`R` Tutorial 2: Importing & exploring your data

Carlo Knotz

Contents

1 Introduction

You have so far learned how to install and set up `R` and `RStudio`, how you can install and load packages, how data look like in `R`, and how you write and use code. All of this was essentially a warm-up.

Now things get a bit more real: This week, you will learn how to open a real research dataset and how to explore it in `R`.

But we will take this one step at a time: You will first learn about data exploration with a small dataset that is already installed on your computer. Then you will import a real dataset (from the *European Social Survey*). This is to prepare you for the in-class exercises, where you will apply the data-exploration techniques you learned in the tutorial to the full-scale ESS dataset.

Important: As before, document your code in a dedicated scriptfile as you work your way through the tutorial – do not rely on the *Console* (unless you are just installing packages or quickly trying things out).

Hvis du ønsker å lese en norsk tekst ***i tillegg****: "Lær deg R", Kapittel 4*

---

2 Setup

2.1 Your project folder

The first thing you need to do is to make sure that you are working in the *Project* (and the associated folder) that you created in the first seminar/lab in Week 1 of the course.[1]

Look at the upper-right corner of the *RStudio* window and check that your project is active. It should **not** say: “Project: (None)”. Instead, you should see the name of the project you created. (If you do see “Project: (None)” written there, you can click on it to open a drop-down menu in which your project should be listed. You can open it there.)

Once you are done with make sure that you know where on your computer your project folder is; navigate there in the Windows File Explorer/Mac Finder.

[1] Did you miss that session? You can read about *Projects* and how you create them here: https://support.posit.co/hc/en-us/articles/200526207-Using-RStudio-Projects or in *Lær deg R*, 4.1.1.3.

2.2 Loading the `bst290` package and the practice dataset

You will remember that you installed a number of *packages* previously, one of which was the `bst290` package. This package includes, among other things, a small practice dataset that you will use in this and the other tutorials to get familiar with the various operations in `R` before you move on to the "real-deal" research datasets.

The practice dataset in the `bst290` package is a fragment of the *European Social Survey* data that were collected in Norway in 2014. In essence, this practice dataset is a mini-version of the full ESS dataset. Where the full ESS includes data for more than 1000 survey participants and hundreds of variables, the practice dataset includes only data for 143 Norwegian respondents and 22 variables.

To access the data, you first need to load the `bst290` package with the `library()` function:

```r
library(bst290)
```

Then you can open the dataset (which is called `ess`) with the `data()` function:

```r
data(ess)
```

If everything worked, then you should now see the `ess` dataset listed in the *Environment* panel (upper right of your screen). You will probably see `<Promise>` written where the dataset summary and the variables should appear — and you can take this literally: `R` promises you that the dataset will appear once you start using it. So, all you need to do is to call up the dataset in some way, for example by simply typing `ess` into the *Console*.

Once the dataset is properly loaded, you should see in the *Environment* panel that the dataset includes 143 observations and 22 variables.

You can also get the dataset directly with the "double-colon" method:

```r
ess <- bst290::ess
```

Translated into human language, this tells `R` to "get the `ess` dataset from the `bst290` package and save it under the name `ess` in the *Environment*".

3 Exploring data in `R`

3.1 A first glimpse

Take a look at the `ess` object in the *Environment* tab — can you see the tiny blue circle with the white triangle/arrow inside it that is directly to the left of `ess`?

If you click on it, you can get more information about the different variables that are included in the dataset.

- You should now see a list of variable names (`name`, `essround`, `idno`,…). Each of these variables is a collection of data points — and therefore stored as a *vector* in `R` (you may remember from the previous tutorial). All these vectors are then combined into the `ess` dataset (or, in `R` lingo, `data.frame`).
- Next to these names, you also see `chr` or `num` written — as you probably remember, this tells you what type of information each variable contains.
- You may also notice that some elements in the list are followed by the phrase `Factor w/ XX levels...` — these are so called `factors` and are a particular type of vector. You will learn about them further below.

3.2 Looking at the data

Let's first get an idea of how the dataset really looks like, which you can do with the `View()` function. To do that, run the following in your *Console*:

```r
View(ess)
```

A new tab should now open and you should see the entire dataset. This should look a bit like Microsoft Excel, a large table with lots of neat and orderly but boring rows and columns of data.

3.3 Printing out the first and last observations with `head()` and `tail()`

Looking at the raw dataset is often quite helpful to get a first idea of what you are working with — but is impractical when you are working with very large datasets.

An alternative way to get a first glimpse of your dataset is to use the `head()` and `tail()` functions. These show you the first and last six rows (observations) of your dataset — in essence, they print out the top or bottom of the dataset.

3.3.1 Default usage

Using them is simple, you just need to specify the name of your dataset within the function. For example, to display the *first* six observations in the `ess` dataset, you run:

```r
head(ess) # This shows you the first 6 observations
```

The result should look like this:

```
  essround  idno cntry   gndr agea
1        7 12414    NO   Male   22
2        7  9438    NO Female   43
3        7 19782    NO Female   58
4        7 18876    NO Female   22
5        7 20508    NO   Male   84
6        7 19716    NO   Male   62
                                                                      edlvdno
1    Fullført 3-4 årig utdanning fra høgskole (Bachelor-, cand.mag., lærerhøgsko
2    Fullført 3-4 årig utdanning fra høgskole (Bachelor-, cand.mag., lærerhøgsko
3                    Fullført 5-6 årig utdanning fra høgskole (master, hovedfag)
4    Fullført 3-4 årig utdanning fra høgskole (Bachelor-, cand.mag., lærerhøgsko
5    Universitet/høgskole, mindre enn 3 år, men minst 2 år (høgskolekandidat, 2-
6 Fullført 5-6 årig utdanning fra universitet (master, hovedfag), lengre profesj
  mainact           mbtru        hinctnta                              tvtot
1    <NA>              No H - 10th decile                      No time at all
2    <NA>              No H - 10th decile  More than 1 hour, up to 1,5 hours
3    <NA>  Yes, currently  K - 7th decile More than 2 hours, up to 2,5 hours
4    <NA>              No  J - 1st decile More than 1,5 hours, up to 2 hours
5 Retired Yes, previously            <NA>                      No time at all
6    <NA>  Yes, currently H - 10th decile  More than 1 hour, up to 1,5 hours
  ppltrst vote            stflife                    gincdif
1       7  Yes Extremely satisfied Neither agree nor disagree
2       9  Yes                   8 Neither agree nor disagree
3       9  Yes                   7             Agree strongly
4       5   No                   7 Neither agree nor disagree
5       7  Yes                   9 Neither agree nor disagree
6       7  Yes                   8          Disagree strongly
                     freehms imwbcnt           happy    health ctzcntr brncntr
1                      Agree       4               9      Good     Yes     Yes
2             Agree strongly       4               9 Very good     Yes     Yes
3             Agree strongly       5               8      Good     Yes     Yes
4                      Agree       5 Extremely happy Very good     Yes     Yes
5 Neither agree nor disagree       5               7 Very good     Yes     Yes
```

```
6                      Agree        6               8       Fair      Yes      Yes
  height weight
1    175     65
2    175     71
3    150     58
4    173     63
5    167     58
6    174     58
```

