Africa CDC R training

Dave Kennedy (UK Public Health Rapid Support Team)

22/06/2021

## Introduction

* Course structure
* Why R?
* Installing R
* Setting up files/folders
* R packages
* Importing data
* Useful resources

## Course structure

Presentations built around content developed for the [Epidemiologist R handbook](https://epirhandbook.com/)

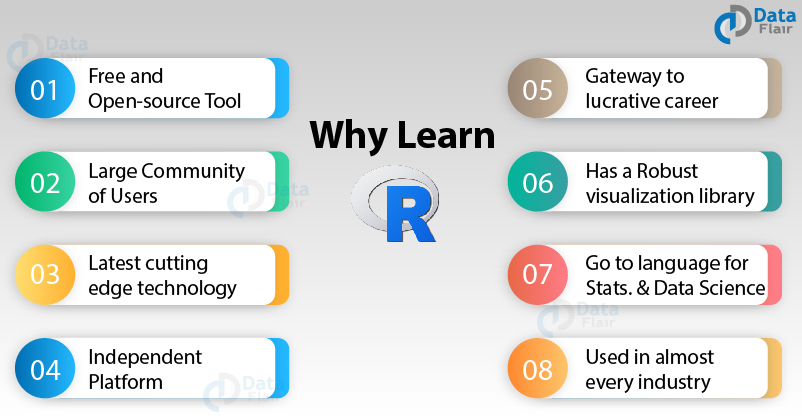
* An excellent resource for all skill/experience levels
* Direct towards specific sections for you to work through in your own time
* 2 hours sessions, twice a week to present key topics and answer questions

Capstone project

- To apply R skills to data you regularly use

- Present on the challenges and enablers

## Why R?

[](https://data-flair.training/blogs/why-learn-r/)

## Installing R

For this course, you will need to install 2 items:

1. R programming language
2. R Studio
   * Integrated Development Environment (IDE)
   * A very helpful resource for writing and running R code

* You will need to install them in this order - First R, then R studio

[Guide to installing R](https://www.earthdatascience.org/courses/earth-analytics/document-your-science/setup-r-rstudio/)

## Setting up files and folders



“Massive Wall of Organized Documents” by Zeusandhera is licensed with CC BY-SA 2.0. To view a copy of this license, visit <https://creativecommons.org/licenses/by-sa/2.0/>

## Setting up files and folders - Best practice

Setting up files and folders will make your analysis (and life!) easier

* Folder structure
* Naming files and folders

## Setting up files and folders - Folder structure

R Studio works best when you use its project function

* Each project contains all of your inputs, outputs and code
* This also makes it easier to share folders with colleagues
  + Everything is in one place!

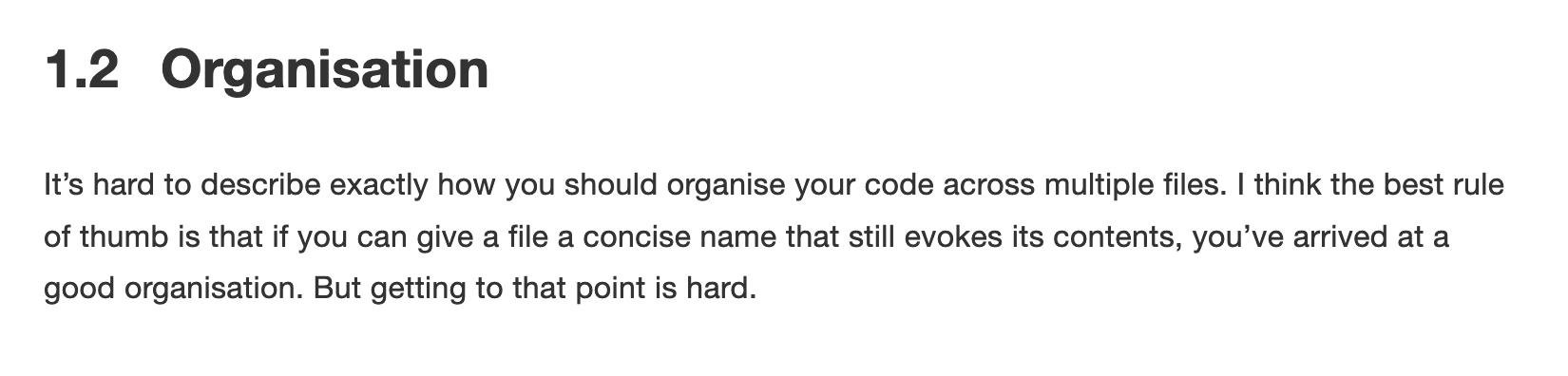
Projects are covered in more detail in [Chapter 6 “R Projects”](https://epirhandbook.com/r-projects.html)

## Setting up files and folders - Naming files and folders

If you want to share your code with colleagues or when you return to code after several weeks/months you will be grateful that you gave your files and folders meaningful names!

