry about)
  - complex
  - raw

R

- R commands are typed at a console prompt
  - A sequence of commands can be assembled into scripts

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

> x <- 1 + 2 # assignment

> x # typing the name of an object will show its contents
[1] 3
```

This ('>') is the R prompt

Results are prefixed with an index number

Assignment

- We *assign* values to objects with "<-"
- We don't usually need to say what data type we want the object to be

```
a <- "hello" # character
b <- TRUE # logical (aka Boolean)
c <- 23.5 # numeric
d <- 23.0 # also numeric
e <- 23 # still numeric
f <- 23L # 'L' suffix ensures that this is an integer
g <- as.integer(23)# another way of assigning an integer
```

Working with data: operators

- R provides basic mathematical operators
  - The operators +, -, / and * all work in the way that you would expect
  - We can raise a number to a power with ^ or **

```
> 9*2
[1] 18
> 9^2 # 9 raised to the power 2, i.e. 9 squared
[1] 81
> 9^0.5 # 9 raised to the power ½ i.e. square root of 9
[1] 3
```

Working with data: operators

- R has two further useful mathematical operators
  - %% is the modulus operator
    - x%y is the remainder of x when divided by y
  - %/% is in the integer division operator
    - x%/%y is the number of times y goes into x

```
> 5%%2
[1] 1

> 5%/%2
[1] 2
```

Data structures

- R has a number of data structures
  - Vectors
  - Lists
  - Matrices
  - Arrays
  - Data frames

Data structures

- Additional libraries can add new structures
- Tidyverse adds
  - tibble (tbl_df)
    - "*Tibbles are data.frames that are lazy and surly: they do less (i.e. they don't change variable names or types, and don't do partial matching) and complain more (e.g. when a variable does not exist).*"

Data structures

- We have concentrated on
  - Vectors
    - The default data structure
    - All elements the same data type
    - When we assign a simple value, it is a vector of length=1
  - Data frames
    - Consist of rows and columns
    - Each column the same data type
    - We can refer to columns using dataframe$column

Data structures

- R has a number of data structures
  - Vectors
  - Lists
  - Matrices
  - Arrays
  - Data frames

Constructing data frames

- We can create a data frame using

  data.frame(*parameters*)

- This can be assigned as with any other data structure

  df <- data.frame(a=1:5,b=6:10)

a and b are columns

- The size of each component should be the same

```
> data.frame(1:5,6:7,11:15)
Error in data.frame(1:5, 6:7, 11:15) :
  arguments imply differing number of
rows: 5, 2
> data.frame(1:5,6:10,11:15)
  X1.5 X6.10 X11.15
1    1     6     11
2    2     7     12
3    3     8     13
4    4     9     14
5    5    10     15
> data.frame(a=1:5,b=6:10,c=11:15)
  a  b  c
1 1  6 11
2 2  7 12
3 3  8 13
4 4  9 14
5 5 10 15
```

- First example – input vectors of different sizes
- Second example – consistent input, vectors not named
- Third example, inputs labelled

Working with data frames

- Data frames have associated metadata

```
> df <-
data.frame(a=1:5,b=6:10,c=11:15)
> dim(df)
[1] 5 3
> colnames(df)
[1] "a" "b" "c"
> rownames(df)
[1] "1" "2" "3" "4" "5"
```

Working with data frames

- We can assign new column names or row names

```
> colnames(df)
[1] "a" "b" "c"
> colnames(df) <- c("red","blue","green")
> df
  red blue green
1   1    6    11
2   2    7    12
3   3    8    13
4   4    9    14
5   5   10    15
```

Working with data frames

- We can refer to elements using numbered matrix notation, or to named elements

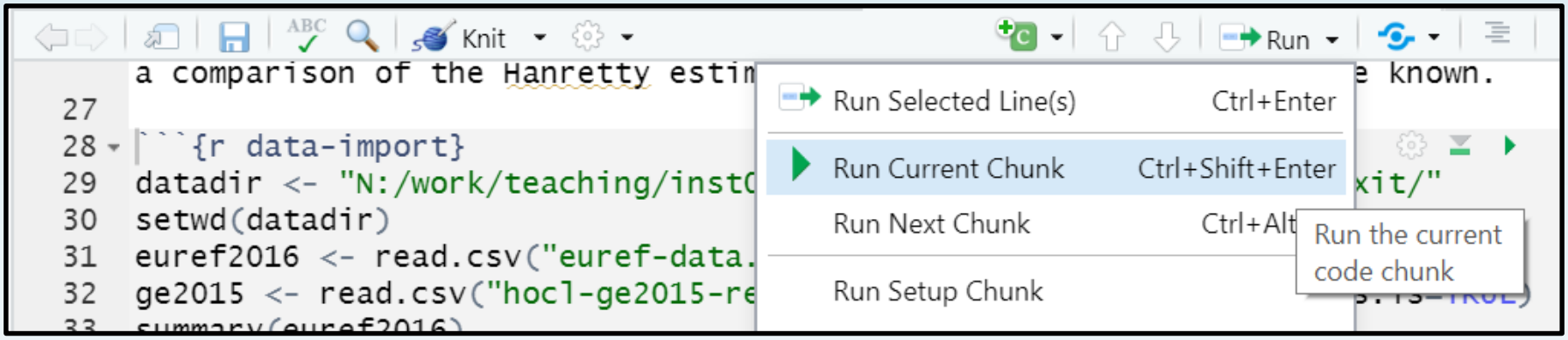
```
> df[3:4,1:2]
  red blue
3   3    8
4   4    9

