INST0065
Data Visualization and GIS

Week 4: Using R – a case study

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

This week

- Recap of last week's introduction to R
- A case study: scented candles
- Using RMarkdown

Last week

- R has multiple plotting libraries
- We have used plot() and ggplot()
  - The syntax to create a basic scatterplot in plot() is simple
  - The syntax is slightly more extended with ggplot()

The relationship between limits and axis ticks

- The limits arrays control the data range to be plotted
- The axis parameters control the placing of tick marks
  - A separate axis() function allows more control

Using xlims and xaxp

- The following scatterplots start with:

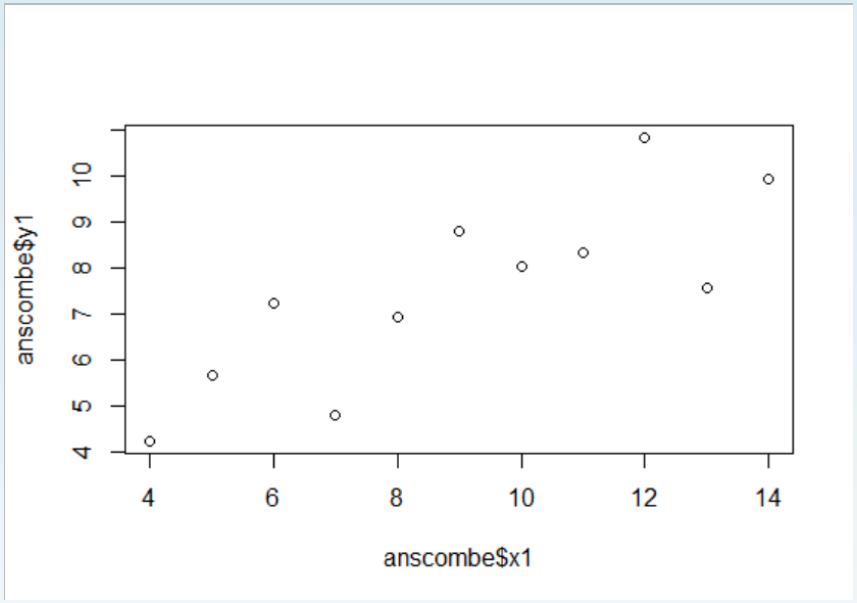
```
plot(anscombe$xn,anscombe$yn)
```

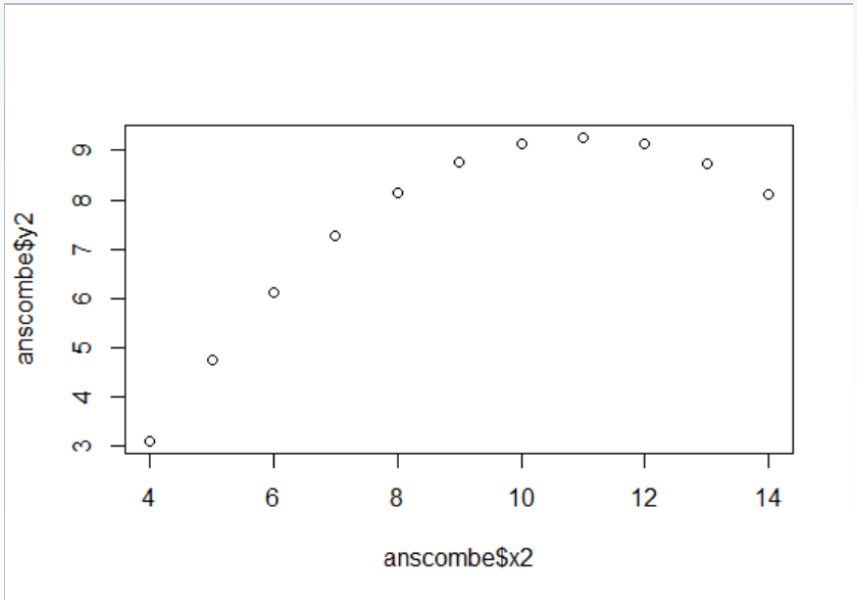
- We will then alter xlims and xaxp, to illustrate how these affect the output

Using xlims and xaxp

Default

- No additional parameters
- The x and y limits of the plot will be based on the input data
  - See difference in x1,y1 and x2,y2



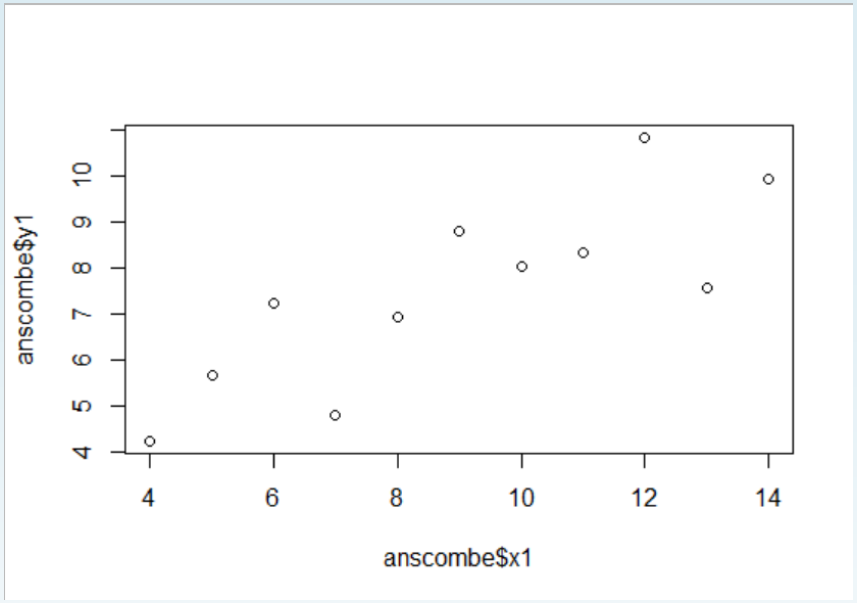


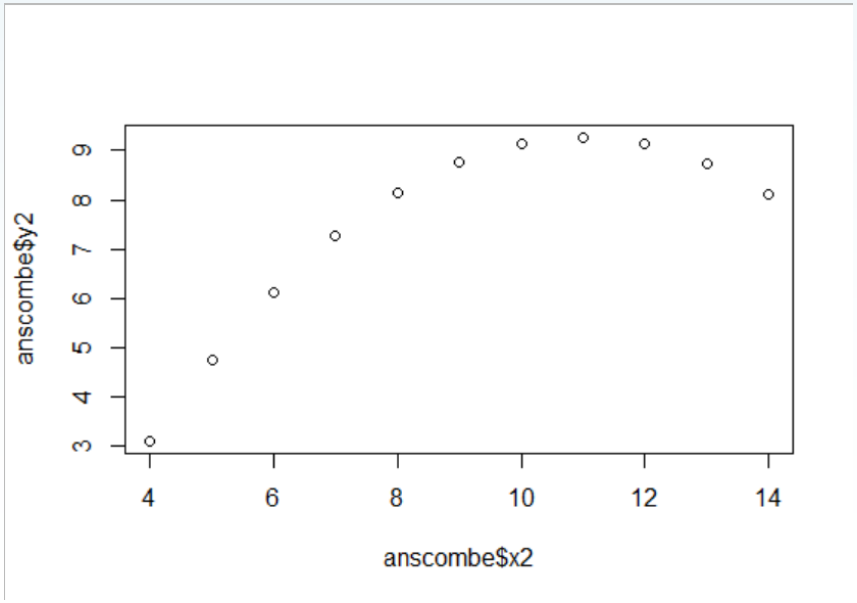
Using xlims and xaxp

Alter xaxp

- plot(….,xaxp=c(start,end, gaps))
- xaxp(start,end,number of gaps between tickmarks)
  - e.g. xaxp=c(4,14,5)
    - Start = 4
    - End = 14
    - Gaps = 5
    - Increment will be:

