Week 1: Set-up and basics

Monica Alexander*

8/1/2022

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 R Markdown file
- The different parts of an R Markdown file
- How to add an R chunk to an R Markdown file and execute the code
- How to knit a R Markdown file
- How to assign values to variables in R
- How to check if variables or numeric values are equal, greater than or less than a number
- How to search for help on an R function
- How to install and load an R package
- How to read in data from a csv file
- How to view a dataset in R
- What the pipe `%>%` is
- The functions `select`, `arrange`, `filter`, `mutate` and `summarize`

Introduction

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.

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

*Thanks to Aida Parnia who greatly improved aspects of this lab.

- 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 folder 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.

A note about folder structure and saving files

During this course you will be downloading and saving a lot of different files. To make it easy to find everything, I would suggest creating a folder called "soc6707" and then within that folder have a "data" folder, "labs" folder and "slides" folder where you save the relevant files. Create an R Project within this folder and then whenever you have to do analysis, open RStudio via that R Project.

Different parts of an R Markdown file

An R Markdown file allows you to type free text and embed R code in the one document. The main parts of an R Markdown 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 knit 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 knitted, some formatting is applied. (you will notice that these lines that start with a - will be formatted as bullet points)
- 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 R Markdown check out this **cheat sheet**.

```
# 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
```

How to knit a R Markdown file

Above I was going on about 'knitting' 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 knit this file, click the Knit button in the menu above. A pdf should pop up, showing a nicely formatted document.

R basics

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

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
x <- 1
# 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
```

Side note: you may see in other R codes that `=` is also used to assign values to a variable.

Values need not just be numbers:

```
# the c() function allows you to create vectors of numbers (or characters)
z <- c(3,4,3.2,5.1)
z
```

```
## [1] 3.0 4.0 3.2 5.1
```

```
# pull out variable parts of z
z[1]
```

```
## [1] 3
```

```
z[3]
```

```
## [1] 3.2
```

```
# length of z
length(z)
```

```
## [1] 4
```

```
# the 5th element doesn't exist
z[5]
```

```
## [1] NA
```

Character strings:

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

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:

```
z
```

```
## [1] 3.0 4.0 3.2 5.1
```

```
max(z)
```

```
## [1] 5.1
```

```
mean(z)
```

```
## [1] 3.825
```

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

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

```
paste(first_name, "June", last_name)
```

```
## [1] "Monica June Alexander"
```

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

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.

```
name <- "Monica"
program_of_study <- "done with exams for life"
paste(name, program_of_study, sep = ", ")
```

```
## [1] "Monica, done with exams for life"
```

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.

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
```

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")
# install.packages("dplyr")
```

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

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.5     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Set Working Directory

For R to access your files, it needs to know where your files are located. You can set your working directory by going to Menu –> Session –> Set Working Directory...

NOTE: if you are using an R Project, you do not need to do this!

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 will probably 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
gss <- read_csv(file = "../data/gss.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   caseid = col_double(),
##   age = col_double(),
##   age_first_child = col_double(),
##   age_youngest_child_under_6 = col_double(),
##   total_children = col_double(),
##   age_start_relationship = col_double(),
##   age_at_first_marriage = col_double(),
##   age_at_first_birth = col_double(),
##   distance_between_houses = col_double(),
##   age_youngest_child_returned_work = col_double(),
##   feelings_life = col_double(),
##   hh_size = col_double(),
##   number_total_children_intention = col_double(),
##   number_marriages = col_double(),
##   fin_supp_child_supp = col_double(),
##   fin_supp_child_exp = col_double(),
##   fin_supp_lump = col_double(),
##   fin_supp_other = col_double(),
##   is_male = col_double(),
##   main_activity = col_logical()
##   # ... with 2 more columns
## )
## i Use `spec()` for the full column specifications.
```

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_first_child age_youngest_chi~ total_children age_start_relat~
##    <dbl> <dbl>           <dbl>             <dbl>          <dbl>            <dbl>
## 1      1  52.7              27                NA              1               NA
## 2      2  51.1              33                NA              5               NA
## 3      3  63.6              40                NA              5               NA
## 4      4  80                56                NA              1               NA
## 5      5  28                NA                NA              0             25.3
## 6      6  63                37                NA              2               NA
```

```
## # ... with 79 more variables: age_at_first_marriage <dbl>,
## #   age_at_first_birth <dbl>, distance_between_houses <dbl>,
## #   age_youngest_child_returned_work <dbl>, 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>, ...
```

