MXB262 Practical 1: Introduction to R, ggplot2, and Rmarkdown

MXB262 QUT

2020

This Week

This week you will learn how to:

0. download R, RStudio, and start writing some code
1. use Rmarkdown to make nice and accessible documents with text, code chunks, and figures

2. set the working directory

3. create a simple plot using `plot()`
4. load R packages
5. use `ggplot()` to visualise data

0. Obtaining R

R can be downloaded from http://cran.r-project.org/ Using a script editor, such as "RStudio," can also be helpful. RStudio can be downloaded from http://www.rstudio.com/

Starting RStudio

Click the RStudio icon to open RStudio. The interface is divided into several panels (clockwise From top left):

1. The source code (supporting tabs)
2. The currently active objects/history
3. A File browser/plot window/package install window/R help window (tabbed)
4. The R console The source code editor (top left) is where you type, edit and save your R code.

The source code editor (top left) is where you type, edit and save your R code. The editor supports text highlighting, autocompletes common functions and parentheses, and allows the user to select R code and run it in the R console (bottom right) with a keyboard shortcut (Ctrl R on windows, command-enter on macs). Code will appear in this font:

```
plot(x,y)
```

Starting a new script

Let's open a new script and save it to your harddrive. In matlab, this is the equivalent to an 'm file'. Instead of typing every command into the console, making a script lets you record and save everything that you do. So next week if you're wondering how you made that line plot, or did that calculation, or ran that multi-line simulation, you can just pull up the R script and run it exactly the same. Remember that #dataviz is about reproducibility – using scripts in R is one step toward that goal.

When writing R scripts, use lots of # comments throughout your R scripts to record what you were thinking or what the code does. Any line or text that starts with # won't run, it's just there to communicate with whoever reads your code later. Usually that is you, but sometimes it will be your tutors or your collaborators. If you liberally use comments you will thank yourself when you go back to the analysis later!

Similarly at the start of your script, put some meta-information about the whole script, such as: # who wrote the code? # what does the code do? # when did you write the code? etc.

The more projects you work on and analyses you do, the more important it is to have this meta- information at the beginning of your code. Every academic and data scientist wishes they were just a little bit better at doing this, so start now.