3.3.2 Looking at specific variables

If the result above seems pretty cluttered and not very informative: Correct. But there is a solution. You can specify that only the first observations of a single variable are shown when you run `head()` or `tail()`. This can help when the dataset contains a larger number of variables and the output therefore becomes cluttered – as was the case here.

Take another quick look at the *Environment* window: You might have noticed that there are dollar symbols (`$`) before each of the variable names in the `ess` dataset. This is a hint to how you can select single variables from a dataset: With the dollar symbol.

The general syntax here is: `dataset$variable`. For example, to select the age-variable `agea` from the `ess` dataset, you would type: `ess$agea`

You can use this with the `head()` function to let `R` show you the first six observations of only the `agea`-variable:

```
head(ess$agea)
[1] 22 43 58 22 84 62
```

Of course, you can do this also with any of the other variables — and this works also with many other functions such as `tail()`, `mean()`, or `summarize()`. More follows!

3.3.3 Defininining the number of observations (“rows”)

You can also tell `R` to show you more or fewer observations when you use the `head()` function. For example, the code below will print out the first 10 observations of the `agea` variable:

```
head(ess$agea, n = 10)
```

You can do the same with the `tail()` function.

(A final note: As is often the case with `R`, there is more than one way to subset a dataset, and these allow you to select more than one variables at a time, or a specific set of observations. We will cover some of them in the next tutorial; for others see e.g.: https://www.statmethods.net/management/subset.html.)

3.4 A quick summary of your data with `summary()`

With `View()`, `head()`, or `tail()`, you can look at the "raw" dataset. This can give you a first idea of what you are working with, but the problem is that you always only see a few data points at a time. Ideally, you would instead get a sense of how the entire dataset or single variables as a whole look like.

This is where you would use summary statistics like the mean ("average"), the median, or others (as explained in Kellstedt & Whitten).

You can get some important summary statistics with the `summary()` function.

This function is again easy to use: You just specify which object you want summarized within the parantheses. In this case, we use the function on the entire `ess` dataset:

```
summary(ess)
```

If you run this, you should get a list of summary statistics for all the variables in the `ess` dataset. For variables that contain numbers ('numeric' variables, or `num`), you get the minimum, the 1st quartile (a.k.a., the 25th percentile), the median, the mean ('average'), the 3rd quartile (or 75th percentile) and the maximum. Where variables have missing observations (`NA`'s), you get these, too.

For non-numeric variables (like `cntry`, for example) you get their 'length' (how many observations they contain) and their type or 'Class'.

But, as before, the output is again a bit cluttered (which is also why it is not shown here). It is therefore more useful to get summary statistics for a single variable by using the `$` symbol. For example:

```
summary(ess$agea)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  16.00   33.50   46.00   47.91   62.00   90.00 
```

Here, `Min.` means "Minimum", `1st Qu.` means "First Quartile", `Median` and `Mean` are obvious, `3rd Qu.` means "Third Quartile", and `Max.` means "Maximum". If you read Kellstedt/Whitten (2018, Chapter 6), then you should know how to interpret these different statistics.

3.5 Specific summary statistics for numeric variables

While `summary()` provides you a whole list of summary statistics, you often want a specific measure of central tendency or spread for a given variable.

These are easy to get in `R`; all you need are four functions, all with quite intuitive names:

- `mean()` for the mean or "average";
- `median()` for the median or "50th percentile";
- `var()` for the variance;
- `sd()` for the standard deviation;

Using these functions is straightforward — for example, to get the mean of the age-variable (`agea`) in the `ess` dataset, you just run:

```
mean(ess$agea)
[1] 47.90909
```

Getting the other summary statistics works the same way:

```
median(ess$agea)
[1] 46

sd(ess$agea)
[1] 18.5658

var(ess$agea)
[1] 344.6889
```

3.5.1 When you have missing observations (NAs)

It is often the case that your variables contain missing information — indicated in R as NA. This happens for example when surveys include sensitive questions about people's incomes or their sexual orientation, which many respondents refuse to disclose The result is then an NA ("not available") for that particular respondent and variable.

Important: The mean(), median(), sd(), and var() functions (and many others) will not give you a proper result *if there is even a single NA in your variable!*

Fortunately, there is an easy solution: All four functions have an option to remove NAs from the data before calculating the respective summary statistic; this option is called na.rm ("NA remove"). You just have to set this option to TRUE (switch it on) to take care of missings, for example:

```
mean(ess$agea, na.rm = TRUE)
[1] 47.90909
```

(Make sure you always add a comma between different parts or "arguments" of a function!)

3.6 Working with categorical or ordinal variables

3.6.1 Introducing *factors*

The variable you have been working with so far, `agea`, is a typical numeric variable: It measures a respondent's age in years, and age is by nature a number. In this case, calculating statistics such as the mean makes sense.

But there are also other variables such as categorical or ordinal variables, where things are a bit different. Consider for example the variable that records the respondent's gender, `gndr`. Obviously, gender is by nature a categorical variable: It has two or more distinct categories (e.g., male, female, diverse), and these categories are *unordered*, meaning 'male' is obviously not a 'higher' or 'better' category than 'female' or 'diverse'. They are all simply different categories people can fall (or be put) into.

Other times, you may be dealing with *ordinal* variables (e.g., a Likert-scale: "disagree completely", "disagree","neither", "agree", "agree completely"). In these cases, there is an order — but you cannot give a precise number for how much higher "agree completely" is compared to "agree". One is *more* than the other, but the difference between them is not clearly defined with a number.