$$\frac{14-4}{5}=2$$



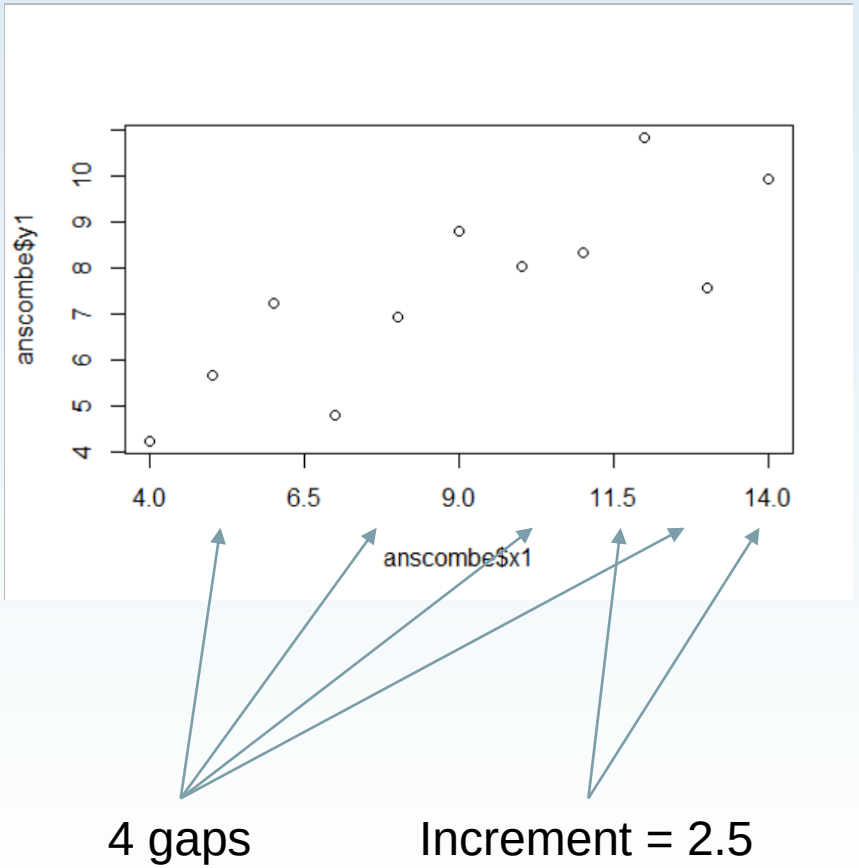


Using xlims and xaxp

Altering xaxp

- xaxp=c(4,14,4)
  - Start = 4
  - End = 14
  - Gaps = 4
  - Increment will be:

$$\frac{14-4}{4} = 2.5$$

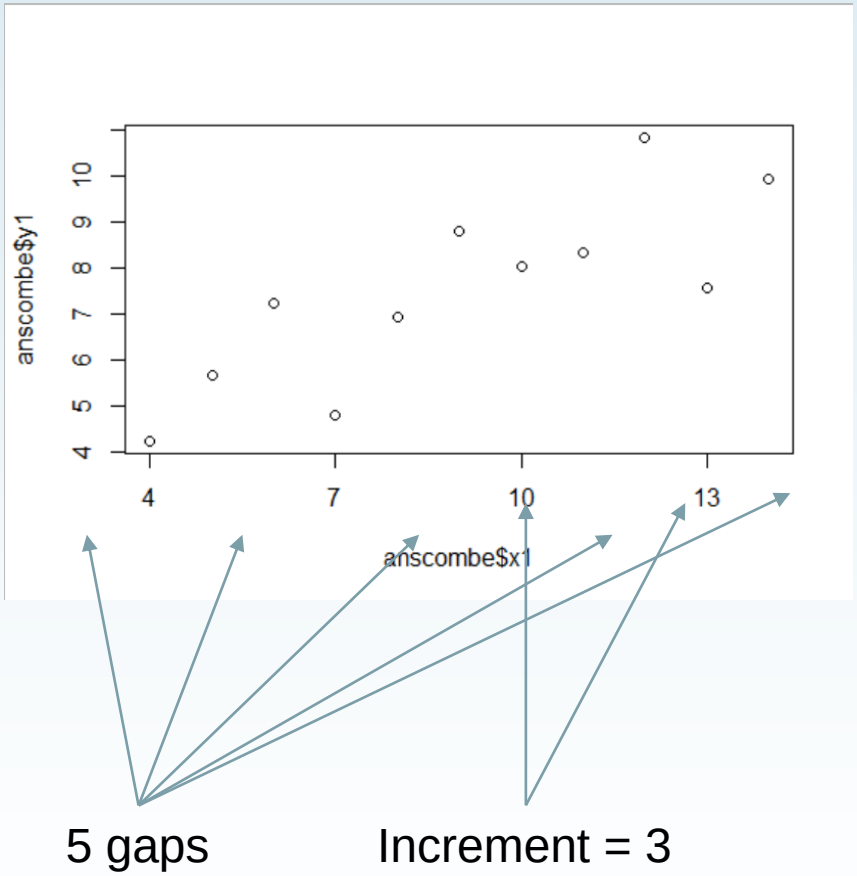


Using xlims and xaxp

Altering xaxp

- xaxp=c(1,16,5)
  - Start = 1
  - End = 16
  - Gaps = 5
  - Increment will be:

$$\frac{16-1}{5}=3$$



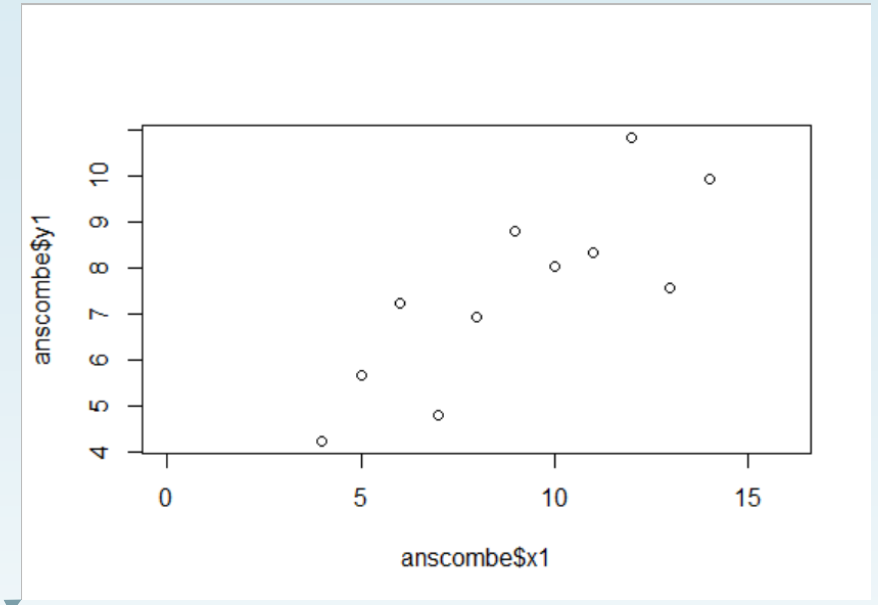
We still have 5 gaps (and thus 6 ticks) – those that don't fit in the range are not plotted

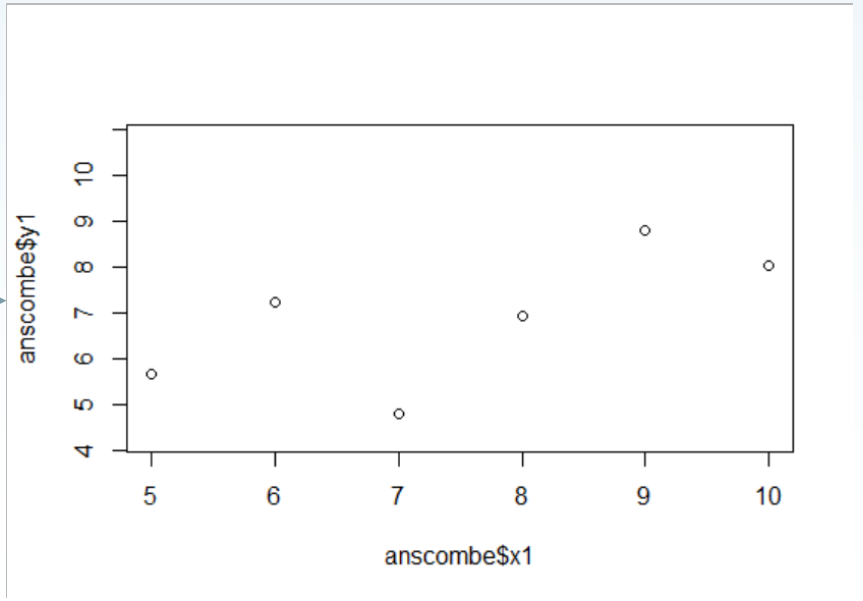
Using xlims and xaxp

xlims

- plot(…,xlims=c(start,end))