Many organisations have style guides to ensure that teams can collaborate on coding projects

* [Tidyverse style guide](https://style.tidyverse.org/files.html#names)

[](https://style.tidyverse.org/files.html#names)

* Key points to remember for naming
  + Keep the name short
    - Instead of “*data\_import\_of\_file\_for\_analysis.R*”
      * “*import\_file.R*”
  + Avoid spaces!
    - Instead of “*import file.R*”
      * “\_” “*import\_file.R*”
      * “-” “*import-file.R*”
      * camelCase “*importFile.R*”

## R packages

What is an R package?

*An R package is a collection of functions which you can use to import, clean, analyse and report your data*

Link to Epidemiologist R handbook - 3.7 “[Packages](https://epirhandbook.com/r-basics.html#packages)”

Packages can simplify your workflow by combining multiple steps into a smaller number of commands

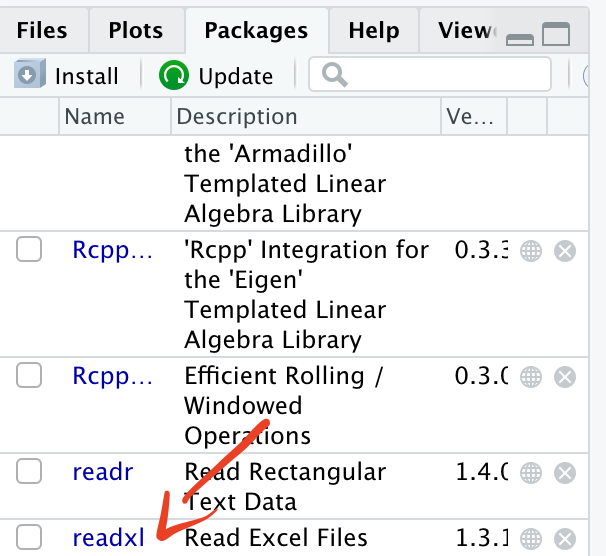
Example: [readxl](https://readxl.tidyverse.org/) is a package of functions used to import data from Excel to R.

## R packages - Installing a package

install.packages(“readxl”)

We have asked R to install the package “readxl”.

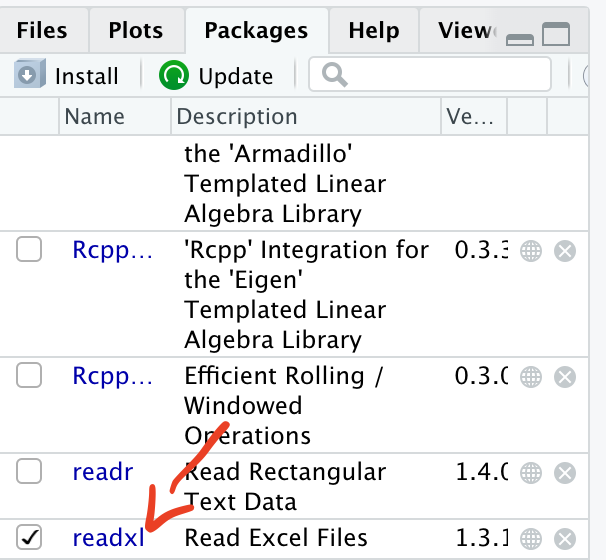
The installation has been successful. The installation only has to take place once. You do not need to re install the packages every time you start a new project as they are saved in your library.

* In R, red text does not mean there has been an error!
* You will now be able to see the package in your list of packages
* 

## R packages - Loading a package

Now that readxl has been installed, you will be able to load it and use its functions

library(readxl)



When the package has been successfully loaded, you will see a tick mark in the box.

## R packages - Using a package

Each package has multiple functions that you can use on your data.

