F28ED: Stats Lab 1 | Introduction to R Studio for Data Analysis

Dr Peter McKenna

02 November 2020

Lab Housekeeping

Create installation script

- In R Studio we write scripts to perform a set of commands
- To add a script click the Script icon below the File tab (top left)

Working with scripts

- Scripts are edited in the *Workspace Tab* - top left window
- Create script headings using `#` (called a "comment" or inactive code)
- **To execute code press `Ctrl + Enter`**
- This can be done for single lines, or multiple lines by highlighting sections

Housekeeping script

- Here is the code for installing and loading the Tidyverse packages
- Be wary of the use of apostrophes around the package name

Starting your own script: Live coding demo

```
# Housekeeping

# install packages

install.packages('readr')   # installs the 'readr' package
install.packages('tidyr')   # installs the 'tidyr' package
install.packages('dplyr')   # installs the 'dplyr' package
install.packages('ggplot2') # installs the 'ggplot2' package
```

```
# load packages

library(readr)   # load 'readr' package
library(tidyr)   # load 'tidyr' package
library(dplyr)   # load 'dplyr' package
library(ggplot2) # load 'ggplot2' package

# Select all of the above text and run using`Ctrl + Enter`
```

- Package installation takes a minute; R will tell you when installation is complete in the Console panel
- Do not be concerned by the text generated in the R console; R packages are stamped to certain versions of R Studio, but work fine once loaded
- Save the script as `Housekeeping.R` when you're finished
- Ok, now everything is installed and loaded we can move on

Introduction

- In this introductory class you are going to learn how to
  - read a .csv data file into R Studio
  - wrangle and explore the data using the *Tidyverse* functions
  - explore the data's distribution
  - generate plots

Considerations

- Today we are skimming the surface of R Studio's functionality
- Programming in R has a steep learning curve relative to other high level stats software (e.g., SPSS)
- You are going to make mistakes - a single symbol out of place will cause errors
- Take time to carefully examine your code

Tidyverse

- Today we are using *Tidyverse* methods to explore and wrangle the data
- The Tidyverse is a set of R packages that allows for more user-friendly programming, relative to what's called *base R*
- For more about the Tidyverse see Hadley Wickham's free online text R for Data Science
- We are covering content from R for Data Science chapters 1,2,3 & 11.

Tidyverse packages

- `readr` : for reading in data
- `tidyr` : for tidying & wrangling data
- `dplyr` : for tidying & wrangling data
- `ggplot2`: for plotting data

Main verbs of R's Tidyverse

The aforementioned packages allow us to use the following Tidyverse verbs:

- `filter` : extract rows
- `select` : extract columns
- `pivot_wider` : spread rows to columns
- `pivot_longer` : gather columns into rows
- `mutate` : compute and append new/existing columns
- `summarise` : summarise data based on stated criteria
  - We will learn more about these during Lab 2

The Pipe operator

- You are going to see a lot of this symbol

```
%>%
```

- This is the *pipe* operator
- Do not fear the pipe operator
- It means "then do this"

Starting your own script: Live coding demo

- Create a new script and call it `fed28_lab_1`

```
# my first bit of R code
x <- 2
y <- 3

x*y

# area rectangle
x <- 43
y <- 62

area_rect <- x*y

area_rect

# fun with functions
sqrt(4)

a <- 64

sqrt(a)
```

Reading in the data

- Start by creating a directory for the raw data, out put data, and figures

---

Create a directory and read in data: Live coding demo

- A good way to keep your data and plots organised in R Studio is to create a set of local directories

```
# create new directory folders
dir.create("data")
dir.create("fig_output")
```

- We then download the data into our new data directory by specifying both the file and directory name
- Note: both the URL and file name are surrounded by speech marks
- Note 2: you specify the R Studio folder in the text before the file name; e.g., "folder/file.csv"

```
download.file("https://vision.hw.ac.uk/webapps/blackboard/content/listContent.jsp?course_id=_106149_1&c
```

Create your first data object (or tibble)

Create your first tibble: Live coding demo

```
data <-                         # assign ( <- ) the label "data" to next set of commands
  read_csv("f28ed_data.csv") # read in csv file called "data-1.csv"
```