- This changes the axis range
  - We can 'zoom out'
    - xlims=c(0,16)
  - We can 'zoom in'
    - xlims=c(5,10)

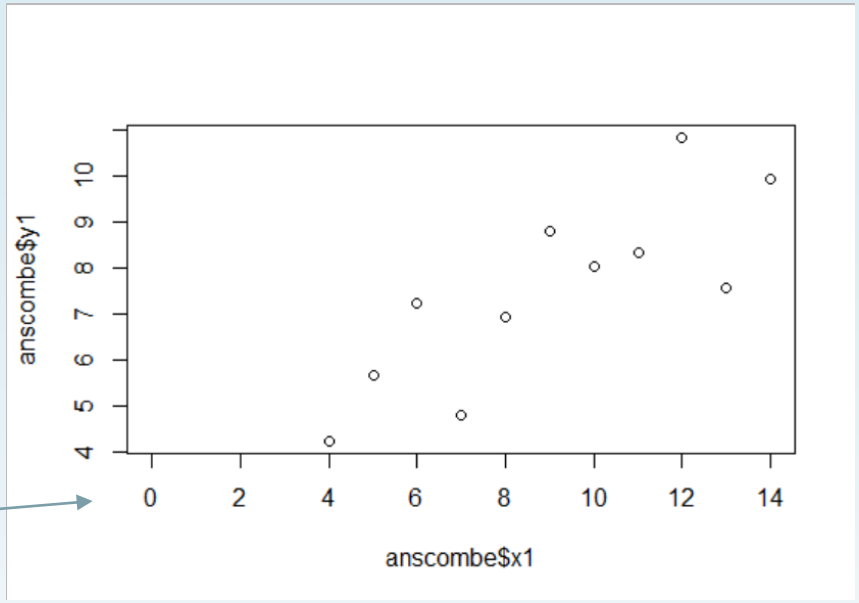




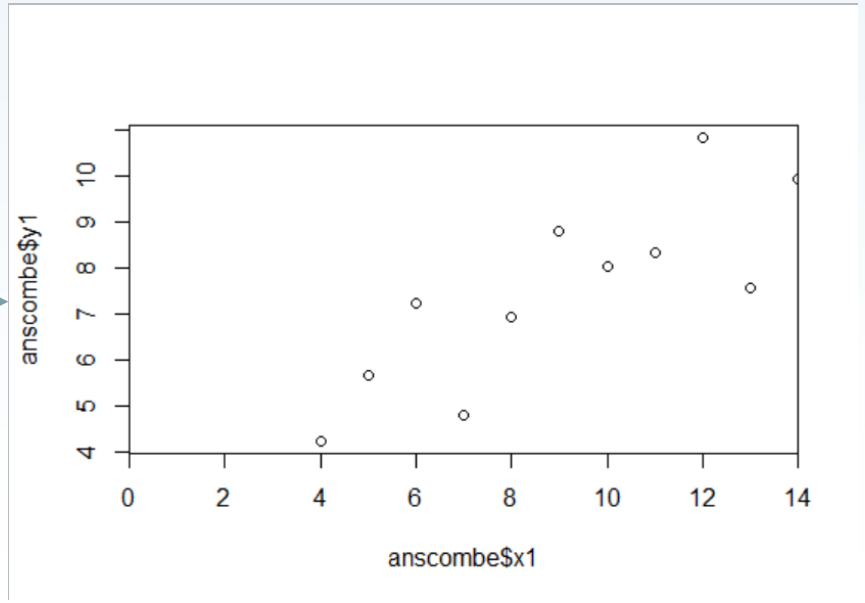
Using xaxs

xlims

- By default, the axis will be padded with a margin
  - xlims=c(0,14)



- This can be suppressed with xaxs="i"
  - `plot(anscombe$x1, anscombe$y1, xlim=c(0,14), xaxs="i")`



Stages in plotting data

- There are a number of stages we can run through
  - Acquiring data
  - Modifying data
    - Re-coding
    - Deriving new values
    - Re-ordering
    - Filtering
  - Plotting data
- The tidyverse collection includes
  - dplyr – handles data modification
  - ggplot – handles data presentation

Notes on documentation

- A variety of sources have been used
- For *tidyverse*, (includes ggplot) the best source is probably:
  - https://www.tidyverse.org/
- For general R documentation, I have mostly used:
  - https://www.rdocumentation.org/
- See also Cheatsheets
    - RStudio: Help -> Cheatsheets
    - Online: https://rstudio.com/resources/cheatsheets/

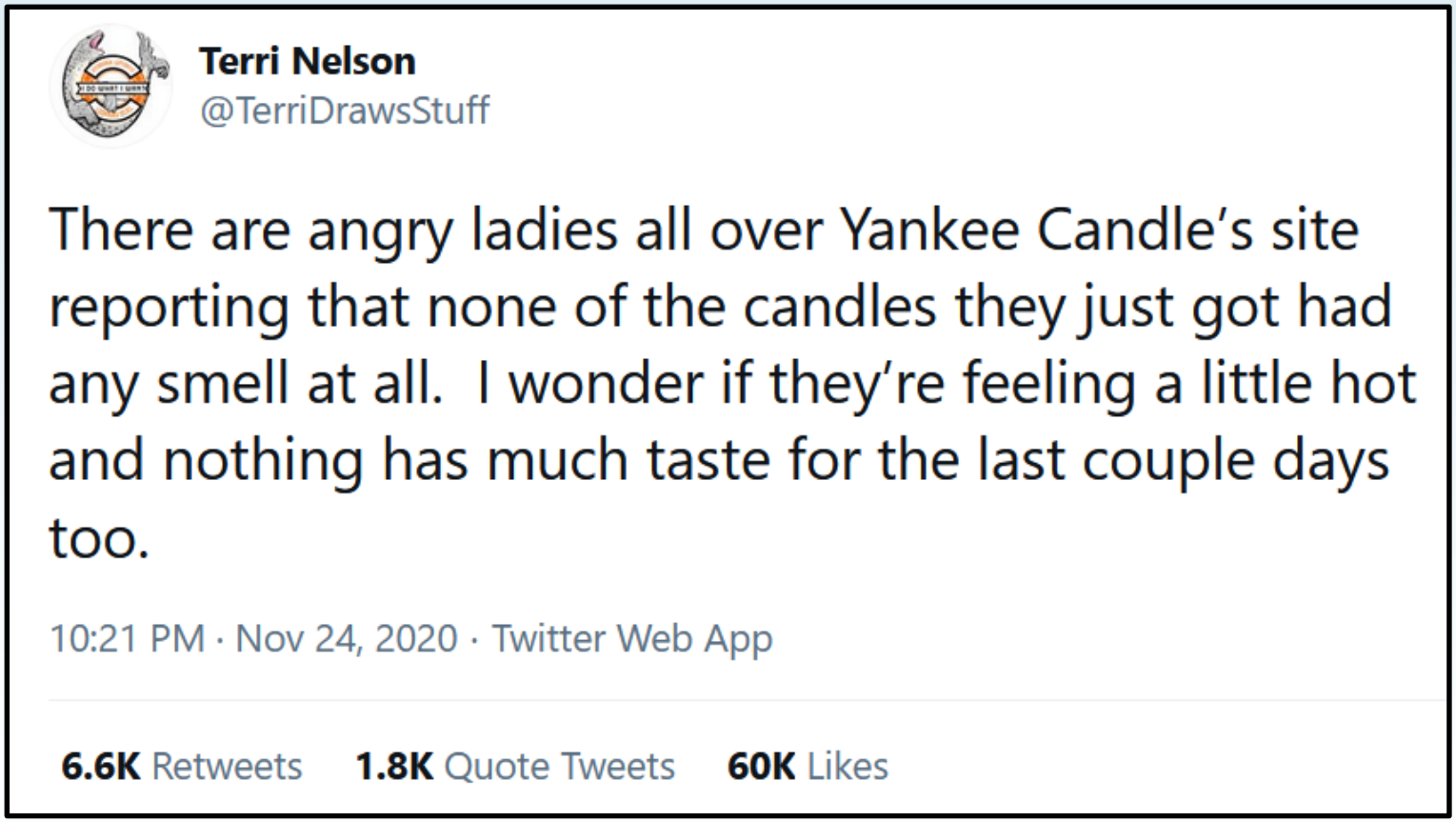
Plotting recipes

- As we proceed, we will generally used 'recipes' to plot some data
  - That is, a command will be shown to plot data in a particular way
  - This can be re-used with other data
  - We should try to understand what is going on in the recipe, so that it can be adjusted to suit new data where needed

Exploring data: scented candles

- This week, we shall look at a claim made in November 2020 about scented candles, and re-create a visualisation which aimed to explore this claim
  - This uses some functions and ideas in R that we have not previously met, so we will study them as we go along

The assertion



Terri Nelson
@TerriDrawsStuff

There are angry ladies all over Yankee Candle's site reporting that none of the candles they just got had any smell at all. I wonder if they're feeling a little hot and nothing has much taste for the last couple days too.