```
# or bottom rows
tail(gss)
```

```
## # A tibble: 6 x 85
##   caseid   age age_first_child age_youngest_chi~ total_children age_start_relat~
##    <dbl> <dbl>           <dbl>              <dbl>          <dbl>             <dbl>
## 1  20597  45.9              21                 NA              2              NA
## 2  20598  71.2              NA                 NA              0              NA
## 3  20599  28                NA                 NA              0              23.8
## 4  20600  43.5              14                 NA              1              NA
## 5  20601  76.4              54                 NA              3              NA
## 6  20602  26.6              NA                 NA              0              21.5
## # ... with 79 more variables: age_at_first_marriage <dbl>,
## #   age_at_first_birth <dbl>, distance_between_houses <dbl>,
## #   age_youngest_child_returned_work <dbl>, 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>, ...
```

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
# first number = number of rows
# second number is the number of columns
dim(gss)
```

```
## [1] 20602    85
```

Important functions

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

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
```

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

```
## # A tibble: 20,602 x 2
##      age education
##    <dbl> <chr>
##  1  52.7 High school diploma or a high school equivalency certificate
##  2  51.1 Trade certificate or diploma
##  3  63.6 Bachelor's degree (e.g. B.A., B.Sc., LL.B.)
##  4  80   High school diploma or a high school equivalency certificate
##  5  28   College, CEGEP or other non-university certificate or di...
##  6  63   High school diploma or a high school equivalency certificate
##  7  58.8 Less than high school diploma or its equivalent
##  8  80   Less than high school diploma or its equivalent
##  9  63.8 High school diploma or a high school equivalency certificate
## 10  25.2 Less than high school diploma or its equivalent
## # ... with 20,592 more rows
```

The pipe

Instead of selecting the age column like above, we can make use of the pipe function. This is the %>% 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)
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
```

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).

```
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
```

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

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 %>%
  filter(age<=30)
```

```
## # A tibble: 2,753 x 85
##    caseid   age age_first_child age_youngest_ch~ total_children age_start_relat~
##     <dbl> <dbl>           <dbl>            <dbl>          <dbl>            <dbl>
##  1      5  28                NA               NA              0             25.3
##  2     10  25.2               4                4              1             NA
##  3     11  15.7              NA               NA              0             NA
##  4     14  26.8               8                5              2             18
##  5     22  25.5              NA               NA              0             NA
##  6     24  16.3              NA               NA              0             NA
##  7     28  19.1              NA               NA              0             NA
##  8     33  16.8              NA               NA              0             NA
##  9     37  29.6              NA               NA              0             NA
## 10     39  18.6              NA               NA              0             NA
## # ... with 2,743 more rows, and 79 more variables: age_at_first_marriage <dbl>,
## #   age_at_first_birth <dbl>, distance_between_houses <dbl>,
## #   age_youngest_child_returned_work <dbl>, 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>, ...
```

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

```
gss %>%
  filter(sex=="Female") %>%
  select(sex, age)
```

```
## # A tibble: 11,203 x 2
##    sex      age
##    <chr>  <dbl>
##  1 Female  52.7
##  2 Female  63.6
##  3 Female  80
##  4 Female  63
##  5 Female  58.8
##  6 Female  80
##  7 Female  63.8
##  8 Female  40.3
##  9 Female  56.8
## 10 Female  26.8
## # ... with 11,193 more rows
```

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:

```
gss %>%
  select(age) %>%
  mutate(age_plus_1 = age+1)
```

```
## # A tibble: 20,602 x 2
##      age age_plus_1
##    <dbl>      <dbl>
##  1  52.7       53.7
##  2  51.1       52.1
##  3  63.6       64.6
##  4  80         81
##  5  28         29
##  6  63         64
##  7  58.8       59.8
##  8  80         81
##  9  63.8       64.8
## 10  25.2       26.2
## # ... with 20,592 more rows
```

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 %>%
  select(age) %>%
  summarize(mean_age = mean(age))
```

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

```
gss %>%
  filter(sex=="Female") %>%
  summarize(count_Female = n())
```

```
## # A tibble: 1 x 1
##   count_Female
##          <int>
## 1        11203
```

In-class exercise

Using `filter`, `select`, and `summarize` commands, find out if the mean age is higher for *Male* or *Female* in our sample.

Review questions

1. Create a new R Markdown file for these review questions
2. Create a variable called "my_name" and assign a character string containing your name to it
3. Find the mean age at first birth (age_at_first_birth) of respondents in the GSS

4. Create a new dataset that just contains GSS respondents who are less than 20 years old.
5. How many rows does the dataset in step 4 have?
6. What is the largest case id in the dataset in step 4?