In `R`, categorical or ordinal variables are usually stored as *factors*. *Factors* are a separate kind of variable or "vector" (next to numeric or `num` and character or `chr` variables). You can think of *factors* as "numbers with labels".

For example, take another look at the *Environment* tab (upper right of your screen) and look for the `gndr` variable. You can see directly that it is designated as a "*Factor*" with 2 levels — but also that there is a row of numbers (`1,2,2,...`) behind the two levels "Male" and "Female".

This means:

- Every male respondent gets the number `1`; that number then gets the label "Male" attached to it;
- Every female respondent gets the number `2`; that number is then labeled "Female";

The same applies also to the (many) other factor variables in the `ess` dataset, or other datasets. Again: *Factors* are essentially just numbers with text labels.

3.6.2 Identifying factor variables

First, you should be able to *identify* that a given variable is indeed a *factor* variable. You can of course see this in the *Environment*, but this works only for a small dataset like the one you are using now. If you would work with the full ESS data, there would be many more variables and not all of them would be shown in the *Environment*.

You can use the `class()` function to let `R` tell you which type or *class* a specific variable is saved as. You use this like the other functions above (`dataset$variable`).

Let's check if the gender-variable (`gndr`) is really saved as a factor, as it should be:

```
class(ess$gndr)
[1] "factor"
```

Now compare this to the age variable (`agea`):

```
class(ess$agea)
[1] "numeric"
```

This is by nature a numeric variable, and it turns out that `R` has stored it properly.

Important: You cannot rely on that this always works! It is often the case that one or more variables in your dataset are *not* stored properly, which then usually causes warnings and errors. In this case, you first need to identify the issue — and you now know how to do that — and then you need to fix it. You will learn how to do this in the next tutorial.

3.6.3 Getting familiar with factor variables

Once you have identified a factor variable, you will usually want to learn more about it. But getting familiar with *factors* can be a bit tricky at first. Many summary statistics will not work here. For example, if you try to calculate the mean of a factor variable, R will refuse to do so:

```
mean(ess$gndr)
Warning in mean.default(ess$gndr): argument is not numeric or logical:
returning NA
[1] NA
```

This does make sense: Many summary statistics are only appropriate if you are dealing with proper numbers, but here you have only categories. But this also means that you have to use different ways to learn how a factor variable in your dataset looks like.

3.6.4 Getting the structure of a factor-type variable

A first option is to let `R` print out the structure of the variable using `str()` ("structure"):

```
str(ess$gndr)
 Factor w/ 2 levels "Male","Female": 1 2 2 2 1 1 1 1 1 1 ...
```

This tells you that `gndr` has two categories ("Male" & "Female") and that these are encoded with the numbers `1` and `2` in the dataset.

What is not fully clear from this output, however, is which number really corresponds to which label — are men now coded as `1` or as `2`? And this is also generally one of the things that can make working with factors daunting: it is a bit difficult to see 'under the hood' of a factor: how its text labels correspond to the numerical values underneath.

But you do have a tool to figure this out!

3.6.5 How numerical values and text labels correspond

The `visfactor()` function in the `bst290` package allows you to see which number corresponds with which label in a given factor-type variable.

For example, to see the labels and numerical values of the `gndr` variable, you would run:

```
visfactor(variable = "gndr", dataset = ess)
 values labels
      1   Male
      2 Female
```

3.6.6 Empty categories in factor-type variables

Another important thing to figure out is whether a particular factor variable in your dataset has empty categories. For example, you might be working with data from a survey in which respondents were asked whether they are working, in education, or unemployed — and it just so happened that none of the respondents were unemployed at the time. In this case, "unemployed" would be an empty category in the data.

The easiest way to see if there are empty categories in a factor variable is to let `R` show you how many observations you have for each of the categories of the variable. To do so, you use the `table()` function.

This is how you would do this with the `gndr` variable:

```
table(ess$gndr)

  Male Female 
    75     68 
```

You see that there are 75 men and 68 women in the dataset — and there are no empty categories.

But now compare this to the case of the `mainact` variable, which tells you about the respondent's main activity of the last seven days (whether they were working, unemployed, etc.):

```
table(ess$mainact)

                              Paid work 
                                     15 
                              Education 
                                      7 
                 Unemployed, looking for job 
                                      0 
             Unemployed, not looking for job 
                                      0 
              Permanently sick or disabled 
                                      1 
                                Retired 
                                      7 
               Community or military service 
                                      0 
Housework, looking after children, others 
                                      3 
                                  Other 
                                      0 
```

It turns out that there are indeed some empty categories: There are no unemployed respondents in the dataset, and none of them was doing military or community services.

An alternative way to identify empty categories is to let `R` first print out which categories a factor variable can *theoretically* have and then compare that to what categories are *actually represented* in the dataset.

To see which categories your factor-variable can *theoretically* have, you use the `levels()` function:

```
levels(ess$mainact)
[1] "Paid work"
[2] "Education"
[3] "Unemployed, looking for job"
[4] "Unemployed, not looking for job"
[5] "Permanently sick or disabled"
[6] "Retired"
[7] "Community or military service"
[8] "Housework, looking after children, others"
[9] "Other"
```

You see that the `mainact` variable has, in theory, nine categories in total, ranging from "Paid work" to "Other".

Now, to see which of these categories are really present in the data, you can use the `unique()` function:

```
unique(ess$mainact)
[1] <NA>
[2] Retired
[3] Paid work
[4] Education
[5] Housework, looking after children, others
[6] Permanently sick or disabled
9 Levels: Paid work Education ... Other
```

You see that only five (plus the `NA`s) of the nine categories are listed — and being unemployed is not one of them.

3.7 Custom functions for summary tables

Since it is a statistical programming language, `R` can be used to generate pretty much any type of summary table for any kind of situation you could think of. In addition, there are special packages for more advanced tables, for instance:

- `gtsummary` (https://www.danieldsjoberg.com/gtsummary/index.html)
- `xtable` (https://cran.r-project.org/web/packages/xtable/vignettes/xtableGallery.pdf)

But: Learning how to use `R` functions to create tables takes a while, and using them can be tedious and prone to errors.