10:21 PM · Nov 24, 2020 · Twitter Web App

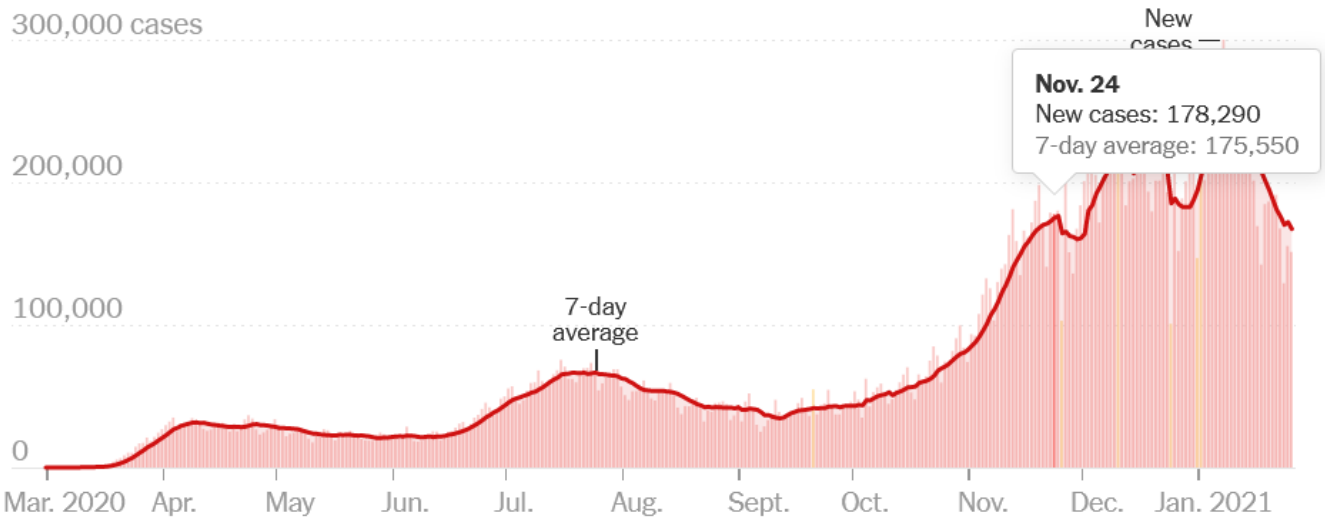
6.6K Retweets **1.8K** Quote Tweets **60K** Likes

The assertion

- Assertion: that people are complaining that scented candles to not have their usual smell
- Implication is that rather than any change in the candles, the customers have lost their sense of smell due to COVID19
  - Are there alternative explanations?
  - Can we test this with data?

Updated January 27, 2021, 12:17 A.M. E.T.

Leer en español



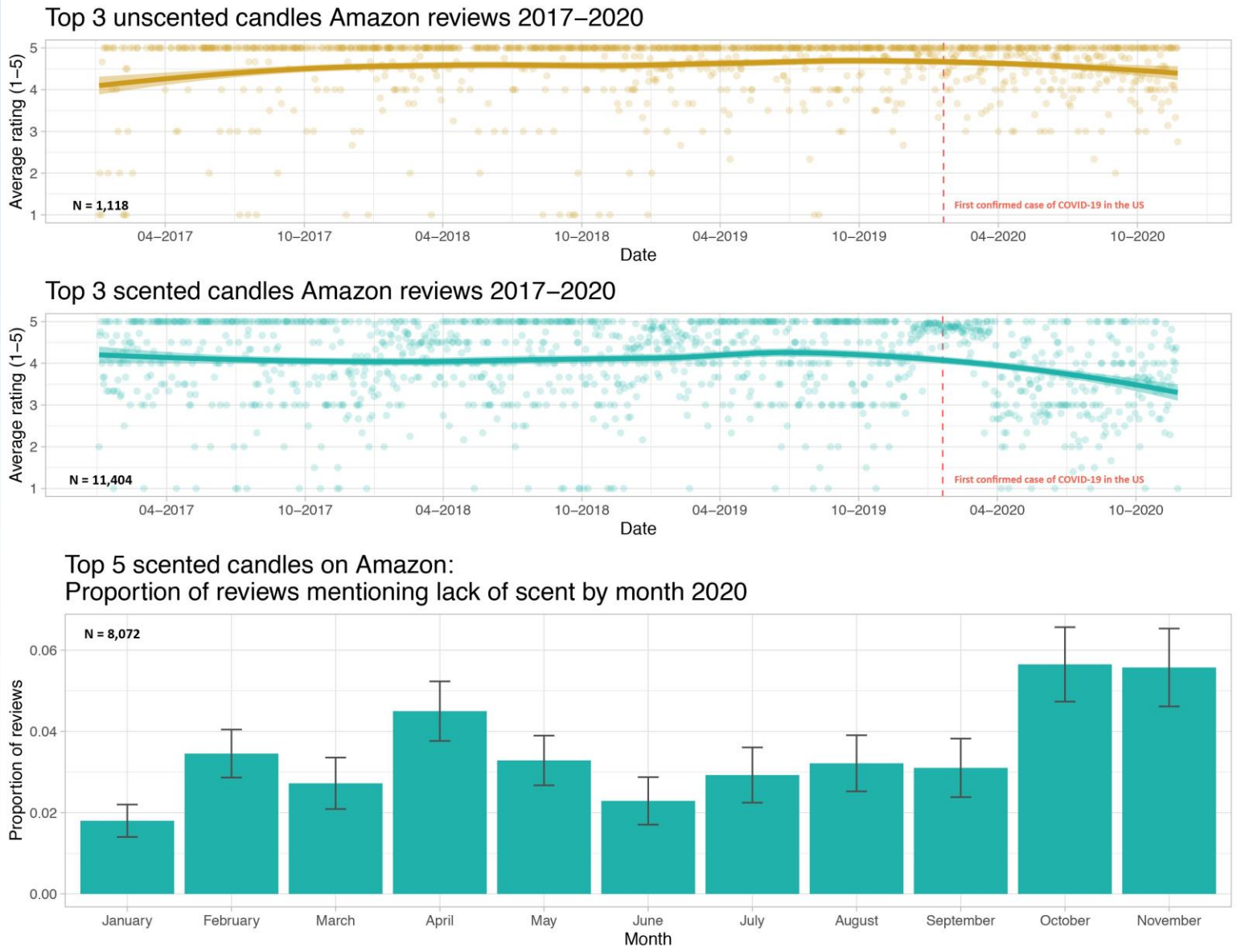
	TOTAL REPORTED	ON JAN. 26	14-DAY CHANGE
Cases	25.4 million+	151,616	-33% ↘
Deaths	425,208	4,205	Flat →
Hospitalized		108,957	-13% →

Day with reporting anomaly. Hospitalization data from the Covid Tracking Project; 14-day change trends use 7-day averages.

