Chapter 3: Importing and Exploring Data

Dr Megan L. Wood

2022-11-30

Chapter Overview:

1. **Installing libraries**
2. **Importing data**
3. **Missing Data**
4. **Creating subsets**
5. **Pipes**
6. **Renaming variables**
7. **Descriptive statistics**

Installing our library

For this workbook, we are going to be using functions from the **tidyverse** package.

You only need to install a package once, but you need to reload it every time you start a new R session using library().

install.package("tidyverse") # notice the quotations

```
library(tidyverse) # call the tidyverse library to our workspace
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1

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

Importing data

Now, let's get some data! You should have saved the data into your project within the "Data" folder. If you followed this set-up, you should be able to load in the csv using the below code.

```
read.csv("./Data/ckat_data.csv") # This loads in the data, but it is not actually saved in our space

ckat.dat <- read.csv("./Data/ckat_data.csv") # This loads in the data, and saves it to our environment
```

We can take a look at the first few rows of data using the head() function.

```
head(ckat.dat)
```

Taking a closer look at our data...

There are many functions within both base R and the tidyverse package which can be used to understand our data a little better. These two functions give us very similar outputs. Often, there a many ways we can achieve the same output, depending on the packages and functions we use, or personal preference.

```
glimpse(ckat.dat)
str(ckat.dat)
```

Missing data

We might also want to check if we have any missing data in our dataset. There are various different ways we can do this. Have a play around with the different examples below.

```
# Are there any missing values in our dataset?
is.na(ckat.dat)

anyNA(ckat.dat)

sum(is.na(ckat.dat))

# We can also check if we have missing data in a particular variable
anyNA(ckat.dat$ID)
```

Creating subsets

filter()

Sometimes, we might be interested in looking at only a smaller subset of our data. For example, we might want to just look at the girls in the sample. We can do this using the **filter()** function. This filters out the number of **rows** we have in our new dataframe.

This takes the format: new_df <- filter(existing_df, variable_name == "Value")

NOTE - we use "==" when something we refer to something as being "equal to". A single "=" is used as another way of assigning values.

```
girls.dat <- filter(ckat.dat, Gender == "Female")
```

We can also subset part of the sample based on numeric values. For example, we might want to run analyses with only children under 8 years.

Notice also, that this still includes females, as we have subsetted from the original dataset.

```
young.dat <- filter(ckat.dat, Age < 8)
```

If we want to be *really* particular, we can combine multiple conditions to create a new subset.

```
u8_girls.dat <- filter(ckat.dat, Age < 8 & Gender == "Female")
```

na.omit()

If, for example, we wanted to have a complete dataset with no missing data at all (i.e., no NA values), we have a special function specific to NA values. This creates a new dataframe where any rows containing a missing value (NA) are removed.

```
ckat_no_missing.dat <- na.omit(ckat.dat)
```

select()

Similarly, we might realise that we don't actually need all the variables we have in our dataset. We can use the **select** function to select only the variables/columns we need. Again, this is useful as we are not making any changes to our raw data.

```
# We select the columns we want to keep.
smaller.dat <- select(ckat.dat, ID, Gender, Handedness, Age, weighted_mean_track, weighted_mean_aim, wei
```

We can also do this another way - depending on the size of our dataframe and how many variables we actually want to retain. We can select the columns we want to *remove*.

```
# We select the columns we want to keep.
smaller_v2.dat <- select(ckat.dat, -Dataset, -AgeMonths, -LCA_SEP, -ethnicity, -ethnicity_3gp)
```

Pipes

%>% is called a "pipe"

Think of it as "and then" … R will then run the first part, then move onto the next part etc.

Let's break this down with an example…

Say we want to both select several columns *and then* filter the rows from those columns. We have already done both of these earlier:

```
# Select the columns of choice
ckat_aim.dat <- select(ckat.dat, ID, Gender, weighted_mean_aim)

# Filter to include females only
filter(ckat_aim.dat, Gender == "Female")
```

However - we can use pipes to simplify these two functions into one. This means that we avoid having to create a new dataframe and store it in the memory and it is quicker to run, too!