> df$blue
[1]  6  7  8  9 10
```

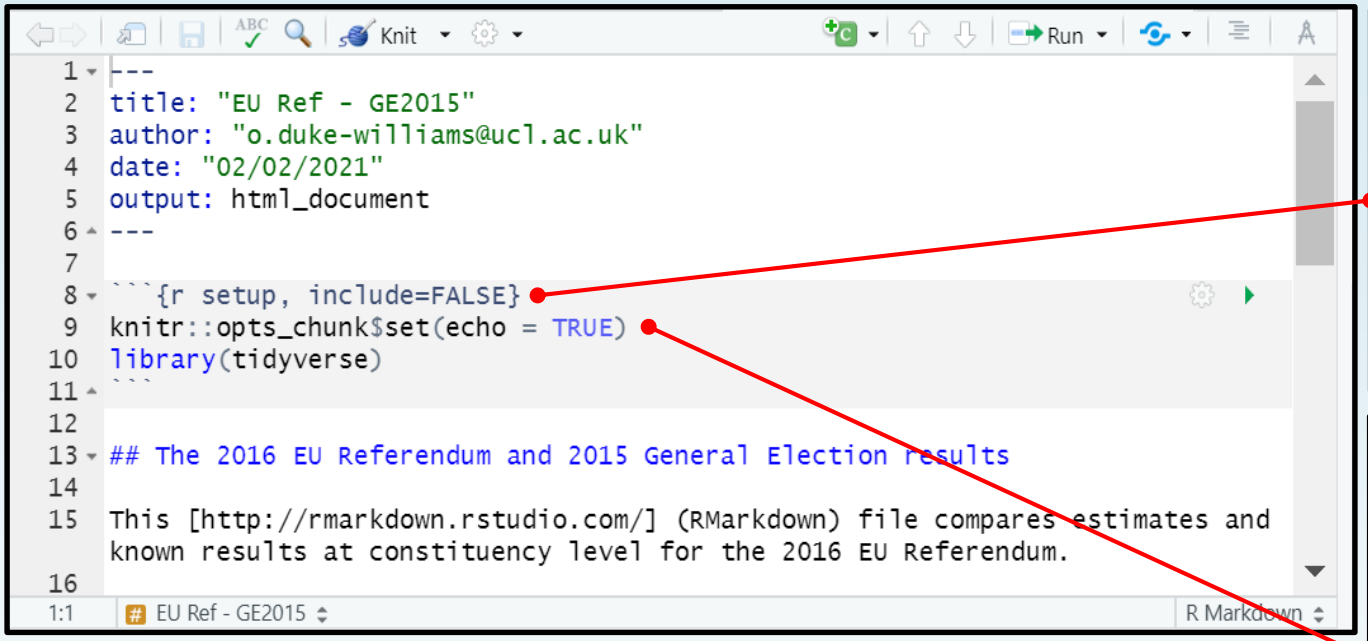
Getting started with RMarkdown

- As a reminder, we can either use an existing template, simply start writing a file from scratch, or use the example 'new file' template in RStudio
- We'll do the latter
  - In RStudio
    - File -> New file -> RMarkdown
    - Set title and check author details
  - In RMarkdown file
    - Delete everything after initial options

Working with RMarkdown



- Each chunk should do a distinct task, rather than have a monolithic amount of code
- Each chunk should have a **unique chunk name**
- Each chunk can be tested as you write the code
- You should check that the whole file works as expected before passing it on



include=FALSE

This is an option for this chunk; the R code will be run, but neither the command nor any output will be displayed

knitr::opts_chunk$set()

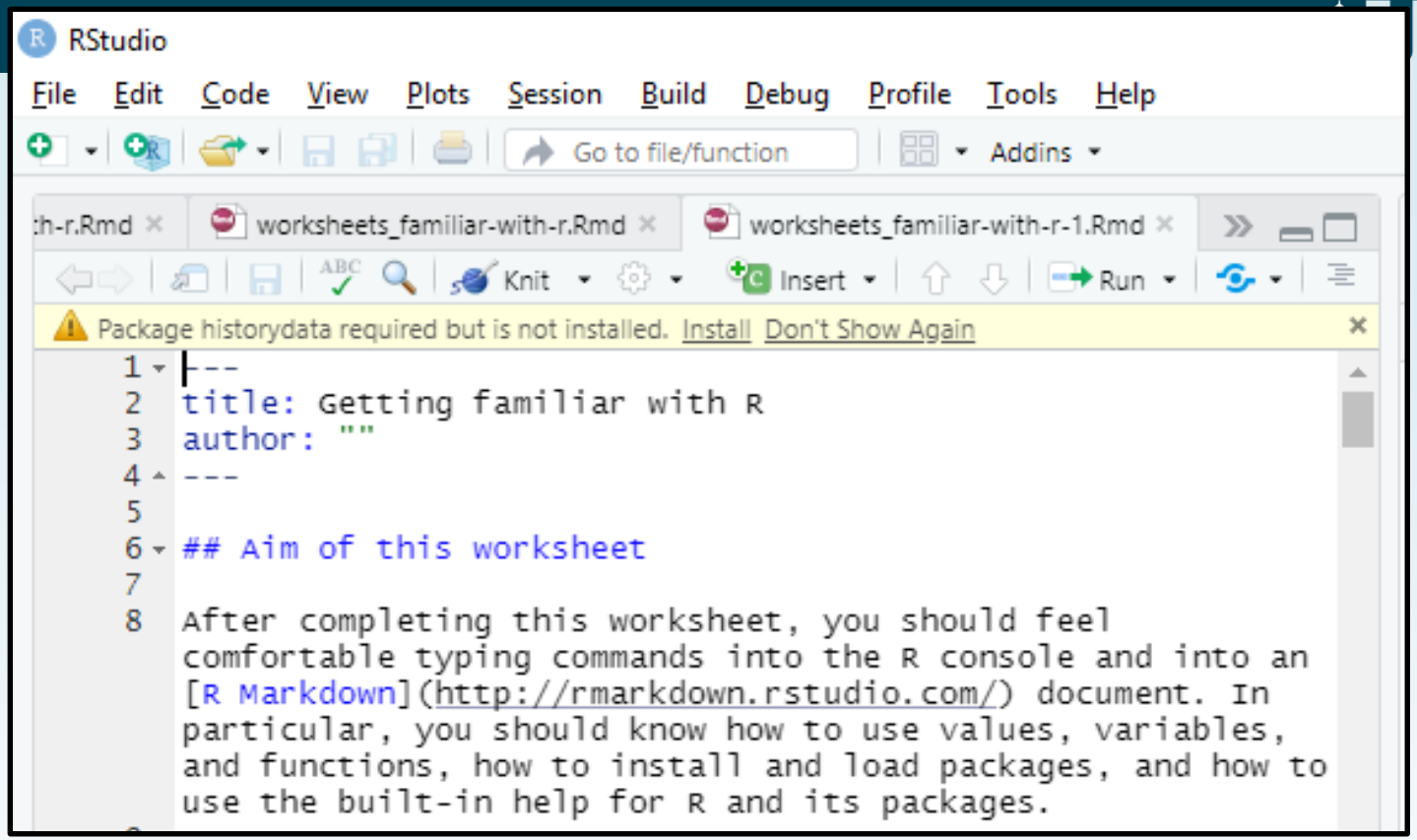
Here we set default options for all chunks.

echo=TRUE means the the R commands are shown in knitted output as well as the results

- Here we have
  - Loaded the tidyverse library (line 10)
  - Started to replace some of the text
- Assume that you are going to give the results to someone else
  - Make sure that you explain what you are doing, and details of the data that are used

Revision markdown files

- Today we've using a series of Rmarkdown files included in the book *Computational Historical Thinking* (https://dh-r.lincolnmullen.com/)
  - These are a very useful resource
  - They mix a summary of things that R can do, with a set of tasks for students to complete
  - They act as a good revision base for us



- The historydata package is used in these worksheets
  - Rstudio may offer to install this
  - You can also install it with:
    `install.packages("historydata")`

https://dh-r.lincolnmullen.com

Search

Most Visited | Getting Started | Add to My Bookmarks | My Bookmarks | Other Bookmarks

Computational Historical Thinking

With Applications in R

Lincoln A. Mullen

About this book

Computational Historical Thinking is a textbook that teaches you how to identify sources and frame historical questions, then answer them through computational methods. These historical methods include exploratory data analysis, mapping, text analysis, and network analysis. These methods are taught using the R programming language, commonly used by digital historians and digital humanists. Chapters on individual methods ground you in particular approaches, and chapters on case studies of historical research walk you through the process of asking and answering computational history questions.

Values

R lets you store several different kinds of *values*. These values are the information that we actually want to do something with.

One kind of value is a number. Notice that typing this number, either in an R Markdown document or at the console, causes the same number to be printed as output.