Source: https://www.nytimes.com/interactive/2020/us/coronavirus-us-cases.html

The response

- The tweet was widely re-tweeted
- One response was Kate Petrova (@kate_ptrv)
  - "I couldn't just walk past this Tweet, so here is some fun #dataviz Scented candles: An unexpected victim of the COVID-19 pandemic 1/n"
  - The thread is worth reading:
    - https://twitter.com/kate_ptrv/status/1332398737604431874?s=20
  - Alternative:
    - https://threadreaderapp.com/thread/1332398737604431874.html


Top 3 unscented candles Amazon reviews 2017–2020
Average rating (1–5)
N = 1,118
First confirmed case of COVID-19 in the US
04–2017
10–2017
04–2018
10–2018
04–2019
10–2019
04–2020
10–2020
Date
Top 3 scented candles Amazon reviews 2017–2020
Average rating (1–5)
N = 11,404
First confirmed case of COVID-19 in the US
Date
Top 5 scented candles on Amazon:
Proportion of reviews mentioning lack of scent by month 2020
N = 8,072
Proportion of reviews
0.00
0.02
0.04
0.06
January
February
March
April
May
June
July
August
September
October
November
Month

Data and code made available

- Data and code examples
  - https://github.com/kateptrv/Candles
- Chrome plugin to download Amazon reviews
  - URL no longer works!
  - Possible alternatives exist, but I have not checked any

R code

- The R code to create visualisations (using ggplot()) is available
  - We will work through this code over several slides
  - We will look at features of R that we haven't covered yet

```
#### SETUP ####
library(readxl)
library(tidyverse)
library(dplyr)
library(ggplot2)
```

- Setting up
  - Several libraries are to be used, including tidyverse
  - The library 'readxl' is also used; this allows us to read data from Excel files

```
Scented_All <- read_excel("Scented_all.xlsx") # 5 candles
Unscented_All <- read_excel("Unscented_all.xlsx")
```

- Reading data
  - We read in two data files, and assign them to the objects 'Scented_All' and 'Unscented_All'
  - If following along a line at a time, it is important to note that R's working directory must be the one that contains these files
    - Use getwd(),dir(), and setwd() to report the current working directory, list it's contents, and set a new working directory
    - Pressing tab whilst writing a file or directory name will show possible expansions

```
#### SCENTED CANDLES ####
s <- Scented_All %>%
  arrange(Date) %>%
  filter(Date >= "2017-01-01") %>%
  filter(CandleID <= 3) %>%
  group_by(Date) %>%
  summarise(Rating=mean(Rating))
```

- Modifying the data (scented candles)
  - Several things are going on in this command
  - There is a new operator we have not used before
  - There are several functions that we have not used before (but are mostly predictable)
    - We'll look at these over the next few slides

```
#### SCENTED CANDLES ####
s <- Scented_All %>%
  arrange(Date)…
```

- The %>% operator
  - This operator works like a 'pipe' in Unix and other operating systems
  - The output (or content) of one function (or object) is passed as input to the next function
  - Only when we have completed all stages, working from left to right, will we actually assign the results to 's'
  - %>% is provided by the magrittr package, which is part of tidyverse

magrittr

The Treachery of Images, Rene Magritte (1929)

```
#### SCENTED CANDLES ####
s <- Scented_All %>%
  arrange(Date)…
```

- The %>% operator
  - The first two elements take the content of Scented_All, and pass them to the function arrange(), with the parameter 'Date'.
  - From maggritr documentation:
    - "By default the left-hand side (LHS) will be *piped in* as the first argument of the function appearing on the right-hand side (RHS)."
    - So, what we are actually doing is: *`arrange(Scented_All,Date)`*
  - **arrange()** sorts a data frame by the nominated column

```
#### SCENTED CANDLES ####
s <- Scented_All %>%
  arrange(Date) %>%
  filter(Date >= "2017-01-01") …
```

- Next, we filter the data using filter()
  - Like arrange, hopefully this is fairly clear
  - **filter()** filters the data, and retains the rows for which the expression is true

```
#### SCENTED CANDLES ####
s <- Scented_All %>%
  arrange(Date) %>%
  filter(Date >= "2017-01-01") %>%
  filter(CandleID <= 3) …
```

- Filtering the data, continued
  - Here we have another filter() function
  - This is done on the data that have already been filtered in the previous step
  - We could combine expressions in filter, using logical operators (& (and), | (or), ! (not))

```
#### SCENTED CANDLES ####
s <- Scented_All %>%
  arrange(Date) %>%
  filter(Date >= "2017-01-01") %>%
  filter(CandleID <= 3) %>%
  group_by(Date) %>% …
```

- Grouping the data
  - **group_by()** identifies a grouping variable
  - It does not alter the original data (but adds the grouping information to the object metadata)
  - group_by() affects how other commands will respond
  - Here, we are saying that we will group the data by Date

```
#### SCENTED CANDLES ####
s <- Scented_All %>%
  arrange(Date) %>%
  filter(Date >= "2017-01-01") %>%
  filter(CandleID <= 3) %>%
  group_by(Date) %>%
  summarise(Rating=mean(Rating))
```

- Summarising / deriving a new value
  - Finally, we create a new value
  - We do this using **summarise()**
    - This is one of the commands that are affected by group_by()
  - We calculate Rating as the arithmetic mean of individual values of Rating from our large set of data
  - The mean is based on the records in each group
  - We can calculate more than one value here
  - We are not changing our original data Scented_All

Results

- 's' is now a table containing the results of summarise
- It starts

```
> s
# A tibble: 1,354 x 2
   Date                Rating
 * <dttm>               <dbl>
 1 2017-01-01 00:00:00   4
 2 2017-01-02 00:00:00   2.6
 3 2017-01-03 00:00:00   4.75
```

- To check, we can look at the filtered values that contribute to the first line

```
> Scented_All %>%
+     arrange(Date) %>%
+     filter(Date >= "2017-01-01") %>%
+     filter(CandleID <= 3)
# A tibble: 7,005 x 4
   CandleID Date                Rating Review
      <dbl> <dttm>                 <dbl> <chr>
 1        1 2017-01-01 00:00:00      3 "Not much scent~
 2        1 2017-01-01 00:00:00      5 "My mom used to~
 3        1 2017-01-02 00:00:00      1 "Maybe this bat~
```

Plotting the data

- The previous slides looked at how we created 's', a summary of the data for scented candles
  - For each day (the date of the review), a mean rating is calculated
- An equivalent bit of code is used to create 'us', a summary of the unscented candles
- We will now look at how the data are plotted
  - Again, we'll break this down

The first plot

```
s1720 <- ggplot(s, aes(x = (as.Date(Date)), y = Rating)) +
  geom_vline(xintercept = as.numeric(as.Date("2020-01-20")),
colour = "indianred1", linetype = "dashed")+
  geom_smooth(method = "loess", size = 1.5, colour =
"lightseagreen", fill = "lightseagreen") +
  geom_point(alpha = 0.2, colour = "lightseagreen")+
  labs(x = "Date", y = "Average daily rating (1-5)", title =
"Top 3 scented candles Amazon reviews 2017-2020")+
  theme_light()+
  theme(plot.title = element_text(size=16))+
  scale_x_date(date_labels = "%m-%Y", date_breaks = "6
month")
```

The first plot

- This does more that we have done we far with ggplot(), but we'll break it down, and see how it works
- First of all, note that we are assigning the result to an object; we won't see the graph until we explicitly give the name of that object

```
> s1720 <- ggplot(…

> s1720 # to see the result
```

```
s1720 <- ggplot(s, aes(x = (as.Date(Date)), y = Rating)) +
  geom_vline(xintercept = as.numeric(as.Date("2020-01-20")),
colour = "indianred1", linetype = "dashed")+
  geom_smooth(method = "loess", size = 1.5, colour =
"lightseagreen", fill = "lightseagreen") +
  geom_point(alpha = 0.2, colour = "lightseagreen")+
  labs(x = "Date", y = "Average daily rating (1-5)", title =
"Top 3 scented candles Amazon reviews 2017-2020")+
  theme_light()+
  theme(plot.title = element_text(size=16))+
  scale_x_date(date_labels = "%m-%Y", date_breaks = "6
month")
```

- This contains several plotting layers, added using '+'
  - A geom_vline() layer
  - A geom_smooth() layer
  - A geom_point() layer

```
s1720 <- ggplot(s, aes(x = (as.Date(Date)), y = Rating)) +
  geom_vline(xintercept = as.numeric(as.Date("2020-01-20")),
colour = "indianred1", linetype = "dashed")+
  geom_smooth(method = "loess", size = 1.5, colour =
"lightseagreen", fill = "lightseagreen") +
  geom_point(alpha = 0.2, colour = "lightseagreen")+
  labs(x = "Date", y = "Average daily rating (1-5)", title =
"Top 3 scented candles Amazon reviews 2017-2020")+
  theme_light()+
  theme(plot.title = element_text(size=16))+
  scale_x_date(date_labels = "%m-%Y", date_breaks = "6
month")
```