- This syntax creates a data object (or tibble) called `data` from the "f28ed_data.csv" data file in our directory
- We could assign any label to this new tibble
- Rule of thumb: simple, meaningful labels work best

Vector (or column) descriptions

- `id` = participant number
- `like` = rating of assistants likability from 1 = "Did not like at all" to 5 = "Liked it a lot"
- `voice_type` = speaker voice type including two levels: Female; Male.
- `conv_len` = duration of the conversation between the user and the smart speaker measured in sec

Details about the data

Examine the data: Live coding demo

```
str(data)     # view the structure of the dataset

glimpse(data) # information dense summary
```

- The data details 30 participants interactions with a smart speaker in two different conditions: Female voice speaker; Male voice speaker
- The data is in the *long format* where each row represents a unique observation
  - This is the preferred format for Tidyverse wrangling
- The experiment is *within-subjects* or *paired samples* - the two are used synonymously which can get confusing so note this down
- *Within-subjects*: participants complete each experimental condition (i.e., they interacted with both the female and male voice speaker)
- This is different to *between-subjects* where separate groups of participants complete each experimental condition
- Between-subjects is also referred to as *independent samples*; I've got you covered here, don't worry

Inspecting the data

- Let's go through each of the commands below
- Checking data with these commands helps you to get a better understanding of the data
- `head` & `tail` for example help you quickly determine if the dataset has been read in properly from top to bottom

```
head(data)    # shows top 6 rows

tail(data)    # shows bottom 6 rows

dim(data)     # number of rows and columns

names(data)   # list vector names

# you can use square brackets to give an index to certain elements of the data

names(data)[2] # give the name of column 2

View(data)    # opens whole dataset in a new tab; note the capital 'V'
```

- Output is generated in the Console panel (bottom left)
- Take your time to go through these commands and make sense of their output

Index and subset tibbles

- You can also use square brackets to index your data frame

```
data[1,4] # value from the first row in the fourth column

data[2,2] # value from the second row in the second column
```

Exercise 1

1. How many groups would a between-subjects version of this experiment have?
2. What are some of the practical challenges of running a within-subjects experiment?
3. Can you think of any benefits of within-subjects designs over between-subjects designs?

Experiment hypothesis

- Hypotheses are theory driven questions that we test to support or refute a particular phenomenon
- In this (hypothetical) experiment the hypothesis is driven by previous work showing that:
  - people prefer female voice smart speakers
  - people talk longer to female voice smart speakers
- So, we predict that:
  - participants will like the female voice more than the male voice
  - participants will also spend more time talking to the female
- We test the probability that the null hypothesis is true (i.e., the data and paramteres will not produce an effect)
- This information can then be used by researchers in computer science to modify speaker voice design and add to discussions of psychological theory