---

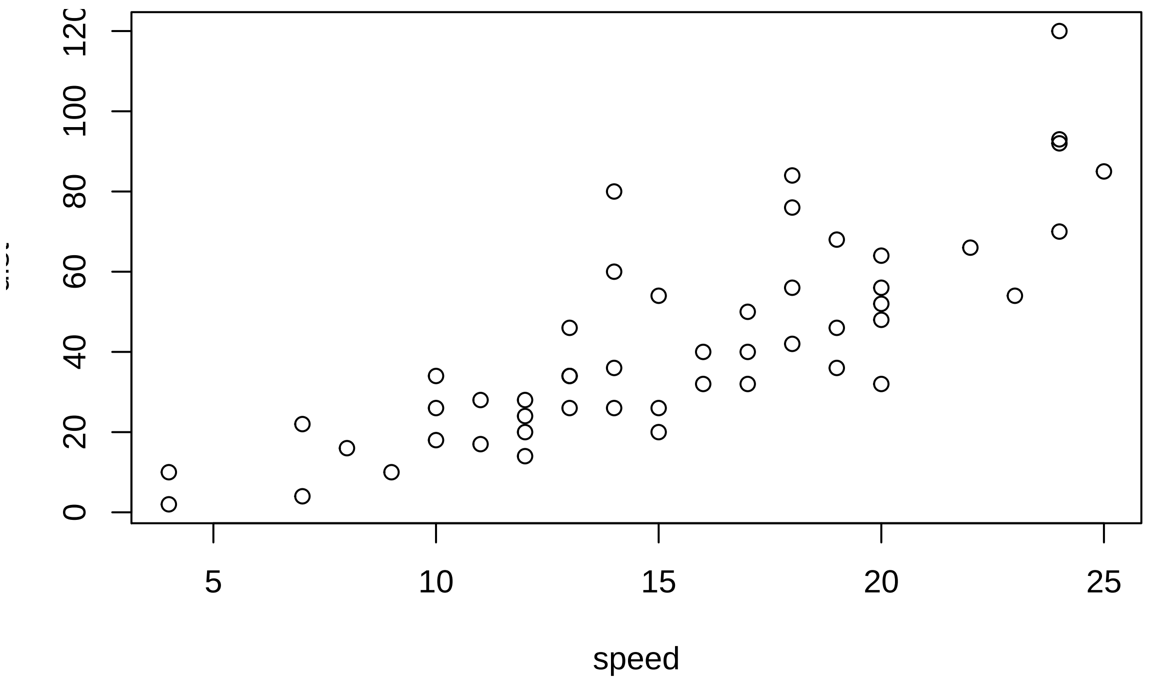
1. R Markdown

This is an R Markdown document. In MXB262 all assignments and worksheets will be completed in this format. 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 (at the top of the Editor window in RStudio – it helpfully has a little ball of wool and knitting needles next to it), a document will be generated that includes the content, figures, and the output of any embedded R code chunks within the document. This is useful because then everything is in one place – you only need to make one document with your text, code, and figures, not create a Word doc and continually copy updated code, results, and figures into it.

The real magic of an R Markdown script is that you can embed an R code chunk like this:

```
plot(cars)
```



In the knitted pdf, you'll see the code ("'{'plot(cars)'}), plus the output from the code (the figure). In the source code (i.e. the script), you'll just see the code, but when you press the green 'play' button next to it the figure will show up.

Creating headers, tables, lists and indentations is easy, you can find a cheat sheet here https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf. Save the cheat sheet to your desktop or print it out so you have it on hand for the workshops (or do what I do, and google it every single time).

You're going to be using R Markdown for all worksheets, Problem Solving tasks 1-6, and the summaries of your projects. If you are having troubles with R Markdown at any point, please ask your tutor in practicals or via email, or you won't be able to submit your assessment.

TASK

1. Update the author (your name) and date fields at the top of this file.
2. Use the knit button to render the file to pdf.
3. Run the code in only the individual chunk above, instead of the full document, by clicking the *Run* button inside the chunk (the green arrow to the right of the "“‘{r cars}" line). Or, run each line of code one by one in the console by placing your cursor inside the chunk and pressing *Cmd+Shift+Enter*.