- We also have components that affect the appearance of the whole plot
  - These are labs(),theme_light(), theme() and scale_x_date()

```
ggplot(s, aes(x = (as.Date(Date)), y = Rating)) +
   geom_point(alpha = 0.2, colour = "lightseagreen")+
```

- Let's look at the plotting layers in turn
  - We can make the plot much simpler, by removing all other layers
- Note that all the layers are based on the same input data:

```
(x = (as.Date(Date)), y = Rating))
```

  - The y variable is our mean Rating
  - The x variable is Date. We use as.Date to force it into a 'Date' datatype

```
ggplot(s, aes(x = (as.Date(Date)), y = Rating)) +
   geom_point(alpha = 0.2, colour = "lightseagreen")
```

- We used geom_point() last week
  - It draws a set of points on to the graph
- The option 'colour=' is used in combination with one of the colour names that R recognises
  - Note that R accepts both 'color' and 'colour' as arguments
- We have not used the alpha setting before.
  - It is a transparency value
    - 0 = transparent; 1 = opaque
  - This makes it more obvious where we have few or many markers

```
ggplot(s, aes(x = (as.Date(Date)), y = Rating)) +
 geom_vline(xintercept = as.numeric(as.Date("2020-01-20")),
colour = "indianred1", linetype = "dashed")
```

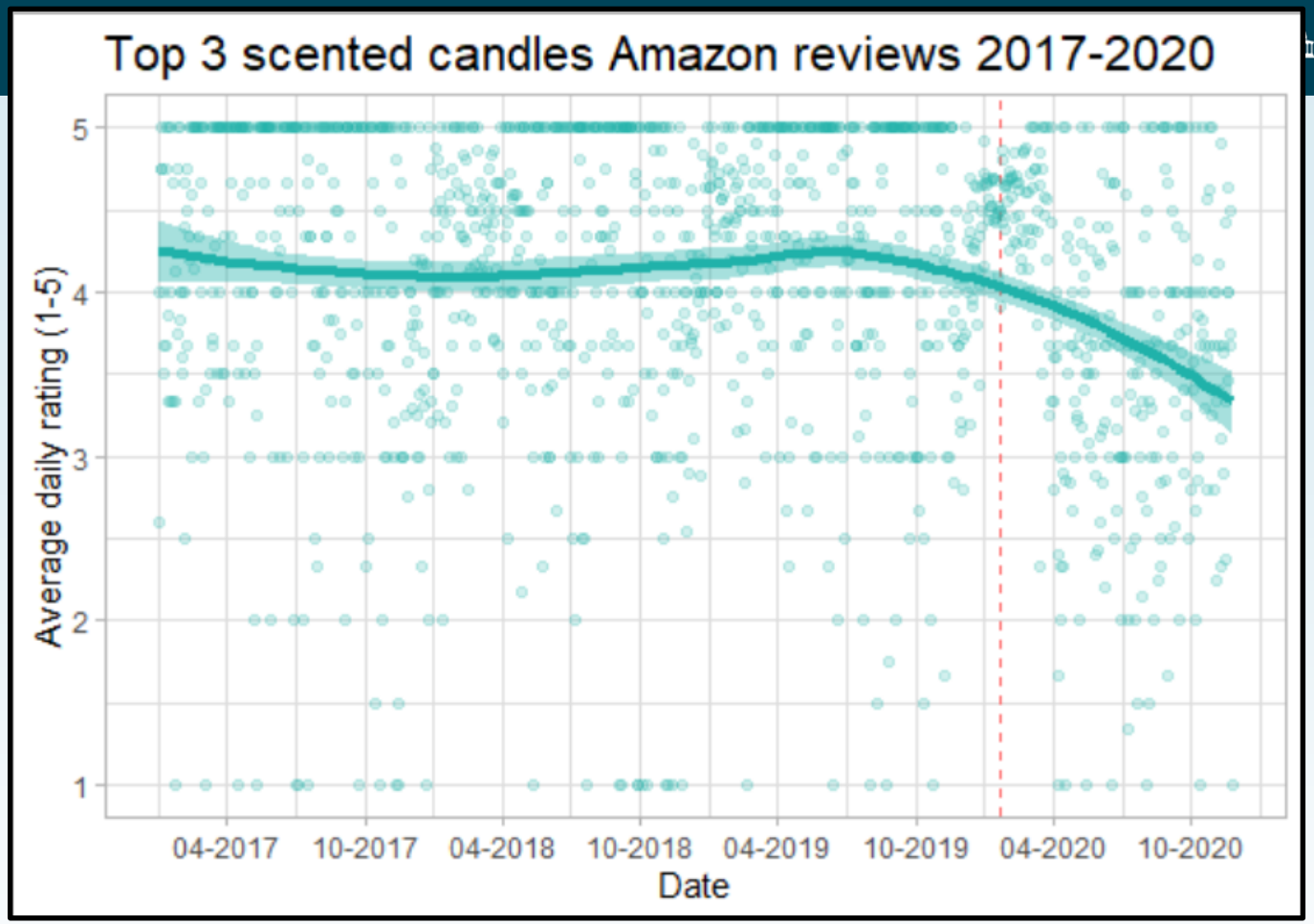
- We haven't used geom_vline() before
  - This draws a vertical line on our graph
  - The location of the line is set by xintercept
    - Note that we have to go through a slightly verbose process in this example of converting a string "2020-01-20" to a Date, and then to a numeric
  - The other options are colour and linetype
  - There is an equivalent geom_hline() function to draw horizontal lines, and a geog_abline() to draw lines of the form $y=ax+b$

```
ggplot(s, aes(x = (as.Date(Date)), y = Rating)) +
 geom_smooth(method = "loess", size = 1.5, colour =
"lightseagreen", fill = "lightseagreen")
```

- The last layer we'll look at is geom_smooth()
  - It draws a smoothed line through a set of points together with a shaded area
  - As the name suggests, this is a smoothed version of the data
  - The line appearance is regulated by size, colour and fill, which should be easy to understand
  - Different methods are available, which apply different statistical models
    - See: https://ggplot2.tidyverse.org/reference/geom_smooth.html

```
s1720 <- ggplot(s, aes(x = (as.Date(Date)), y = Rating)) +
...
 labs(x = "Date", y = "Average daily rating (1-5)", title =
"Top 3 scented candles Amazon reviews 2017-2020")+
  theme_light()+
  theme(plot.title = element_text(size=16))+
  scale_x_date(date_labels = "%m-%Y", date_breaks = "6
month")
```

- The last four lines affect the appearance of the graph
  - labs() allows us to set labels for the axes and the main title
  - theme_light() indicates that we want to use one of the built-in themes
    see: https://ggplot2.tidyverse.org/reference/ggtheme.html
  - theme() then overrides that with some bespoke settings, in our case the text size for the title
  - scale_x_date() similarly allows us to override some of the built-in defaults for scales that use a date:
    see: https://ggplot2.tidyverse.org/reference/scale_date.html



- The final graph
  - A points layer
  - A smoothed line through the data
  - A vertical line

Continuing with the scented candles example

- The scented candles script continues with another scatterplot, which generates a very similar graph using unscented candles as the source for the review data
- These scatterplots are both based on Rating
- The script then adds a bar chart, which is based on text within the review

```
#### NO SCENT FUNCTION ####
no_scent <- function(x){
  case_when(
    str_detect(x, "[Nn]o scent") ~ "1",
    str_detect(x, "[Nn]o smell") ~ "1",
    ...
    str_detect(x, "[Ll]ike nothing") ~ "1",
    TRUE ~ x
  )

}
```

- Before creating the bar chart, a function is created
- See how we assign the function to 'no_scent'
- function(x) means that we are going to pass it an argument

```
#### NO SCENT FUNCTION ####
no_scent <- function(x){
  case_when(
    str_detect(x, "[Nn]o scent") ~ "1",
    str_detect(x, "[Nn]o smell") ~ "1",
    ...
    str_detect(x, "[Ll]ike nothing") ~ "1",
    TRUE ~ x
  )

}
```