---

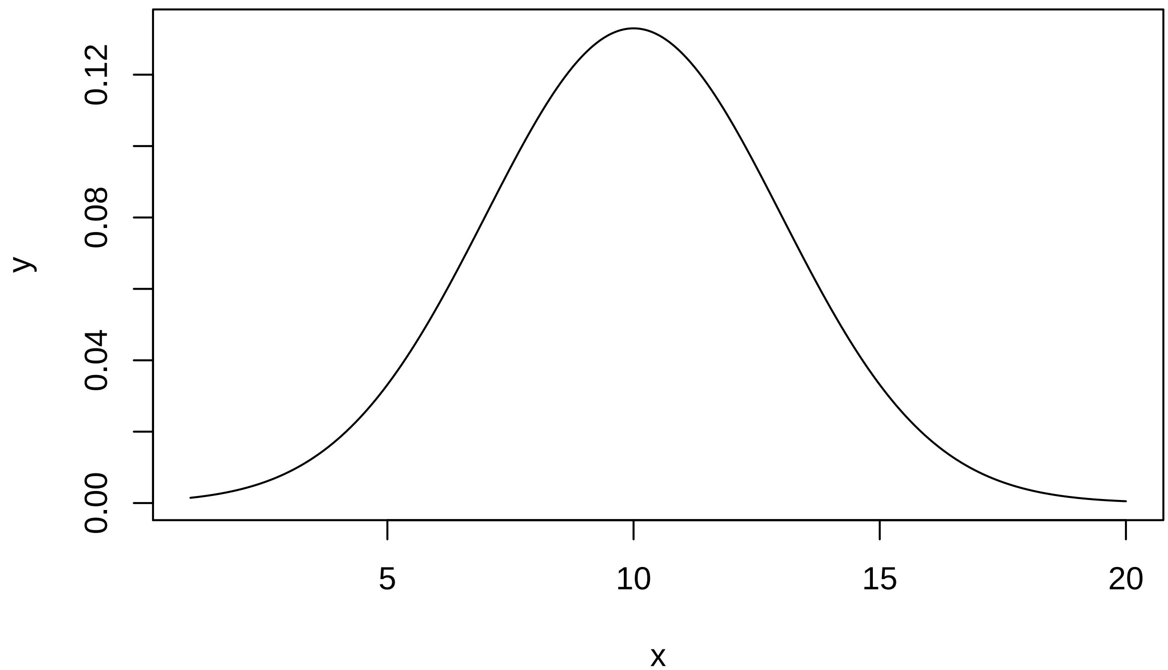
Distributions and statistical tests

- Determining how to analyse our data depends on its distribution.

- We check the distribution of ordinal data (e.g., our "like" column) using a barplot
- Ratio data (e.g., our "conv_len" column) is explored using a histogram
- The type of distribution observed will determine which statistical test we perform on the data

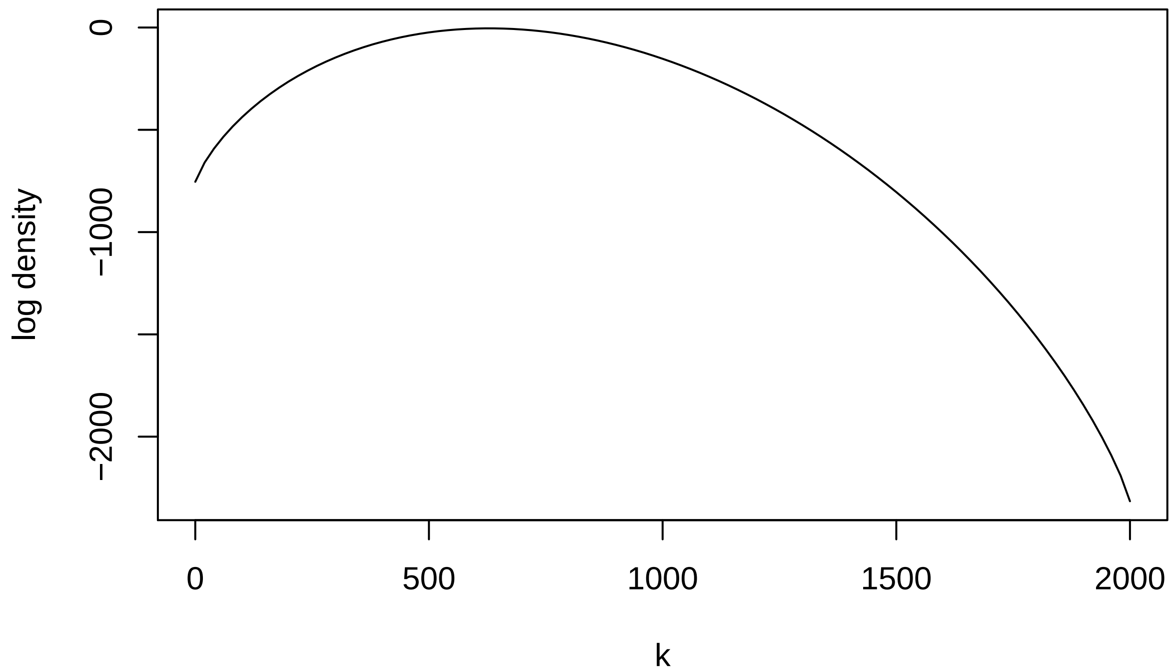
A parametric distribution

- This is what is known as a **Gaussian** (or parametric) distribution
- You can see why it is referred to as bell-shaped
- You can tell whether a distribution is parametric or not by its symmetry