3.7.1 Functions from the `bst290` package

To make your life easier while you take this course, you can use special functions from the `bst290` package to easily generate the most important descriptive tables you will need:

- `oppsumtabell`: To generate univariate summary tables; this is helpful for numeric variables.
- `oppsum_grupp`: To get a table with summary statistics for one variable, over categories of another variable; this is helpful when you have a a numeric and a categorical variable.

3.7.2 Using oppsumtabell

`oppsumtabell` produces a table with the most important summary statistics of one or more *numeric* variables.[2] All you need to do is specify the dataset that contains your variable(s) and the specific variables you want summary statistics for.

For example, to get summary statistics for the `agea` variable you just run:

```r
oppsumtabell(dataset = ess, variables = "agea")
```

```
Variable        agea
Observations    143.00
Average          47.91
25th percentile  33.50
Median           46.00
75th percentile  62.00
Stand. Dev.      18.57
Minimum          16.00
Maximum          90.00
Missing           0.00
```

To do the same for more than one variable, you run:

```r
oppsumtabell(dataset = ess, variables = c("agea","height","weight"))
```

```
Variable        agea   height weight
Observations    143.00 142.00 138.00
Average          47.91 173.76  78.63
25th percentile  33.50 167.25  65.00
Median           46.00 174.00  75.00
75th percentile  62.00 180.00  88.75
Stand. Dev.      18.57   8.78  19.20
Minimum          16.00 147.00  50.00
Maximum          90.00 196.00 182.00
Missing           0.00   1.00   5.00
```

This table shows summary statistics for age (`agea`) and the respondent's body height and weight.

Can you interpret each of the statistics shown (again, see Kellstedt/Whitten 2018, Chapter 6).

[2]It does also work with factor variables, but you will get a warning message.

3.7.3 Norwegian language support

You can choose to have the table labelled in Norwegian (NB), if you want. All you have to do is to activate the `norsk`-option of the `oppsumtabell()` function and set it to `TRUE` (or `T`):

```r
oppsumtabell(dataset = ess,
             variables = c("agea","height","weight"),
             norsk = TRUE)
```

```
Variabel      agea   height weight
Observasjoner 143.00 142.00 138.00
Gjennomsnitt   47.91 173.76  78.63
25. persentil  33.50 167.25  65.00
Median         46.00 174.00  75.00
75. persentil  62.00 180.00  88.75
Standardavvik  18.57   8.78  19.20
Minimum        16.00 147.00  50.00
Maksimum       90.00 196.00 182.00
Manglende       0.00   1.00   5.00
```

If you take a look at the new version of the table, you will see that all English labels ("standard deviation", "observations") are replaced with their Norwegian equivalents ("standardavvik", "observasjoner").

3.7.4 Exporting the table to Word

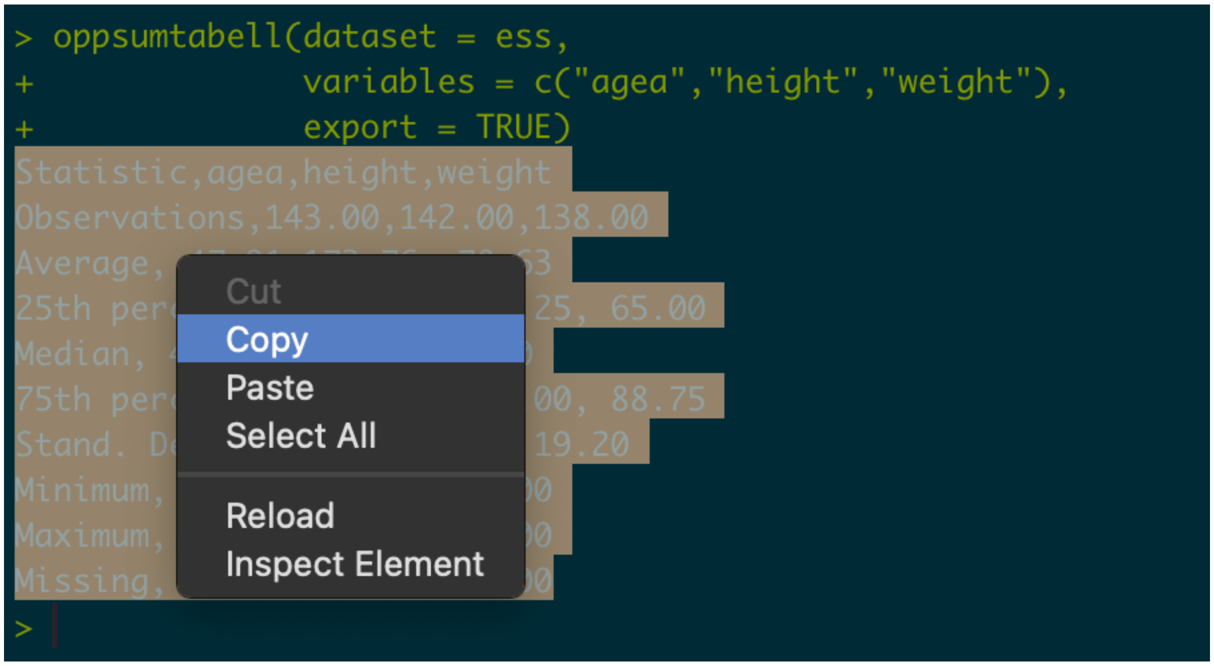
`oppsumtabell` also has an *export*-functionality: You can switch on the export-function to get a result that you can directly copy and paste into a Word document and then transform into a nice, publication-quality table.