```{r}
42
```

(@) Create a numeric value that has a decimal point:

```{r}

```

Rmarkdown files

- "Getting familiar with R"
  - This should all be straightforward
  - A few functions are included that we haven't used before, but they should be obvious
    - length(*object*) tells us the length (number of elements) of a vector etc.
    - glimpse(*object*) shows us part of a tibble / data frame

Rmarkdown files

- "Data structures"
  - Again, mostly straightforward
    - Includes some notes on *matrices* which we have looked at briefly, and *lists* which we haven't focussed on
  - Additional functions etc
    - str(*object*) – tells us the structure of *object*
    - unique(*vector*) – extract the unique elements of a vector
    - Using [[n]] rather than [n] to refer to parts of a list

Rmarkdown files

- "Data structures"
  - Again, mostly straightforward
    - Includes some notes on *matrices* which we have looked at briefly, and *lists* which we haven't focussed on
  - Additional functions etc
    - str(*object*) – tells us the structure of *object*
    - unique(*vector*) – extract the unique elements of a vector
    - Using [[n]] rather than [n] to refer to parts of a list

- "Functions"
  - Includes how to write your own functions; we haven't really done (apart from one exercise)
  - Should be easy to follow

- "Data manipulation"
- "Data visualization"
  - Neither file works for me as downloaded

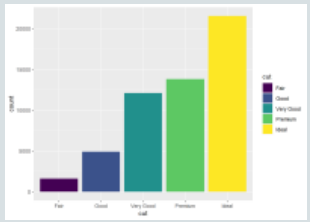
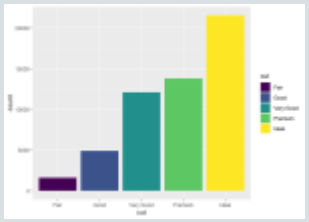
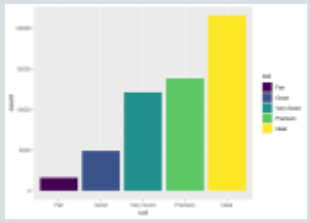
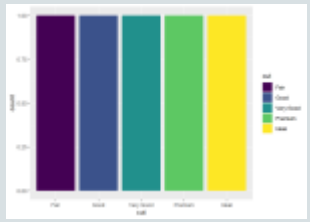
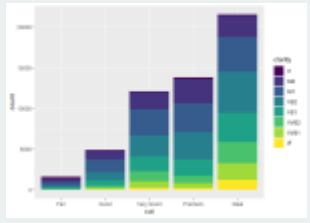
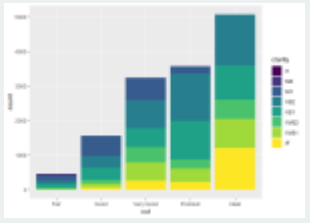
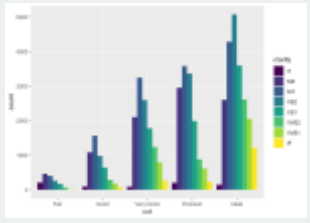
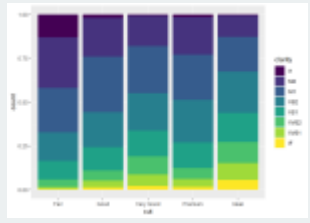
Visualisation with ggplot

- We can plot with ggplot()
- This is an additional library provided as part of tidyverse
- Weeks 4 and 5 focussed on scatterplots and barcharts
- We can build plots up with multiple layers
- For each layer we need to define an aesthetic mapping – how data is presented in output

From last week

- We can look at combinations of
  - Same or different x and fill variables
  - Dummy x variable
  - Stack, identity, dodge and fill options
  - coord_flip()
  - coord_polar(), coord_polar("y")

Effect of fill and position

	stack	identity	dodge	fill
x=cut fill=cut				
x=cut fill=clarity				

Effect of other elements

- We will see this in the attached Rmarkdown file