A non-normal distribution

- And this is what a non-normal (non-gaussian) distribution looks like
- Notice that the area under the line is not symmetrical



- **Because Likert data is discrete rather than continuous we can assume that it is not normally distributed.**
- This is beacuse of the greater degree of variance offered by continous data (i.e., 1.026354 v 6)

Plotting the data

- We use the `ggplot2` package to generate graphics
- Note, `ggplot2` uses `+` instead of `%>%` - programming can be fickle

Continuous data: checking the distribution of conversation length

- For continuous (interval or ratio scale) data we plot a histogram to check shape of distribution

---

Plot continuous data: Live coding demo

```
# Plot convrsation length data distribution

  p1 <-                                          # creating an object for plotting allows you to...
```

```
    data %>%                                          # ...make amendments retrospectively
    ggplot(.) +
    geom_histogram(mapping = aes(x = conv_len),
                   bins = 15) +
    labs(x = "\nConversation Length (seconds)", # label the axis; `\n` creates a new line to keep tidy
         y = "Frequency\n") +
    theme_classic(base_size = 20)

# notice also that the distribution has a near Gaussian (bell) shape

# Add fill by voice_type

  p2 <-
    data %>%
    ggplot(.) +
    geom_histogram(mapping = aes(x = conv_len,
                                 fill = voice_type), # using fill here we can see where observations li
                   bins = 15) +
    labs(x = "\nConversation Length (seconds)",
          y = "Frequency\n") +
    theme_classic(base_size = 20)

# save the plot into the fig_output folder
# you can rename the plot so it has a meaningful label

ggsave("fig_output/conv_len_hist.png", p2, width = 15, height = 10)
```

Exercise 2

1. Would you say the data is parametric (normally distributed) or non-parametric (not normally distributed)?
2. Why do you say so?

Ploting discrete data summary

- Because we can assume that our Likert data is non-parametric we plot a summary of the data instead
- The best way to present discrete data is to plot a **stacked bar chart**

Stacked barplot of Likert data: Live coding demo

- To start we need to generate a summary of the data to pass to ggplot2 commands

```
# then generate Likert summary data

  likert <-
    data %>%
```

```r
    select(voice_type, like) %>%     # select the columns of interest
    group_by(voice_type, like) %>%   # group by the vectors of interest
    summarise(n = n()) %>%           # summarise the data by counting no. observations
    mutate(freq = n / sum(n))  # create a frequency vector using the new count vector

# then pass to ggplot2

  p3 <-
    likert %>%                         # our new summary object
    ggplot(., aes(x = voice_type,      # x-axis variable = voice type
                  y = freq,            # y-axis variable = frequency
                  fill = like)) +      # split bar by Likert response
    geom_bar(stat = "identity",        # use the data as is
             colour = "black") +
    labs(x = "Voice Type\n",           # label the axis
         y = "\nProportion of responses") +
    coord_flip() +                       # flip horizontally
    theme_classic(base_size = 20)      # generate with classic theme and font size 20 (clearer and tidier

# save plot into our `fig_output` directory

ggsave("fig_output/likert_bar_sum.png", p3, width = 15, height = 10)
```

- Have a look at the response frequencies
- Can you tell which speaker was preferred from eye-balling the plot?

I can see the type of distribution the data has, so what test do I use?

- For within-subjects experiments with 2 groups
  - a **t-test** is performed on parametric data
  - a **Mann Whitney U** test is performed on non-parametric data

Lab 2 information

- In lab 2 we will be learning more about data wrangling and analysis
- If you are keen to learn more about R Programming here is material related to both sessions

  R for Data Science – useful for learning *Tidyverse* syntax

  The R Book – useful for data analysis syntax

  ggplot2 cheatsheet – useful for learning to plot with ggplot2

  Data Wrangling and Tidying cheatsheet – useful for wrangling and tidying syntax of the *Tidyverse*

  R Studio cheatsheets – useful guides for all things R Studio

  Plotting means and error bars – useful guide for plotting means and error bars with ggplot2