For example, to export the last table from above you simply add `export=TRUE` to your code:

```
oppsumtabell(dataset = ess,
             variables = c("agea","height","weight"),
             norsk = TRUE,
             export = TRUE)
Variabel,agea,height,weight
Observasjoner,143.00,142.00,138.00
Gjennomsnitt, 47.91,173.76, 78.63
25. persentil, 33.50,167.25, 65.00
Median, 46.00,174.00, 75.00
75. persentil, 62.00,180.00, 88.75
Standardavvik, 18.57,  8.78, 19.20
Minimum, 16.00,147.00, 50.00
Maksimum, 90.00,196.00,182.00
Manglende,  0.00,  1.00,  5.00
```

This result arguably looks even less presentable than the other one, but:

1. Copy the result as it is displayed in the *Console* (see also the screenshot below);
2. Open a Word document;
3. Paste the copied text into the document;
4. Select the copied text and, in Word, open the ‘Table’ menu in the menu bar at the top; there, select ‘Convert’ and then ‘Convert text to table…’;
5. In the menu, under “Separate text at” (“Skill tekst ved”), select “Other” (“Annet”) and enter a comma into the field next to that option. The number of columns at the top should then also automatically adjust. Then click ‘OK’;
6. Polish the table using the familiar options in Word;



```
> oppsumtabell(dataset = ess,
+              variables = c("agea","height","weight"),
+              export = TRUE)
Statistic,agea,height,weight
Observations,143.00,142.00,138.00
Average,                          3
25th per                         25, 65.00
Median,                          0
75th per                         00, 88.75
Stand. De                        19.20
Minimum,                         00
Maximum,                         00
Missing,                         00
>
```

Figure 1: Selecting & copying the results from the `R` console

3.7.5 Using `oppsum_grupp`

Sometimes you want summary statistics for one variable, but separately for different categories of another variable. For example, assume you are interested in whether (and if yes, by how much) Norwegian men are on average taller than Norwegian women.

The `oppsum_grupp()` function produces a summary table that contains the same statistics as the ones you get from `oppsumtabell()`, but now broken down by categories of a second variable (which should ideally have only a few distinct categories!).

To get summary statistics for body height for men and women separately (i.e., over the categories of `gndr`) you run:

```
oppsum_grupp(dataset = ess, variable = "height", by.var = "gndr")
 gndr   Observations Average Stand. Dev. 25th percentile Median 75th percentile
 Female 67           167.87  6.66        164.00          168.00 173.00
 Male   75           179.03  6.90        174.00          178.00 183.50
 Minimum Maximum Missing
 147.00  180.00  1
 165.00  196.00  0
```

You can see that men are, on average, around 11 centimeters taller than women, and that the smallest woman is smaller than the smallest man (and the same for the tallest individuals in the sample).

Like `oppsumtabell()`, `oppsum_grupp()` also has an export function (`export = TRUE`) and Norwegian language support (`norsk = TRUE`).

3.7.6 Further help

You now know how to get quick summary statistics for a dataset or specific variables in a dataset. Of course, this tutorial covered only the essentials and there are many other ways to summarize your data. But these essentials should help you when you do your first steps as a political or social data analyst.

Also, if you want to get more detailed help on any of the functions covered in this tutorial, you can always resort to the functions' help files. For example, to get the help file for the `mean()` or `oppsumtabell()` functions, you just type the following into your *Console* tab and press Enter:

- `?mean`
- `?oppsumtabell`

The help files also contain examples that show you how to use the functions. Feel free to explore!

4 Importing a real dataset into `R`

Now you know how you can get familiar with a new dataset and do an *exploratory data analysis* (EDA) in `R`. The next step is get your hands on some real data. This is what you learn in this part of the tutorial and, if you like, the appendix.

4.1 The *European Social Survey*

The *European Social Survey (ESS)* is a large survey project that is conducted in countries all over Europe, including in Norway, and which has been running for several years now. In each round, between several hundred to more than 2000 randomly selected persons in each participating country give information (anonymously, of course) about their political opinions and behavior, their views about society, and their income, jobs, work situation, and families. Their responses are then made machine-readable and stored in dataset files, which anyone can use for free.

You can use the *ESS* to study, for example, why people vote or participate otherwise politically (e.g., by joining demonstrations or protests), which parties they voted for, how people think about social inequality, climate change, sexuality, the welfare state, and many other topics. Political scientists and sociologists often use data from the *ESS* in their research.[3]

See https://www.europeansocialsurvey.org/ for more details.

[3]Examples are: Rehm, P. (2009). Risks and redistribution: An individual-level analysis. *Comparative Political Studies*, 42(7):855–81; Giger, N. and Nelson, M. (2013). The welfare state or the economy? Preferences, constituencies, and strategies for retrenchment. *European Sociological Review*, 29(5):1083–94; Hooghe, M., Reeskens, T., Stolle, D., and Trappers, A. (2009). Ethnic diversity and generalized trust in Europe: A cross-national multilevel study. *Comparative Political Studies*, 42(2):198–223; Gallego, A. (2007). Unequal political participation in Europe. *International Journal of Sociology*, 37(4):10–25; or Finseraas, H. (2008). Immigration and preferences for redistribution: An empirical analysis of European survey data. *Comparative European Politics*, 6(4):407–431.

4.1.1 Accessing & downloading *ESS* data

You can access all data from the *ESS* via the SIKT Data Portal: https://ess.sikt.no/en/.[4]

Once you have the page open:

1. Scroll down and choose *ESS Round 7 – 2014. Immigration, Social inequalities in health.*
2. Choose **ESS7 - integrated file, edition 2.3**. You will then be forwarded to another page.
3. Click on the red "Download" button that is shown on the upper right of your screen.
4. You should then be forwarded a login page. Choose *"Logg in med Feide"* and use your UiS credentials to log in. (you may be able to jump over that step if you are already logged on to Feide, e.g., via *Canvas*).
5. Once you are logged in, you will be directed back to the ESS Data Portal – and you will now see three different download buttons (CSV, SPSS, Stata) on the upper right of your screen.
6. Click on the **Stata** button.
7. The data and a few other files will be downloaded as part of a compressed *ZIP* file. Unpack and open that file. The folder that opens will contain one file that ends with `.dta`.[5]
8. **The file ending with `.dta` is the dataset file. Copy/move this file into your project folder (in Windows File Explorer/Mac Finder).** Ideally, give the file a shorter name that is easier to type (e.g., `ess7.dta`).

Once you have your data file stored within your project folder, you can go back to *RStudio*.

Here, you can check if everything worked by opening the *Files* tab in the lower-right corner. The dataset file should be listed here (next to all the other files in this folder).

[4]Alternatively, go to https://www.europeansocialsurvey.org/ and click on "Data" in the menu at the top. On the following page, click on "ESS Data Portal" button.

[5]If you cannot see the file endings ("extensions"), you need to activate this in File Explorer/Finder. You should find instructions for your particular operating system if you google for example "show file extensions in Windows" or "show file extensions in Mac".

