Week 1: Intro to R

Monica Alexander and Aida Parnia

Table of contents

1 By the end of this lab you should know

- The different panes in RStudio and what they do
- How to open and make a new Quarto file
- The different parts of an Quarto file
- How to add an R chunk to an Quarto file and execute the code
- How to render a Quarto file
- Basic R coding:
    - standard mathematical operations
    - assigning values to objects
    - types of of variables
    - checking if variables or numeric values are equal, greater than or less than a number
    - different types of objects
    - what a function is and some important functions
- How to search for help on an R function
- How to install and load an R package

2 Introduction

2.1 A note about folder structure and saving files

During this course you will be downloading and saving a lot of different files. Sometimes it can be hard to find evertthing. To make it easy to find everything, I would suggest creating a folder called "soc6302" and then within that folder have a "data" folder, "labs" folder and "slides" folder where you save the relevant files.

2.2 Added bonus: R Projects

RStudio projects are associated with R working directories. They are good to use for several reasons:

- Each project has their own working directory, so make dealing with file paths easier
- Make it easy to divide your work into multiple contexts
- Can open multiple projects at one time in separate windows

To make a new project in RStudio, go to File –> New Project. If you've already set up a repo for this class, then select 'Existing Directory' and choose the folder that will contain all your class materials. This will open a new RStudio window, that will be called the name of your folder.

In future, when you want to do work for this class, go to your class folder and open the .Rproj file. This will open an RStudio window, with the correct working directory, and show the files you were last working on.

2.3 RStudio

RStudio has four different panes

1. The top left is the source pane: this is where the files that you will edit are loaded
2. The bottom left is the console. This pane shows R code that is executed
3. The top right is the environment and history. This shows the variables, datasets and other objects that have been loaded into the R environment, and the history of R code/commands that have been executed.
4. The bottom right shows the files in your current folder, plots, packages loaded, and the help files.

2.4 Different parts of an Quarto file

An Quarto file allows you to type free text and embed R code in the one document. The main parts of an Quarto file are

- the YAML: this is the bit at the top of the document surrounded by dashes. The YAML tells Markdown information like: what the title and date is, who the author is, and what the Markdown file should be render as (in this case, a pdf document).
- Headings: lines starting with # or ## or ### etc, with the text colored in blue. One # is a main heading, two ## is a sub-heading, etc
- Free text, like what this text is. Note that when the document is rendered, some formatting is applied. (you will notice that these lines that start with a - will be formatted as bullet points)
- You can view in either 'Source' mode, which shows no formatting, or 'Visual' model, which gives you an idea of the document formatting
- R chunks, shown in gray, like the one below.
    - to add an R chunk, go to the menu above this pane and click Insert –> R
    - to execute the code within the chunk, click the green play button on the right hand side of the chunk. Once you do this below, you should see the output (4) below the chunk
    - Alternatively, you can execute the code by going to Run –> current chunk in the menu above, making sure your cursor is within the code chunk
    - note that lines that start with a # within an R chunk are comments
    - to just execute one line, select that link and go Run –> Selected line (or Cmd+return on a Mac or Ctrl+Enter on Windows)

For a quick guide on codes for Quarto check out this **summary of basics**. This document is also a useful intro.

```
# This is a comment
2+2
```

```
[1] 4
```

```
# Similar to any calculator R is sensitive to the order of operations
5+(8*2)
```

```
[1] 21
```

```
(5+8)*2
```

```
[1] 26
```

```
12*(7)
```

```
[1] 84
```

2.5 How to render a Quarto file

Above I was going on about 'rendering' the document. This means to compile it to output of a particular format that is more readable or more usual for a document. In our case we are compiling to a pdf. To render this file, click the render button in the menu above. A pdf should pop up, showing a nicely formatted document.

3 R basics

Now we're going to go through some basics of R coding.

3.1 Assign values to variables

The chunk above we used R as a simple calculator (2+2) We can also assign **values** to **variables** with the back arrow i.e. `<-`. For example (execute this chunk to see the outcomes)

```
# assigning the variable x to have a value of 1
# helping ourselves by commenting code
x <- 1
monica <- 87
this_number <- 8
# assigning the variable y to have a value of 2
y <- 2
# print these
x
```

```
[1] 1
```

```
y
```

```
[1] 2
```

```
# we can add these together too
x+y
```

```
[1] 3
```

```
x*y
```

```
[1] 2
```

```
2*x - 3*y
```

```
[1] -4
```

```
# this is my code with sample size
sample_size <- 110

# squaring
sample_size^2
```

```
[1] 12100
```

3.2 Different types of variables

The above variables were numeric. But we can have character strings:

```
instructor_name <- "Monica Alexander"
first_name <- "Monica"
last_name <- "Alexander"
```