```
# As above using pipes
ckat.dat %>%
  select(ID, Gender, weighted_mean_aim) %>%
  filter(Gender == "Female")
```

Have a go with the below example - first run the function how we have previously, then try and do the same thing using pipes to combine multiple functions into one call.

```
# Now your turn...


# Omit the NA values
ckat_no_NA.dat <- na.omit(ckat.dat)

# Filter to include only children under 6
filter(ckat_no_NA.dat, Age < 6)

# Using pipes...?
```

Renaming variables/columns

One important aspect in data wrangling is re-naming variables/columns. This is especially important when we receive data that includes variable names that are not very clear or meaningful.

For example, we may decide that for ease, we want to rename the variable **weighted_mean_track** into simply **tracking**.

This takes the format:

my_data %>% rename(new_name = old_name, new_name2 = old_name2)

```
ckat.dat <- ckat.dat %>% #this overwrites the ckat.dat table
  rename(
    tracking = weighted_mean_track
  )
```

Try to do the same with **weighted_mean_aim** and **weighted_mean_steer** (don't forget to set up your code chunk!)

Descriptive Statistics

Descriptive statistics are another really important way for us to better understand the data we are working with! It also makes it easy to spot any mistake or irregularities that we can then inspect further.

Mean and standard deviation

Let's start with trying to work out the mean and standard deviation of children's ages.

```
mean(ckat.dat$Age)
sd(ckat.dat$Age)
```

As you can see, this does not produce a valid output. The mean cannot be computed when we have missing data. Therefore, we need to tell R to ignore it in the analysis. Unlike when we used na.omit earlier, we haven't actually removed these missing data from our dataframe - just from this particular code.

```
mean(ckat.dat$Age, na.rm = TRUE)
sd(ckat.dat$Age, na.rm = TRUE)
median(ckat.dat$Age, na.rm = TRUE)
min(ckat.dat$Age, na.rm = TRUE)
max(ckat.dat$Age, na.rm = TRUE)
```

We can also run within the **summarise** function to wrap the ouput in a nice **tibble** for us. This becomes especially useful when we conduct multiple descriptives together.

summarise()

```
ckat.dat %>%
  summarise(mean = mean(Age, na.rm = TRUE))
```

Adding in multiple descriptives in one go: Notice here how we have named the column (before the equals). Try this without and see what happens to your output!

```
ckat.dat %>%
  summarise(mean = mean(Age, na.rm = TRUE),
            sd = sd(Age, na.rm = TRUE))
```

group_by()

Another super useful function is **group_by**. This is where those pipes we learned earlier really come in handy!

Now we want to know how the mean age varies by gender (hopefully it doesn't by much!). We can do this using the **GROUP BY** function and add it into our code.

```
ckat.dat %>% # select the dataframe we are interested in
  group_by(Gender) %>% # group the output by the variable "Gender"
  summarise(mean = mean(Age, na.rm = TRUE), # Summarise the age data by mean and sd
            sd = sd(Age, na.rm = TRUE))
```

count()

If we wanted to simply **count** the number of individuals within each group, we can use the **n** function. Have a play around with the below code to see how this works.

```
ckat.dat %>%
  group_by(Gender) %>%
  summarise(n = n())
```

```
ckat.dat %>%
  group_by(Gender, Age) %>%
  summarise(n = n())
```

```
## ‘summarise()‘ has grouped output by ’Gender’. You can override using the
## ‘.groups‘ argument.

## # A tibble: 22 x 3
## # Groups:   Gender [3]
##    Gender   Age     n
##    <chr>  <int> <int>
##  1 Female     4   376
##  2 Female     5   462
##  3 Female     7   701
##  4 Female     8  1265
##  5 Female     9   506
##  6 Female    10    52
##  7 Female    NA    12
##  8 Male       0     1
##  9 Male       4   385
## 10 Male       5   441
## # ... with 12 more rows
```