4.2 The `haven` package

`R` by itself can open *some* types of dataset files, but not all of them. Among the types of files that `R` itself cannot open are those that were created for other (commercial) data analysis programs:

- `.sav`, the file format for `SPSS`
- `.dta`, the file format for `Stata`
- `.sas7bdat`, the file format for `SAS`

The ESS dataset file you just downloaded is a `.dta` file — which means this dataset is saved in the Stata file format, and `R` by itself cannot open it.

But, luckily, there are a few packages that allow you to import these types of files into `R`. One of these is the `haven` package, and this is the one we will be using in this course.[6]

`haven` is a part of the `tidyverse` collection (see https://haven.tidyverse.org/), which means that you already installed it when you installed the `tidyverse` earlier.

[6]Other alternatives are `foreign`, `memisc`, or `readstata13`.

4.2.1 `haven`'s data import functions

`haven` includes three functions to import the three main "commercial" dataset file formats:

- `read_sav()` for `.sav` files
- `read_dta()` for `.dta` files
- `read_sas()` for `.sas7bdat` files

We will work here with the `read_dta()` function, but the other functions will be important if you ever need to open a dataset that is stored in the *Stata* or *SAS* file formats.

4.2.2 Loading `haven`

Since `haven` is an external package, you first need to load it if you want to use it:

```r
library(haven)
```

4.2.3 Using the `read_dta()` function

The `read_dta()` function is easy to use: You just specify the correct name of the `.dta` file you want to open, and you need to make sure that you store the dataset as an *object* in `R` by using the assignment operator (`<-`).

For example, if your dataset file is named `ess7.dta`, your code to import it into `R` will look like this:

```
ess7 <- read_dta("ess7.dta")
```

This code opens the `ess7.dta` file and stores it as the `ess7`-object in the `R` working environment. Obviously, if your dataset file has a different name, then you need to use that name!

If all worked out, then your dataset file should appear in the *Environment* tab (upper-right corner) under *Data*. There, you will also see that the full ESS dataset includes more than 40.000 observations and more than 600 variables.

4.3 A bit of data cleaning with `labelled`

The `haven` package has a lot of advantages, but also some disadvantages. One disadvantage is that `haven` has its own way of organizing datasets once they are imported into `R`: The `labelled` format.[7]

The `labelled`-format looks and works *a bit* different from the standard format you are now used to. For example, if you scroll a bit down in the *Environment* tab, you will notice a lot of extra gibberish starting with `attr(*,...)`. These are the *attributes* of the variables such as variable and value labels — a bit of extra information about your data.

But if you look carefully, you will also notice that some variables are tagged as `chr+dbl` or `dbl+lbl` — this means "character-type variable with labels" and "numeric variable with labels". These are special formats for variables. If you work more with `R` and the `labelled` format, you will learn that it is not actually difficult to handle — but, for now, it is safer to stick with the standard format that you are more familiar with.

Fortunately, it is possible to quickly convert the dataset from the `labelled` to the traditional format with (yet another...) package. To add to the confusion, this package is also called `labelled`...

Obviously, to be able to use this package and function, you need to quickly install the package now with:

```
install.packages("labelled")
```

[7]See https://larmarange.github.io/labelled/index.html for details.

4.3.1 `labelled::unlabelled()`

Once you have installed the `labelled` package, you can convert the dataset with the `unlabelled()` function:

```
ess7 <- labelled::unlabelled(ess7)
```

If you now scroll down in the *Environment*, you will notice that the tags are back to the ones you know: `chr`, `num`, and `Factor`. But you still have the extra information stored as *attributes*.

Let's do a bit of exploring to show you why the *attributes*-stuff is useful. You can use the `attributes()` function to let `R` print out the attributes associated with a given variable.

For example, to get the attributes of the `gndr` variable, you run:

```
attributes(ess7$gndr)
$levels
[1] "Male"   "Female"

$class
[1] "factor"

$label
[1] "Gender"
```

You directly get some of the most important information about the variable: The variable's label ("Gender"), the category labels ("Male","Female"), and how the variable is stored (as a "factor"; more next week!).

4.3.2 Generating a data dictionary

If you take a quick look at the `ess7` data object in the *Environment*, you notice that it contains 601 variables. Such a large number of variables is typical for a real-life survey dataset, but it also means that it can be difficult to get an overview over all the variables and their values.

Fortunately, there is a function to easily create a *data dictionary* or *codebook* that is included in `labelled`: the `generate_dictionary()` function.

Using this function is easy — you just need to make sure to save the function's output in a new object like `dict_ess7`:

```r
dict_ess7 <- labelled::generate_dictionary(ess7)
```

You will now see a new object in your *Environment* called `dict_ess7`.[8] If you now run `View(dict_ess7)`, you get a neat table that shows you the name, label, and value labels of all the variables in your dataset.

Now you know how you can get survey data for Norway and many other countries on a wide variety of topics from a highly trusted source! Take also a few minutes to explore the ESS website and their Data Portal to see which topics they cover and which variables they have in each survey round!

[8]Technically, the `dict_ess7` dictionary is itself a dataset-type object, which means you can also do some data exploration with it. This goes beyond the scope of this tutorial, but feel free to play around with it.

5 De-bugging exercises

The final part of this tutorial (and the next three) are interactive de-bugging challenges. You will get a set of code 'chunks' that have some problem in them — and your job is to fix these problems.

1. In `RStudio`, navigate to the *Tutorial* tab (upper-right corner of your screen, where the *Environment* tab is).
2. Start the interactive exercise for this tutorial (*"De-bugging exercises: Getting to know your data"*), pop out the window (the little button between the house and red stop button) and maximize, and follow the instructions there.[9]

[9] If there are no tutorials called *"De-bugging exercises:. . . "* shown, just restart `R` by clicking on "Session" in the menu at the top of your screen, and there on "Restart R". You may also have to install the `learnr` package — in that case, `RStudio` will let you know and you only have to do this once.

6 (Voluntary!) Importing other types of datasets

Now you know how to import data from the European Social Survey into `R` and do an exploratory data analysis. But, obviously, the ESS is only one of many, many datasets that you can use.[10] And not all datasets will be available in `SPSS`- or `Stata`-format. Some will only be available Excel (`.xls` or `.xlsx`) files.