Logical: either TRUE or FALSE

```
day_is_tuesday <- FALSE
month_is_january <- TRUE

# complement is !
!day_is_tuesday
```

```
[1] TRUE
```

Factor: this is a character variable that can have an order associated with it (more later)

```
my_object <- as.factor("pen")
```

3.3 Different types of objects

Vectors: an object with multiple elements, all of the same type:

```
# the c() function allows you to create vectors of numbers (or characters)
z <- c(3,2,3,4)
z
```

```
[1] 3 2 3 4
```

```
z2 <- c("Monday", "Tuesday", "Wednesday")
z2
```

```
[1] "Monday"    "Tuesday"   "Wednesday"
```

```
places_ive_lived <- c("Australia", "USA", "Canada")

places_ive_lived
```

```
[1] "Australia" "USA"       "Canada"
```

Data frames (tibbles):

- Closest thing to a dataset that we deal with
- Each column is a different variable, each row is an observation
- Columns (variables) can be different types

More on this later.

3.4 Functions

Functions in R are commands that take arguments and do operations to variables/objects. For example, the `paste` function pastes two (or more) strings together:

```
first_name
```

```
[1] "Monica"
```

```
last_name
```

```
[1] "Alexander"
```

```
paste(first_name, last_name)
```

```
[1] "Monica Alexander"
```

```
my_full_name <- paste(first_name, "June", last_name)
```

Sidenote: R is sensitive to capitalization, both in commands and in variable names. For example using `Paste` you would get an error.

Other useful functions:

```r
z
```

```
[1] 3 2 3 4
```

```r
min(z)
```

```
[1] 2
```

```r
max(z)
```

```
[1] 4
```

```r
mean(z)
```

```
[1] 3
```

```r
length(z)
```

```
[1] 4
```

```r
x
```

```
[1] 1
```

```r
# check if x is numeric
is.numeric(x)
```

```
[1] TRUE
```

```r
# check if x is character
is.character(x)
```

```
[1] FALSE
```

3.5 In-class Exercise

Using the codes you learned, write a code that outputs, “your name, program of study” and send the code in the chat during the tutorial.

3.6 Getting help

To see what a function does, and to check the arguments, type a “?” and then the function name, for example:

```
?paste
```

Once you execute the code above, you should see that the help file for paste has appeared in the bottom right pane.

3.7 Logical statements

It is useful to check to see if variables or objects are less than, equal to or greater than numbers. Below are some examples. Note that:

- Equality is two = signs (not one)
- Each of these statements returns a **logical** value i.e. TRUE or FALSE

```
#equals
x==1
```

```
[1] TRUE
```

```
x==2
```

```
[1] FALSE
```

```
# greater than
y>1
```

```
[1] TRUE
```

```
x>1
```

```
[1] FALSE
```

```
# greater than or equal to
x>=1
```

```
[1] TRUE
```

```
# less than
x<9
```

```
[1] TRUE
```

4 Install tidyverse and load tidyverse

Throughout this course we will be using the tidyverse package a lot. You will need to install it. You can do so by either

- going to Menu: Tools –> Install packages –> type "tidyverse" click Install, or
- uncomment the code below and execute:

```
# install.packages(tidyverse)
```

Once tidyverse is installed, load it in using the library command:

```
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.4.0      v purrr   0.3.5
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.1      v stringr 1.4.1
v readr   2.1.2      v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

Side note: commands in code chunks, e.g. use this to suppress messages:

```
library(tidyverse)
```

```
[1] 4
```

5 Reading in data

The tidyverse package contains a lot of useful functions. One is `read_csv` function, which allows us to read in data from CSV files.

We are going to read in the GSS csv file. Note that you might have to change the file path below depending on where you saved the gss file.

```
# make sure the file name points to where you've saved the gss file
# for example, I have it saved in a "data" folder
library(tidyverse)
gss <- read_csv(file = "../data/gss.csv")
```

```
Rows: 20602 Columns: 85
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (63): sex, place_birth_canada, place_birth_father, place_birth_mother, p...
dbl (21): caseid, age, age_first_child, age_youngest_child_under_6, total_ch...
lgl  (1): main_activity

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The gss object is a data frame and contains a row for each respondent and a column for each variable in the dataset. The gss object technically is what is called a **tibble** – this is a weird word and originates from the fact that the guy that made the tidyverse package is from New Zealand and when people from NZ say "table" it sounds like "tibble".

You can look at the gss file by going to the "Environment" pane and clicking on the table icon next to the gss object, or by typing `View(gss)` into the console.