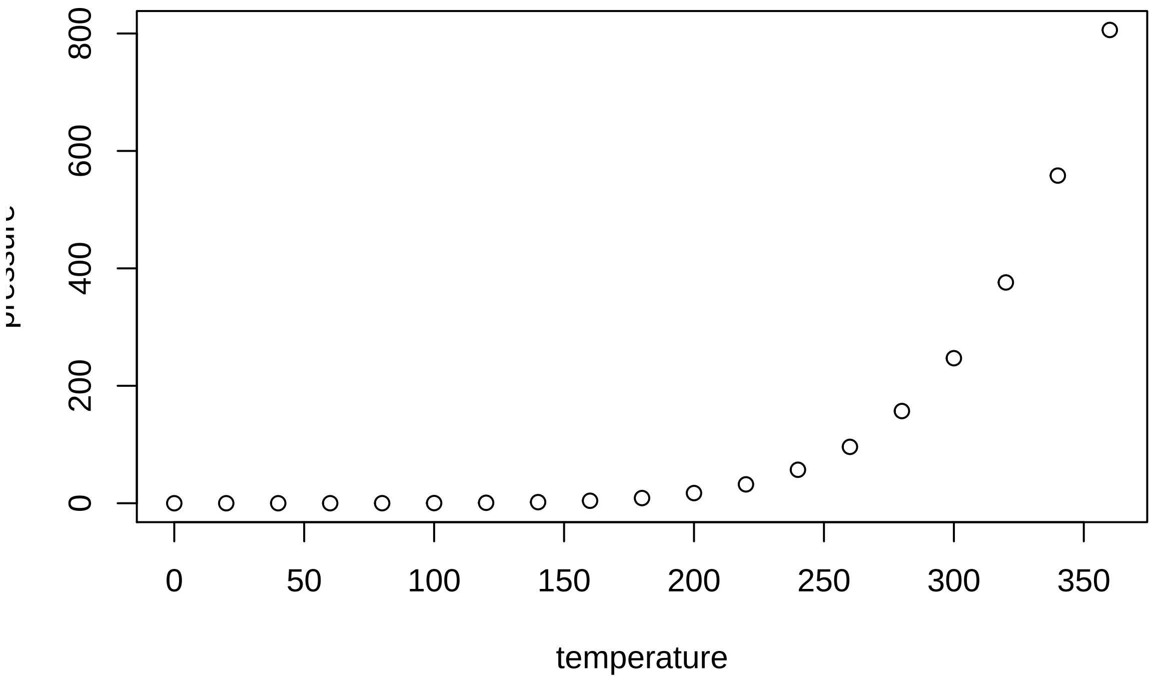
---

2. Adding code chunks

TASK

4. Add a new code chunk by clicking the *Insert R* button on the toolbar or by pressing *Cmd+Option+I* for mac or *ctrl+alt+I* for pc.
5. Within that code chunk add a new plot using the `plot()` function, and the dataset `discoveries`

Note: the `echo = FALSE` parameter can be added to the code chunk to prevent the R code that generated the plot from being printed. Like this:



When you pressed knit, you only saw the figure – not the code that generated it (look back up in the html file to the cars plot – you can see the code was printed for that one).

---

3. Setting the working directory

To use R you will to access data, save files, and sometimes access images. The folder that stores all documents for a project is called the `working directory`. R needs to know where to look for data files and/or where to save generated files, that is, where the `working directory` for this project is. Follow the following steps to set-up the working directory for MXB262.

1. In the console use the `setwd("add file path here")` function to set the working directory to a file of your choice. You must keep the " " marks inside the brackets.
2. Type the `getwd()` function in the console to check that returned file path is correct.

NOTE: Please do not submit any code that contains the `setwd()` function. Please either just set it on your computer each time you open R Studio, or comment the line of code out by placing a # in front of the code. If you include it, when the tutor is trying to mark your code it will try to reorganise the files on the tutor's computer. This can make marking very difficult!

---

4. Data

This unit is all about data visualisation, which means we need data. To make this easy to start with, we will be using data that is already clean, ready to use and available within R. See package for a list of all datasets already loaded into R.

TASK

5. Create a new chunk and add a plot of one of the datasets listed on the (R data packages webpage. Use the `plot()` function that we already used in the code chunk above.

---

5. Loading packages

Packages in R can contain functions as well as data. R comes with some base packages when you first install it, but if you wish to use additional packages you need to *install* and then *load* them.

To install a package use the `install.package("package_name")` function (do this on every new computer), and to load the package use `library(package_name)` (do this every time you re-open R).

Use the code chunk below to install and load the *ggplot2* library. Note that the first line is commented out using `#` to stop the `install.packages()` function being run everytime the document is rendered. You will need to delete the `#` and run both lines. I suggest you add the `#` back in once you have successfully installed ggplot2.

```r
#install.packages("ggplot2")
library(ggplot2)
```

---

6. Visualisation in R - ggplot()

Throughout these practicals we'll be using Hadley Wickham's `Grammar of Graphics` package (more commonly called `ggplot2`). You have installed this in the code chunk above. ggplot2 has a specific format for creating plots, graphs and other visualisations. Once you understand the format, making and customising graphics is intuitive.