To read more about a particular package type

?readxl



Help documentation for readxl

For this example, we want to import data that is currently stored in an Excel formatted file “.xlsx”

So we can use the function *read\_xlsx f*rom the readxl package

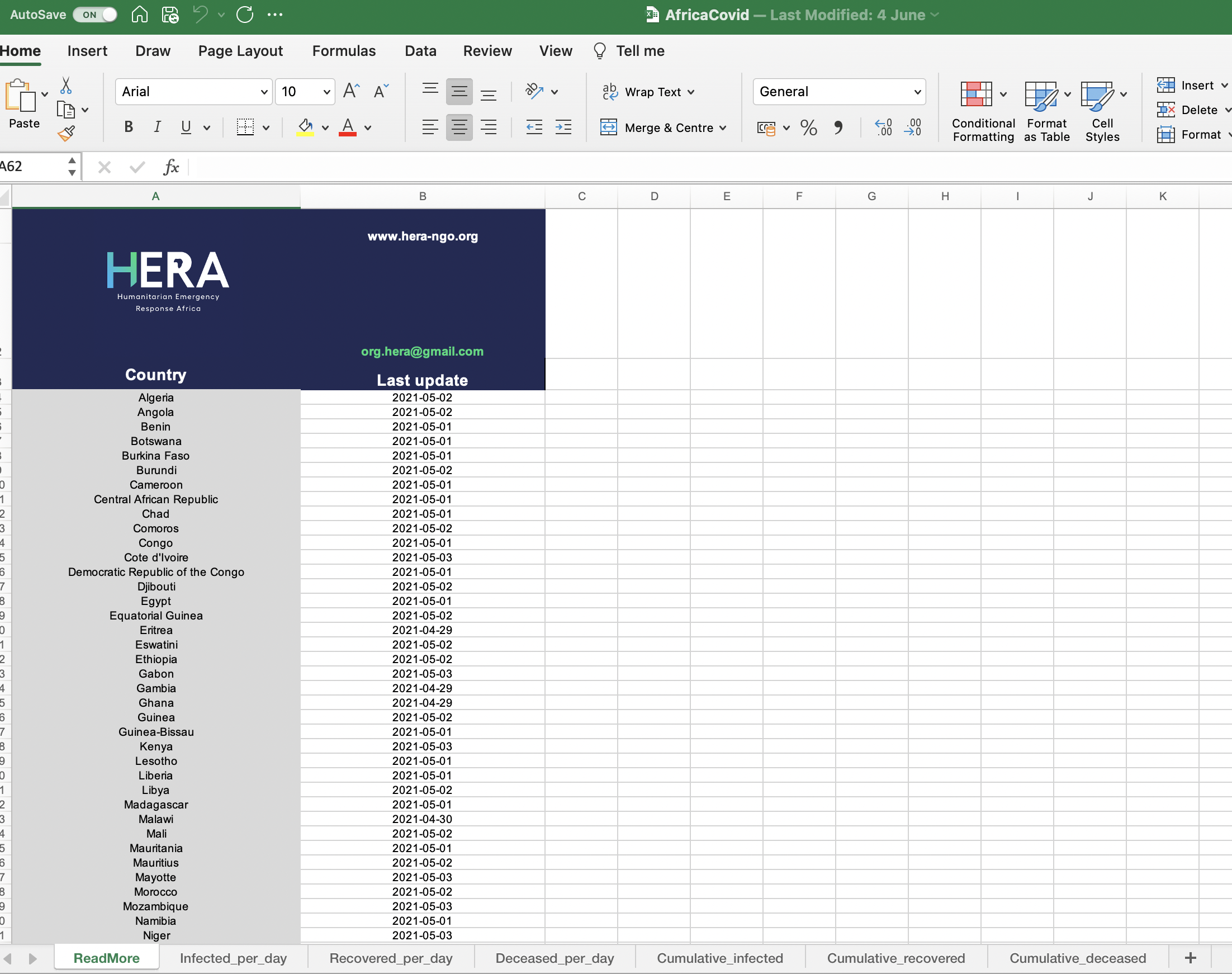
read\_xlsx(here('data','AfricaCovid','AfricaCovid.xlsx'))

## New names:  
## \* `` -> ...1

## # A tibble: 58 x 2  
## ...1 `www.hera-ngo.org`  
## <chr> <chr>   
## 1 <NA> org.hera@gmail.com  
## 2 Country Last update   
## 3 Algeria 44318   
## 4 Angola 44318   
## 5 Benin 44317   
## 6 Botswana 44317   
## 7 Burkina Faso 44317   
## 8 Burundi 44318   
## 9 Cameroon 44317   
## 10 Central African Republic 44317   
## # … with 48 more rows

But what does this show? And how can we use it?

* Before importing a file from Excel, it may be helpful to open it in Excel so we can see what data are stored in the file



So when we tell R to use the function read\_xlsx, it reads the first sheet which is called “ReadMore”.

It looks like this is a summary sheet with information about when data for each country was last updated.

So how do we tell R to read in a different sheet from the Excel file?

## R packages - Using a package 4

Question - How many confirmed cases of COVID were recorded across Africa in July 2020?