You can print out the top rows of the gss object by using `head`

```
head(gss)
```

```
# A tibble: 6 x 85
  caseid   age age_fir~1 age_y~2 total~3 age_s~4 age_a~5 age_a~6 dista~7 age_y~8
   <dbl> <dbl>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
1      1  52.7        27      NA       1    NA        NA    25.9      30      NA
2      2  51.1        33      NA       5    NA        NA    NA        NA      NA
3      3  63.6        40      NA       5    NA        NA    23.2      NA      NA
4      4  80          56      NA       1    NA        NA    27.3      NA      NA
```

```
5      5  28          NA      NA       0    25.3      NA    NA         NA      NA
6      6  63          37      NA       2    NA        NA    25.8       NA      NA
# ... with 75 more variables: feelings_life <dbl>, sex <chr>,
#   place_birth_canada <chr>, place_birth_father <chr>,
#   place_birth_mother <chr>, place_birth_macro_region <chr>,
#   place_birth_province <chr>, year_arrived_canada <chr>, province <chr>,
#   region <chr>, pop_center <chr>, marital_status <chr>, aboriginal <chr>,
#   vis_minority <chr>, age_immigration <chr>, landed_immigrant <chr>,
#   citizenship_status <chr>, education <chr>, own_rent <chr>, ...
```

We can print the dimensions of the gss object (number of rows and number of columns)

```
# output of this is a vector of 2 numbers
dim(gss)
```

```
[1] 20602    85
```

```
# first number = number of rows
# second number is the number of columns
```

6 Important functions

This section illustrates some important functions that make manipulating datasets like the gss dataset much easier.

6.1 select

We can select a column from a dataset. For example the code below selects the column with the respondents age:

```
select(gss, age)
```

```
# A tibble: 20,602 x 1
     age
   <dbl>
 1  52.7
 2  51.1
 3  63.6
 4  80
 5  28
 6  63
 7  58.8
 8  80
 9  63.8
10  25.2
# ... with 20,592 more rows
```

6.2 The pipe

Instead of selecting the age column like above, we can make use of the pipe function. This is the |> or %>% notation. It looks funny but it may help to read it as like saying “and then”. On a more technical note, it takes the first part of code and *pipes* it into the first argument of the second part and so on. So the code below takes the gss dataset AND THEN selects the age column:

```
gss |>
  select(age)
```

```
# A tibble: 20,602 x 1
     age
   <dbl>
 1  52.7
 2  51.1
 3  63.6
 4  80
 5  28
 6  63
 7  58.8
 8  80
 9  63.8
10  25.2
# ... with 20,592 more rows
```

Notice that the commands above don’t save anything. Assign the age column to a new object called `gss_age`

```
gss_age <- gss |>
  select(age)
```

6.3 arrange

The arrange function sorts columns from lowest to highest value. So for example we can select the age column then arrange it from smallest to largest number. Note that this involves using the pipe twice (so taking gss AND THEN selecting age AND then arranging age).

```r
# sort from smallest to largest
gss |>
  select(age) |>
  arrange(age)
```

```
# A tibble: 20,602 x 1
     age
   <dbl>
 1  15  
 2  15  
 3  15  
 4  15  
 5  15  
 6  15  
 7  15  
 8  15.1
 9  15.1
10  15.1
# ... with 20,592 more rows
```

Sort from highest to lowest:

```r
# sort from largest to smallest
gss |>
  select(age) |>
  arrange(-age)
```

```
# A tibble: 20,602 x 1
     age
   <dbl>
 1    80
 2    80
 3    80
 4    80
 5    80
 6    80
 7    80
 8    80
 9    80
10    80
# ... with 20,592 more rows
```

Side note: you need not press enter after each pipe but it helps with readability of the code.

```
gss |> select(age)
```

```
# A tibble: 20,602 x 1
     age
   <dbl>
 1  52.7
 2  51.1
 3  63.6
 4  80
 5  28
 6  63
 7  58.8
 8  80
 9  63.8
10  25.2
# ... with 20,592 more rows
```

6.4 `filter`

To filter rows based on some criteria we use the `filter` function. e.g. filter to only include those aged 30 or less:

```
gss_lt30 <- gss |>
  filter(age<=30)
```