- case_when is similar to the switch structure in many programming languages
- case_when takes a series of expressions that will evaluate to TRUE or FALSE
- If they are TRUE, then they return the value given after '~', and processing stops

```
#### NO SCENT FUNCTION ####
no_scent <- function(x){
  case_when(
    str_detect(x, "[Nn]o scent") ~ "1",
    str_detect(x, "[Nn]o smell") ~ "1",
   ...
    str_detect(x, "[Ll]ike nothing") ~ "1",
    TRUE ~ x
  )

}
```

- Ending with 'TRUE ~ x' means that if we do not get a match in any of the previous tests, we will always match 'TRUE', (and then return 'x', the original value)
- The examples here look for various phrases as regular expression patterns

```
#### NO SCENT FUNCTION ####
no_scent <- function(x){
  case_when(
    str_detect(x, "[Nn]o scent") ~ "1",
    str_detect(x, "[Nn]o smell") ~ "1",
    ...
    str_detect(x, "[Ll]ike nothing") ~ "1",
    TRUE ~ x
  )

}
```

- Thus, this would return '1' for a review that contains the words "no scent"

```
s5 <- Scented_All %>%
  arrange(Date) %>%
  filter(Date >= "2020-01-01") %>%
  mutate(noscent = no_scent(Review)) %>%
  mutate(noscent = ifelse(noscent != 1, 0, 1)) %>%
  mutate(month = reorder(format(Date, '%B'), Date)) %>%
  group_by(month) %>%
  add_tally() %>%
  summarise(n =n, noscent = sum(noscent)) %>%
  mutate(nsprop = noscent/n) %>%
  mutate(se = sqrt((nsprop*(1-nsprop))/n)) %>%
  summarise(n=mean(n), se=mean(se), nsprop=mean(nsprop))
```

- The no_scent() function is used here, where we create the data frame s5
- Again, we'll go through this

```
s5 <- Scented_All %>%
  arrange(Date) %>%
  filter(Date >= "2020-01-01") %>%
  mutate(noscent = no_scent(Review)) %>%
  mutate(noscent = ifelse(noscent != 1, 0, 1)) %>%
  mutate(month = reorder(format(Date, '%B'), Date)) %>%
  group_by(month) %>%
  add_tally() %>%
  summarise(n =n, noscent = sum(noscent)) %>%
  mutate(nsprop = noscent/n) %>%
  mutate(se = sqrt((nsprop*(1-nsprop))/n)) %>%
  summarise(n=mean(n), se=mean(se), nsprop=mean(nsprop))
```

- arrange() and filter() are used in a similar way to before
- mutate() is a new function – it creates values in a new column, or modifies existing values
- Note that modifications are applied in the working data that will be assigned to s5

```
s5 <- Scented_All %>%
  arrange(Date) %>%
  filter(Date >= "2020-01-01") %>%
  mutate(noscent = no_scent(Review)) %>%
  mutate(noscent = ifelse(noscent != 1, 0, 1)) %>%
  mutate(month = reorder(format(Date, '%B'), Date)) %>%
  group_by(month) %>%
  add_tally() %>%
  summarise(n =n, noscent = sum(noscent)) %>%
  mutate(nsprop = noscent/n) %>%
  mutate(se = sqrt((nsprop*(1-nsprop))/n)) %>%
  summarise(n=mean(n), se=mean(se), nsprop=mean(nsprop))
```

- Firstly, we use mutate() to create a new value, noscent, that is the output from our no_scent() function
- Secondly, we re-code the new noscent into a consistent 0/1 binomial variable

```
s5 <- Scented_All %>%
  arrange(Date) %>%
  filter(Date >= "2020-01-01") %>%
  mutate(noscent = no_scent(Review)) %>%
  mutate(noscent = ifelse(noscent != 1, 0, 1)) %>%
  mutate(month = reorder(format(Date, '%B'), Date)) %>%
  group_by(month) %>%
  add_tally() %>%
  summarise(n =n, noscent = sum(noscent)) %>%
  mutate(nsprop = noscent/n) %>%
  mutate(se = sqrt((nsprop*(1-nsprop))/n)) %>%
  summarise(n=mean(n), se=mean(se), nsprop=mean(nsprop))
```

- month is reordered (i.e. so it is not treated alphabetically)
- We then identify that we will group by month
- add_tally() adds a count to each record in a column labelled *n*

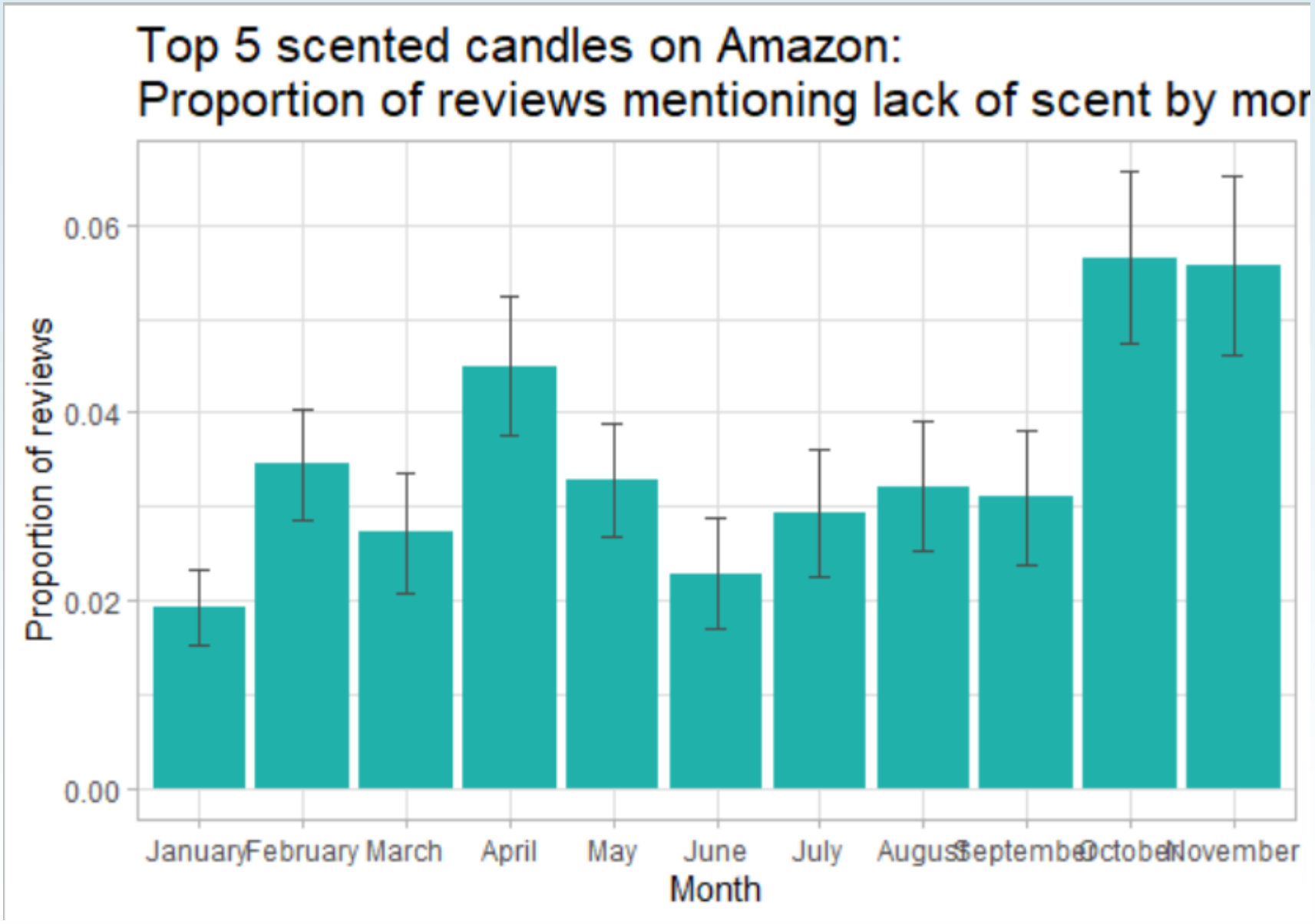
```
s5 <- Scented_All %>%
  arrange(Date) %>%
  filter(Date >= "2020-01-01") %>%
  mutate(noscent = no_scent(Review)) %>%
  mutate(noscent = ifelse(noscent != 1, 0, 1)) %>%
  mutate(month = reorder(format(Date, '%B'), Date)) %>%
  group_by(month) %>%
  add_tally() %>%
  summarise(n =n, noscent = sum(noscent)) %>%
  mutate(nsprop = noscent/n) %>%
  mutate(se = sqrt((nsprop*(1-nsprop))/n)) %>%
  summarise(n=mean(n), se=mean(se), nsprop=mean(nsprop))
```