First step - Import data from the sheet containing information on COVID cases

We can use the excel\_sheets function from readxl to get the names of all sheets in the Excel workbook

excel\_sheets(here('data','AfricaCovid','AfricaCovid.xlsx'))

## [1] "ReadMore" "Infected\_per\_day" "Recovered\_per\_day"   
## [4] "Deceased\_per\_day" "Cumulative\_infected" "Cumulative\_recovered"  
## [7] "Cumulative\_deceased" "SDN FLore" "GHA Flore"   
## [10] "SLE Flore" "ZAF Flore"

From this list we can see that we want to import data from the sheet “Infected\_per\_day”.

read\_xlsx(here('data','AfricaCovid','AfricaCovid.xlsx'), sheet="Infected\_per\_day")

## # A tibble: 53 x 492  
## ISO COUNTRY\_NAME AFRICAN\_REGION `43831` `43832` `43833` `43834` `43835`  
## <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 DZA Algeria Northern Afri… 0 0 0 0 0  
## 2 AGO Angola Southern Afri… 0 0 0 0 0  
## 3 BEN Benin Western Africa 0 0 0 0 0  
## 4 BWA Botswana Southern Afri… 0 0 0 0 0  
## 5 BFA Burkina Faso Western Africa 0 0 0 0 0  
## 6 BDI Burundi Central Africa 0 0 0 0 0  
## 7 CMR Cameroon Central Africa 0 0 0 0 0  
## 8 CAR Central African… Central Africa 0 0 0 0 0  
## 9 TCD Chad Central Africa 0 0 0 0 0  
## 10 COM Comoros Eastern Africa 0 0 0 0 0  
## # … with 43 more rows, and 484 more variables: 43836 <dbl>, 43837 <dbl>,  
## # 43838 <dbl>, 43839 <dbl>, 43840 <dbl>, 43841 <dbl>, 43842 <dbl>,  
## # 43843 <dbl>, 43844 <dbl>, 43845 <dbl>, 43846 <dbl>, 43847 <dbl>,  
## # 43848 <dbl>, 43849 <dbl>, 43850 <dbl>, 43851 <dbl>, 43852 <dbl>,  
## # 43853 <dbl>, 43854 <dbl>, 43855 <dbl>, 43856 <dbl>, 43857 <dbl>,  
## # 43858 <dbl>, 43859 <dbl>, 43860 <dbl>, 43861 <dbl>, 43862 <dbl>,  
## # 43863 <dbl>, 43864 <dbl>, 43865 <dbl>, 43866 <dbl>, 43867 <dbl>,  
## # 43868 <dbl>, 43869 <dbl>, 43870 <dbl>, 43871 <dbl>, 43872 <dbl>,  
## # 43873 <dbl>, 43874 <dbl>, 43875 <dbl>, 43876 <dbl>, 43877 <dbl>,  
## # 43878 <dbl>, 43879 <dbl>, 43880 <dbl>, 43881 <dbl>, 43882 <dbl>,  
## # 43883 <dbl>, 43884 <dbl>, 43885 <dbl>, 43886 <dbl>, 43887 <dbl>,  
## # 43888 <dbl>, 43889 <dbl>, 43890 <dbl>, 43891 <dbl>, 43892 <dbl>,  
## # 43893 <dbl>, 43894 <dbl>, 43895 <dbl>, 43896 <dbl>, 43897 <dbl>,  
## # 43898 <dbl>, 43899 <dbl>, 43900 <dbl>, 43901 <dbl>, 43902 <dbl>,  
## # 43903 <dbl>, 43904 <dbl>, 43905 <dbl>, 43906 <dbl>, 43907 <dbl>,  
## # 43908 <dbl>, 43909 <dbl>, 43910 <dbl>, 43911 <dbl>, 43912 <dbl>,  
## # 43913 <dbl>, 43914 <dbl>, 43915 <dbl>, 43916 <dbl>, 43917 <dbl>,  
## # 43918 <dbl>, 43919 <dbl>, 43920 <dbl>, 43921 <dbl>, 43922 <dbl>,  
## # 43923 <dbl>, 43924 <dbl>, 43925 <dbl>, 43926 <dbl>, 43927 <dbl>,  
## # 43928 <dbl>, 43929 <dbl>, 43930 <dbl>, 43931 <dbl>, 43932 <dbl>,  
## # 43933 <dbl>, 43934 <dbl>, 43935 <dbl>, …

We can see a snapshot of the data from the sheet “Infected\_per\_day”

## Objects - In R, everything is an object

So far we have installed, loaded and used a package (readxl)

But how do we use the information generated from these actions?