You can think of building a plot like building a house, the essential ingredients are: the building materials (your data) and a floor plan (called the geom in ggplot). After the house is built you can add fittings and fixtures like taps, door handles, and the house number (title, labels, change the colours).

house	plot	ggplot function
Building Materials	Your data set and which variables in the dataset to use	`ggplot(dataset, aes(x = variable, y = variable)`
Floor plan	what type of `geom` (graph or plot)	`+ geom_point()`

house	plot	ggplot function
Fittings and fixtures	add a title, change axis labels, add colours etc	`+ ggtitle() + ylab(), + xlab()` etc
Add it all together		`ggplot(dataset, aes(x = variable, y = variable) + geom_bar() + ggtitle()`

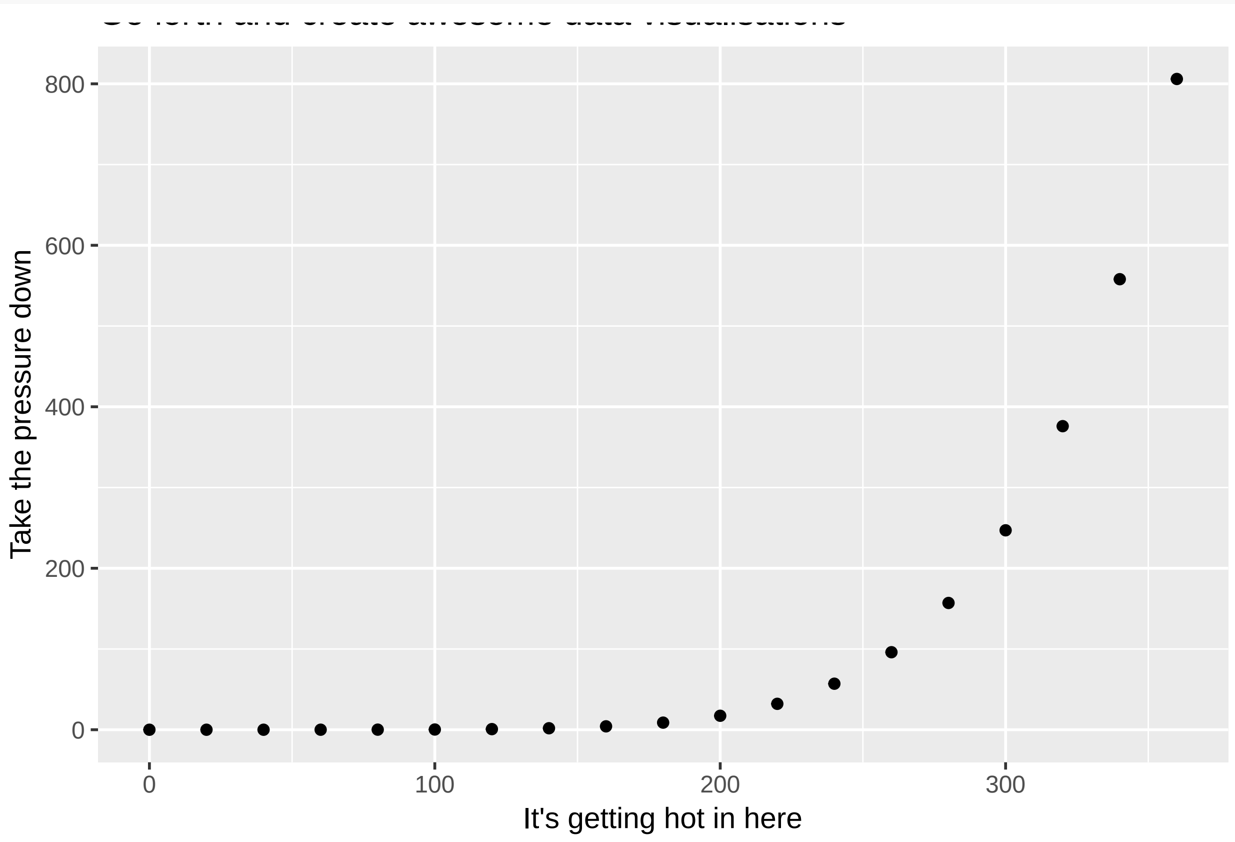
The following code chunk re-creates the pressure plot above but using ggplot.

```r
# 1. Load ggplot2 library/package
library(ggplot2)

# 2. The building materials are: ggplot(pressure, aes(x= temperature, y = pressure))
# 3. Define the floor/graph plan (tell ggplot which type of graph to make) : + geom_point()
# 4. Add titles and axis labels: ggtitle("Go forth and create awesome data visualisations") + ylab("Tak
# 5. Assign the plot to an object called 'scatterplot_house'

scatterplot_house <- ggplot(pressure, aes(x= temperature, y = pressure)) + geom_point() + ggtitle("Go fo

#print the plot
scatterplot_house
```



A cheat sheet for ggplot can be found https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf.