- We then go through a two stage process with summarise()
  - Count the total number and total number with noscent
  - Then calculate the proportion (for each month) with no scent, and the standard error

Creating a bar chart

```
s5r <- ggplot(s5, aes(x=as.factor(month), y = nsprop, group
= month))+
  geom_bar(stat = "identity", fill = "lightseagreen")+
  geom_errorbar(aes(ymin = (nsprop-se), ymax = (nsprop+se)),
width=0.2, colour = "gray30")+
  labs(x = "Month", y = "Proportion of reviews", title =
"Top 5 scented candles on Amazon: \nProportion of reviews
mentioning lack of scent by month 2020")+
  theme_light()+
  theme(plot.title = element_text(size=16))
```

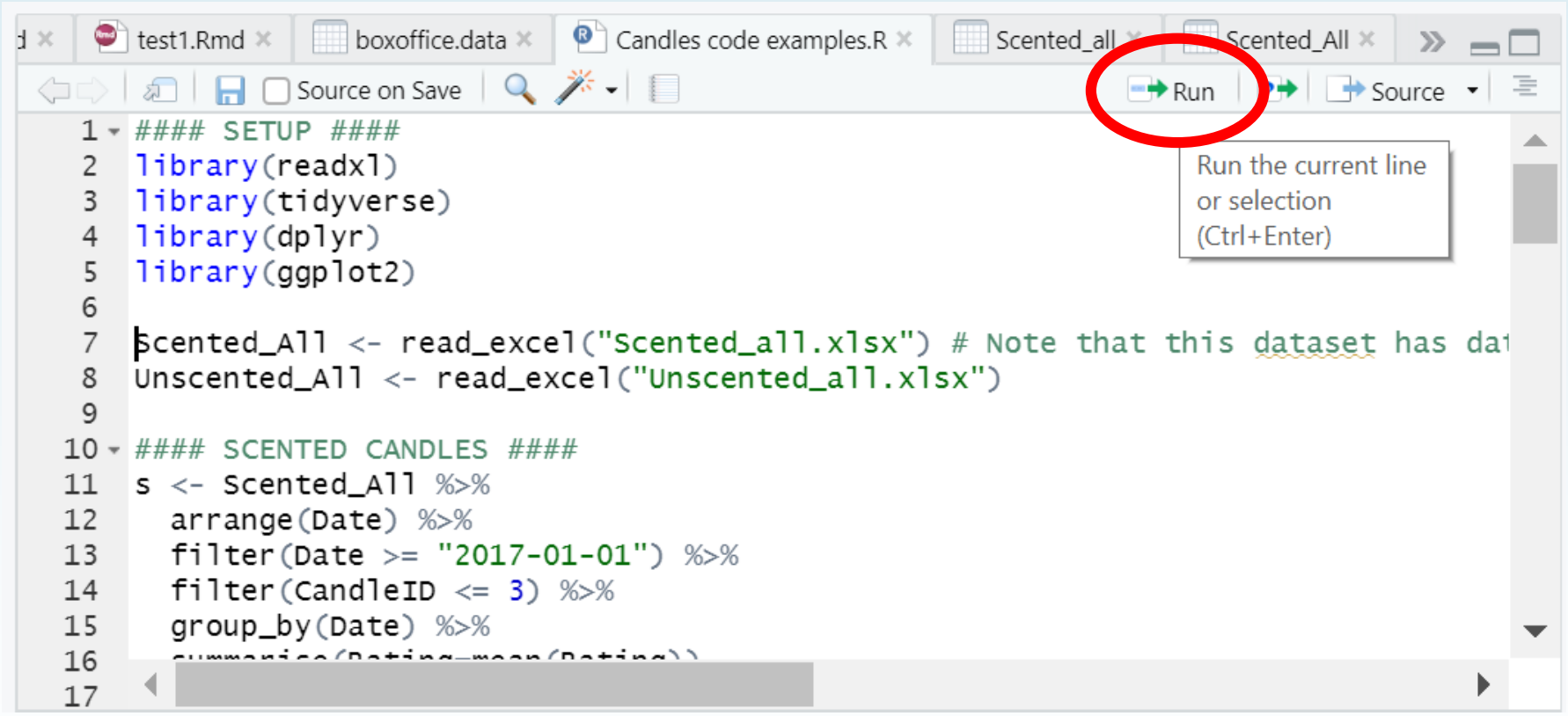
- Finally, we generate another chart
  - This has two layers – a bar chart, and a set of error bars drawn over those bars
    - We used our calculated standard error values to draw these
  - Note that month is being used as a variable with factors


Top 5 scented candles on Amazon:
Proportion of reviews mentioning lack of scent by mor
Proportion of reviews
0.06
0.04
0.02
0.00
January
February
March
April
May
June
July
August
September
October
November
Month

Storing commands

- So far, we have proceeded by typing commands directly at the console
  - This is not ideal, and especially not in the case of long multi-line commands
- There are a number of approaches
  - We can write a series of commands in a file, and 'play' some or all of those commands
- We can put together commands, text, and output in an RMarkdown file

Using a .R file



- This is simply a text file with a .R file extension

RMarkdown

- RMarkdown is a variant of markdown
- Markdown is a lightweight markup language, written in a plain text file

```
Heading
=======

Regular text, with a [link] (http://example.com)

Sub-heading
-----------

1. A numbered
2. List
```

RMarkdown

- RMarkdown simply adds some extra components
- Information
  - See R for Data Science
    - Printed book: chapter 21
    - Online version: chapter 27
  - See Cheatsheet
    - RStudio: Help -> Cheatsheets - > RMarkdown Cheatsheet
    - Online: https://raw.githubusercontent.com/rstudio/cheatsheets/master/rmarkdown-2.0.pdf

How to start

- In RStudio
  - New file -> RMarkdown…
  - This should be edited and saved
- In a text editor
  - Create a file with a basic template

```
---
title: "Untitled"
author: "o.duke-williams@ucl.ac.uk"
date: "28/01/2021"
output: html_document
---

```{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>.
...
```

Header section
RStudio fills in the initial details
Default output is html

Default options
Leave as is

This is regular text that gets copied to output

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}
summary(cars)
```

This is a chunk of r code between ```
The opening tag has {r *chunkname*}

Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

Output

- Important – you should save this as an .Rmd file before proceeding
- In RStudio click 'Knit'
  - This will save to the nominated output type
  - It can be over-ridden and knitted to another form
  - Knitting to PDF may require additional software

Untitled

o.duke-williams@ucl.ac.uk

28/01/2021

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.

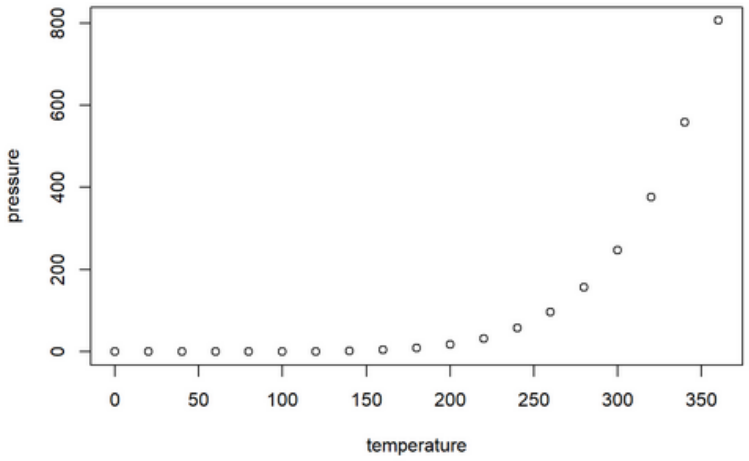
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

Copied text

An echoed R command

R output

Plot output

Note that the image data is embedded within the document