In this voluntary extra part of the tutorial, you will learn how you can import datasets that are stored in this format, and also how you can download and import files directly from a website.

[10]See e.g., https://github.com/erikgahner/PolData for an overview over datasets for political and social research.

6.1 The *Comparative Political Data Set*

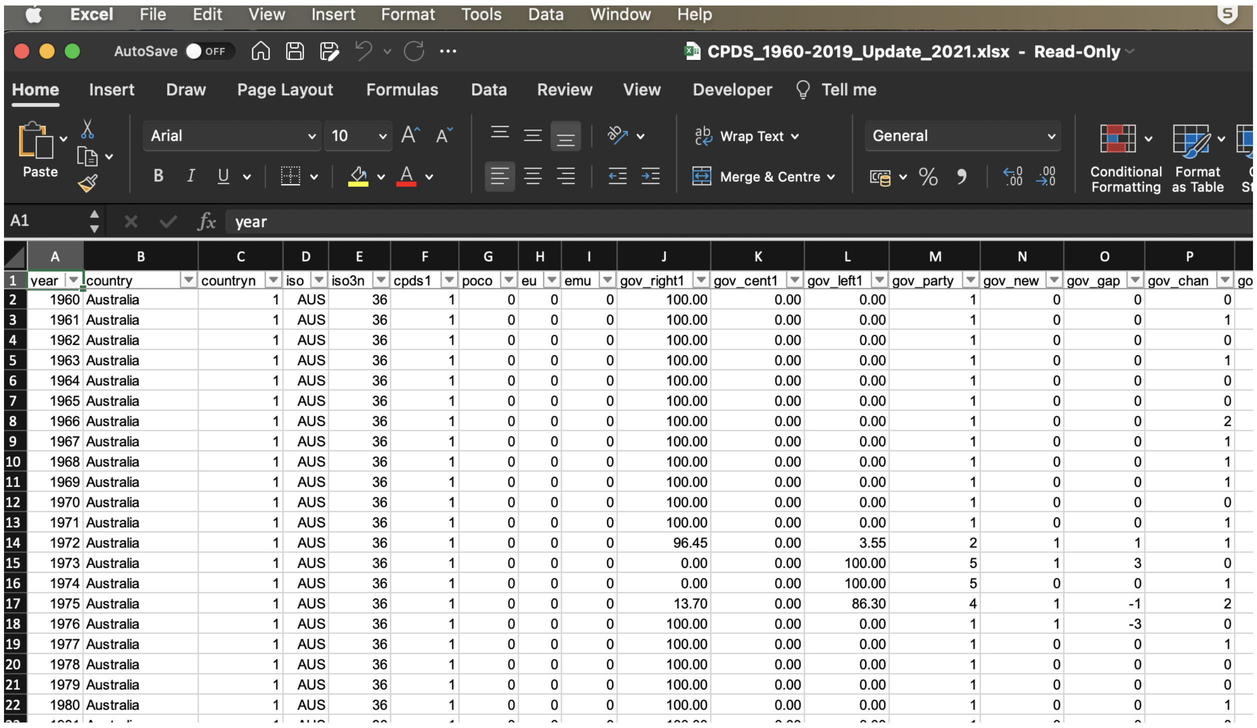
A widely used dataset in political science and sociology is the *Comparative Political Data Set* or CPDS. This dataset is maintained by a team at the University of Berne in Switzerland (see https://cpds-data.org/).

As the name suggests, the CPDS is a *comparative* or *cross-country comparative* dataset: Where the ESS includes data on *individual citizens*, the CPDS includes data on *countries*. With this dataset, you can for example compare countries' political institutions, their party systems, their welfare states and labor market policies, and economic performance. In this dataset, the units of observation are countries — or, more accurately, country-years.

6.2 Taking a look at the CPDS in Excel

To get famililar with the CPDS, you will first take a look at it in a format that you are probably used to: Microsoft Excel. If you go to https://www.cpds-data.org/index.php/data, you see listed under **Data** two links, one to a Stata file and one to an Excel file.

Click on the **Excel** link to download the dataset to your computer. Then open the file in Excel to take a look. You should see something similar to the screenshot below:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	year	country	countryn	iso	iso3n	cpds1	poco	eu	emu	gov_right1	gov_cent1	gov_left1	gov_party	gov_new	gov_gap	gov_chan
2	1960	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	0
3	1961	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	1
4	1962	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	0
5	1963	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	1
6	1964	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	0
7	1965	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	0
8	1966	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	2
9	1967	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	1
10	1968	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	1
11	1969	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	1
12	1970	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	0
13	1971	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	1
14	1972	Australia	1	AUS	36	1	0	0	0	96.45	0.00	3.55	2	1	1	1
15	1973	Australia	1	AUS	36	1	0	0	0	0.00	0.00	100.00	5	1	3	0
16	1974	Australia	1	AUS	36	1	0	0	0	0.00	0.00	100.00	5	0	0	1
17	1975	Australia	1	AUS	36	1	0	0	0	13.70	0.00	86.30	4	1	-1	2
18	1976	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	1	-3	0
19	1977	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	1
20	1978	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	0
21	1979	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	0
22	1980	Australia	1	AUS	36	1	0	0	0	100.00	0.00	0.00	1	0	0	1

Figure 2: The CPDS dataset in Excel

Just lots and lots of boring data. You also see different country names and years — every single row represents a single country in a single year. Therefore, the unit of observation is country-years.

6.3 Importing the CPDS as a Stata (`.dta`) file

If you now go back to the website of the CPDS dataset (https://www.cpds-data.org/index.php/data), you see that there is also a link to download the dataset as a Stata (`.dta`) file. This is the file that you will now import into `R`.

6.3.1 Using `haven` to import the CPDS dataset

Now to the interesting part: Importing the dataset into `R`.

Theoretically, you could now download the `.dta` file to your computer, store it into your working directory, and then import it from there.

But there is also an easier way: You can directly download it using the link on the CPDS website! To download and import the dataset via the link, you need to go back to the CPDS website and copy the link to the `Stata` file (right-click on the link in your browser; "Copy link").