Filter takes any logical arguments. If we want to filter by participants who identified as *Female*, we use `==` operator.

```
gss |>
  filter(sex=="Female")
```

```
# A tibble: 11,203 x 85
   caseid   age age_fi~1 age_y~2 total~3 age_s~4 age_a~5 age_a~6 dista~7 age_y~8
    <dbl> <dbl>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
 1      1  52.7       27      NA       1      NA    NA      25.9      30      NA
 2      3  63.6       40      NA       5      NA    NA      23.2      NA      NA
 3      4  80         56      NA       1      NA    NA      27.3      NA      NA
 4      6  63         37      NA       2      NA    NA      25.8      NA      NA
 5      7  58.8       40      NA       2      NA    NA      18.3      NA      NA
 6      8  80         59      NA       7      NA    22.1    22.9      NA      NA
 7      9  63.8       NA      NA       0      NA    NA      NA        NA      NA
 8     12  40.3       NA      NA       0      NA    NA      NA        NA      NA
 9     13  56.8       35      NA       4      NA    23.8    NA        NA      NA
10     14  26.8        8       5       2      18    NA      18.8      NA      NA
# ... with 11,193 more rows, 75 more variables: feelings_life <dbl>, sex <chr>,
#   place_birth_canada <chr>, place_birth_father <chr>,
#   place_birth_mother <chr>, place_birth_macro_region <chr>,
#   place_birth_province <chr>, year_arrived_canada <chr>, province <chr>,
#   region <chr>, pop_center <chr>, marital_status <chr>, aboriginal <chr>,
#   vis_minority <chr>, age_immigration <chr>, landed_immigrant <chr>,
#   citizenship_status <chr>, education <chr>, own_rent <chr>, ...
```

6.5 mutate

We can add columns using the mutate function. For example we may want to add a new column called `age_plus_1` that adds one year to everyone's age:

```
gss2 <- gss |>
  mutate(age_plus_1 = age+1)
```

Let's create a new variable `live_in_ontario`, which is equal to "Yes" if respondent lives in ontario and "No" if they don't. Use the ifelse function

```
gss3 <- gss |>
  mutate(live_in_ontario = ifelse(province=="Ontario", "Yes", "No"))

gss3 |>
  select(province, live_in_ontario)
```

```
# A tibble: 20,602 x 2
   province         live_in_ontario
   <chr>            <chr>
```

```
 1 Quebec           No
 2 Manitoba         No
 3 Ontario          Yes
 4 Alberta          No
 5 Quebec           No
 6 Quebec           No
 7 Nova Scotia      No
 8 Quebec           No
 9 British Columbia No
10 Saskatchewan     No
# ... with 20,592 more rows
```

6.6 `summarize`

The `summarize` function is used to give summaries of one or more columns of a dataset. For example, we can calculate the mean age of all respondents in the gss:

```
gss |>
  summarise(mean_age  = mean(age))
```

```
# A tibble: 1 x 1
  mean_age
     <dbl>
1     52.2
```

SHORTCUTS (MAC)

- Execute code: command+return
- To write a pipe (|>) : command+shift+M

7 Review questions

1. Create a new Quarto file for these review questions
2. Find the mean age at first birth (age_at_first_birth) of respondents in the GSS

We need to calculate the mean after removing the missing values (NA)

```
gss |>
  summarise(mean_age_fb = mean(age_at_first_birth, na.rm = TRUE))
```

```
# A tibble: 1 x 1
  mean_age_fb
        <dbl>
1        26.9
```

3. Create a new dataset that just contains GSS respondents who are less than 20 years old.

```
gsslt20 <- gss |>
  filter(age<20)
```

4. How many rows does the dataset in step 4 have?

```
dim(gsslt20)
```

```
[1] 692  85
```

5. What is the largest case id in the dataset in step 3?

```
gss |>
  arrange(-caseid)
```

```
# A tibble: 20,602 x 85
   caseid   age age_fi~1 age_y~2 total~3 age_s~4 age_a~5 age_a~6 dista~7 age_y~8
    <dbl> <dbl>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
 1  20602  26.6       NA      NA       0    21.5    NA      NA        NA      NA
 2  20601  76.4       54      NA       3    NA      20.7    22.4      NA      NA
 3  20600  43.5       14      NA       1    NA      NA      29.3      NA      NA
 4  20599  28         NA      NA       0    23.8    NA      NA        NA      NA
 5  20598  71.2       NA      NA       0    NA      28.4    NA        NA      NA
 6  20597  45.9       21      NA       2    NA      20.4    24.8      NA      NA
 7  20596  72.5       33      NA       3    NA      39      40        NA      NA
 8  20595  39.8       NA      NA       0    NA      NA      NA        NA      NA
 9  20594  47.7       18      NA       4    NA      NA      29.7      NA      NA
10  20593  80         59      NA       2    NA      NA      22.3      NA      NA
# ... with 20,592 more rows, 75 more variables: feelings_life <dbl>, sex <chr>,
#   place_birth_canada <chr>, place_birth_father <chr>,
#   place_birth_mother <chr>, place_birth_macro_region <chr>,
#   place_birth_province <chr>, year_arrived_canada <chr>, province <chr>,
#   region <chr>, pop_center <chr>, marital_status <chr>, aboriginal <chr>,
#   vis_minority <chr>, age_immigration <chr>, landed_immigrant <chr>,
#   citizenship_status <chr>, education <chr>, own_rent <chr>, ...
```