We will be using Hadley Wickham's free online book R for Data Science to learn more about ggplot. You can start at Chapter 3 Data Visualisation, however the introductory chapters may also be useful if you are interested in starting there outside of class time.

TASKS

6. Work through Chapter 3 of R for Data Science. Read and follow all the examples along in your own R console for sections:

- 3.1
- 3.2 (including all exercises)
- 3.3 (and exercises 1-3)
- 3.4
- 3.5 (skip these exercises in class)
- 3.6 (skip these exercises).

You can go back to the exercises you skipped outside of class time, although they aren't critical at this stage of the unit.

7. Work through (i.e. use the code to generate for yourself) the first 3 plots in Donald Trump's twitter example that we talked about in the lecture. Although all of the code here is useful, pay particular attention to the ggplot commands being used here. These are all simple, but well-chosen and effective plots. NOTE: you will need to install the twitteR package first. You won't be able to set up the Twitter authentication, so ignore the second code chunks on the website – start by downloading the dataset by running the third code chunk. Once they start getting into the word analysis, they stop walking us through the visualisations. Code for those figures follows here, so keep following and running all of the analysis and data-wrangling on the blog but jump back to grab this code when you reach each of these questions and figures:

What were the most common words in Trump's tweets overall?

```r
tweet_words %>%
count(word, sort = TRUE) %>%
head(20) %>%
mutate(word = reorder(word, n)) %>%
ggplot(aes(word, n)) +
geom_bar(stat = "identity") +
ylab("Occurrences") +
coord_flip()
```

Which are the words most likely to be from Android and most likely from iPhone?

```r
android_iphone_ratios %>%
group_by(logratio > 0) %>%
top_n(15, abs(logratio)) %>%
ungroup() %>%
mutate(word = reorder(word, logratio)) %>%
ggplot(aes(word, logratio, fill = logratio < 0)) +
geom_bar(stat = "identity") +
coord_flip() +
ylab("Android / iPhone log ratio") +
scale_fill_manual(name = "", labels = c("Android", "iPhone"),
values = c("red", "lightblue"))
```

And we can visualize it with a 95% confidence interval:

```r
library(scales)
sentiment_differences %>%
ungroup() %>%
mutate(sentiment = reorder(sentiment, estimate)) %>%
mutate_each(funs(. - 1), estimate, conf.low, conf.high) %>%
ggplot(aes(estimate, sentiment)) +
geom_point() +
geom_errorbarh(aes(xmin = conf.low, xmax = conf.high)) +
scale_x_continuous(labels = percent_format()) +
labs(x = "% increase in Android relative to iPhone",
y = "Sentiment")
```

We're especially interested in which words drove this different in sentiment. Let's consider the words with the largest changes within each category:

```r
android_iphone_ratios %>%
inner_join(nrc, by = "word") %>%
filter(!sentiment %in% c("positive", "negative")) %>%
mutate(sentiment = reorder(sentiment, -logratio),
word = reorder(word, -logratio)) %>%
group_by(sentiment) %>%
top_n(10, abs(logratio)) %>%
ungroup() %>%
ggplot(aes(word, logratio, fill = logratio < 0)) +
facet_wrap(~ sentiment, scales = "free", nrow = 2) +
geom_bar(stat = "identity") +
theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
labs(x = "", y = "Android / iPhone log ratio") +
scale_fill_manual(name = "", labels = c("Android", "iPhone"),
values = c("red", "lightblue"))
```