Then you use that link in the `read_dta()` function like this:

```r
cpds <- read_dta("https://www.cpds-data.org/images/Update2021/CPDS_1960-2019_Update_2021.dta")
```

This code does the following:

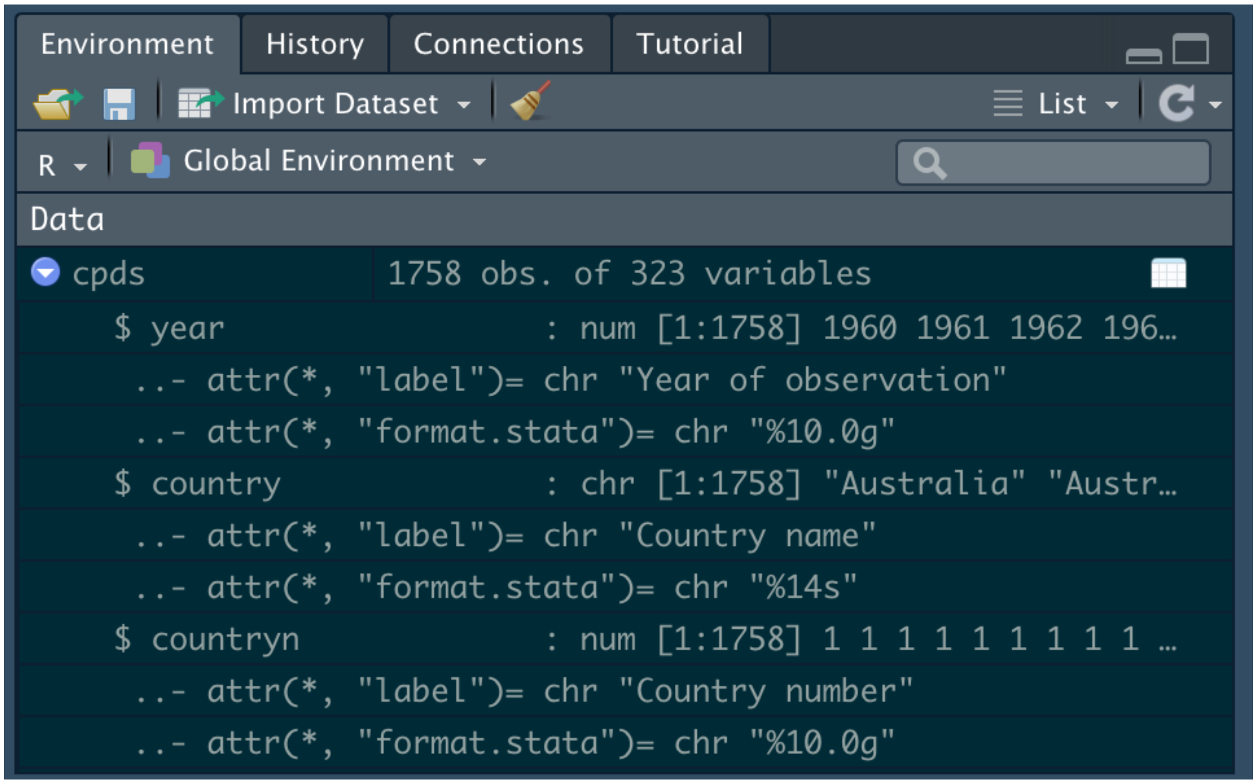
1. `read_dta()` uses the link to download and import the CPDS data file…
2. …and the result — the dataset — gets saved into the `cpds` object with the assignment operator

If all worked, then you should see the `cpds` object appear in your working environment (in the upper-right corner of your screen).

6.3.2 What you see

Just from the information you should see now, you can tell that the `cpds` dataset includes 1759 observations and 323 variables (at the time I'm writing this at least).

As before, you can get more detailed information about the variables included in the dataset if you click on the little blue circle with the white "play" symbol inside (just to the left of `cpds` in the environment). Once you click on this, a list of **variable names** unfolds, for example `year`, `country`, or `iso`.



```
cpds                 1758 obs. of 323 variables
    $ year            : num [1:1758] 1960 1961 1962 196…
     ..- attr(*, "label")= chr "Year of observation"
     ..- attr(*, "format.stata")= chr "%10.0g"
    $ country         : chr [1:1758] "Australia" "Austr…
     ..- attr(*, "label")= chr "Country name"
     ..- attr(*, "format.stata")= chr "%14s"
    $ countryn        : num [1:1758] 1 1 1 1 1 1 1 1 1 …
     ..- attr(*, "label")= chr "Country number"
     ..- attr(*, "format.stata")= chr "%10.0g"
```

Figure 3: The CPDS dataset in `R`

You see that there are different variables (`year`,. . .) and some extra lines reading `attr()`, containing the variables' labels.

You can again look at the data directly using `View(cpds)` — it should look basically as it did in Excel, only that you get more informative labels for your variables.

6.4 Importing Excel (`.xls`/`.xlsx`) files

Excel is widely used and there is therefore a way to import Excel dataset files into `R`. To do this, you need to install yet another package: `readxl` ("read Excel" — the last symbol is a small L, not a 1).

You already know how to install and load packages, so I trust that you can install the `readxl` package by yourself.

6.4.1 Importing the CPDS dataset as an Excel file

Earlier you downloaded and opened the CPDS dataset as an Excel file — now you will import that file into `R`.

Important: Unlike `haven`, `readxl` cannot download files directly from the internet. This means that if you want to import an Excel file with `readxl`, you first have to download it to your computer and then open the file from there.

The easiest way to do this is to save the CPDS Excel file into your `R` project folder (the one you created in the first tutorial).

Once you have the CPDS Excel-file stored in your project folder, you can import it with `readxl`.

Specifically, you use the `read_excel()` function, which can import both the older `.xls` and the newer `.xlsx` formats. You use `read_excel()` similarly to how you use `read_dta()` from `haven`.

For example, with the CPDS data file this would look like this:

```
cpds <- read_excel("CPDS_1960-2019_Update_2021.xlsx")
```

Once you run the function, the dataset should appear in your *Environment* tab. Most likely, it will look a bit different compared to when the dataset was imported with `haven`, but the dataset itself will be the same.

There are many different ways to read data from Excel files — you can read specific *sheets* from a larger file, or also only parts of a sheet. See the `readxl` documentation for details: https://readxl.tidyverse.org/

6.4.2 De-bugging exercises

There are also some de-bugging exercises specifically for this last voluntary part of the data-import tutorial. You can access these via *Tutorials* ("Getting your data into R").