We assign the information to “objects”

[Section in Epidemiologist for R handbook about Objects](https://epirhandbook.com/r-basics.html#objects)

*“Everything you store in R - datasets, variables, a list of village names, a total population number, even outputs such as graphs - are* ***objects*** *which are* ***assigned a name*** *and* ***can be referenced*** *in later commands.”*

## Objects - Example

To explain objects, we will calculate a value, assign it to an object and then use the object for a second calculation.

2+2

## [1] 4

We can assign the calculation “2+2” to an object called “a”

a <- 2+2

We can then use the object a to show the results of the calculation

a

## [1] 4

We can also use this value for further calculations such as adding 4 to the object a

a + 4

## [1] 8

b <- a+4

The result of this calculation is now stored in the object “b”

b

## [1] 8

## Objects: Assigning our data to an object

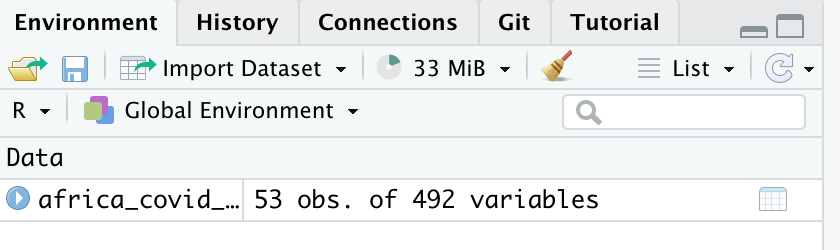
In the previous section, we used the function read\_xlsx from the package readxl to import data from an Excel spreadsheet.

But we didn’t assign this to an object, so it is not possible to use the data from the import step.

We can assign the data to an object and then conduct further analysis.

africa\_covid\_cases <- read\_xlsx(here('data','AfricaCovid','AfricaCovid.xlsx'), sheet="Infected\_per\_day")

You will now see the object in the “Environment” section of R Studio.



Now the data have been assigned to the object “africa\_covid\_cases”, we can start to work with the data.

## Working with data

In the africa\_covid\_cases object, there are 53 obs (observations) of 492 variables.

So what does this mean?

We can look at our data to get more information

africa\_covid\_cases

## # A tibble: 53 x 492  
## ISO COUNTRY\_NAME AFRICAN\_REGION `43831` `43832` `43833` `43834` `43835`  
## <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 DZA Algeria Northern Afri… 0 0 0 0 0  
## 2 AGO Angola Southern Afri… 0 0 0 0 0  
## 3 BEN Benin Western Africa 0 0 0 0 0  
## 4 BWA Botswana Southern Afri… 0 0 0 0 0  
## 5 BFA Burkina Faso Western Africa 0 0 0 0 0  
## 6 BDI Burundi Central Africa 0 0 0 0 0  
## 7 CMR Cameroon Central Africa 0 0 0 0 0  
## 8 CAR Central African… Central Africa 0 0 0 0 0  
## 9 TCD Chad Central Africa 0 0 0 0 0  
## 10 COM Comoros Eastern Africa 0 0 0 0 0  
## # … with 43 more rows, and 484 more variables: 43836 <dbl>, 43837 <dbl>,  
## # 43838 <dbl>, 43839 <dbl>, 43840 <dbl>, 43841 <dbl>, 43842 <dbl>,  
## # 43843 <dbl>, 43844 <dbl>, 43845 <dbl>, 43846 <dbl>, 43847 <dbl>,  
## # 43848 <dbl>, 43849 <dbl>, 43850 <dbl>, 43851 <dbl>, 43852 <dbl>,  
## # 43853 <dbl>, 43854 <dbl>, 43855 <dbl>, 43856 <dbl>, 43857 <dbl>,  
## # 43858 <dbl>, 43859 <dbl>, 43860 <dbl>, 43861 <dbl>, 43862 <dbl>,  
## # 43863 <dbl>, 43864 <dbl>, 43865 <dbl>, 43866 <dbl>, 43867 <dbl>,  
## # 43868 <dbl>, 43869 <dbl>, 43870 <dbl>, 43871 <dbl>, 43872 <dbl>,  
## # 43873 <dbl>, 43874 <dbl>, 43875 <dbl>, 43876 <dbl>, 43877 <dbl>,  
## # 43878 <dbl>, 43879 <dbl>, 43880 <dbl>, 43881 <dbl>, 43882 <dbl>,  
## # 43883 <dbl>, 43884 <dbl>, 43885 <dbl>, 43886 <dbl>, 43887 <dbl>,  
## # 43888 <dbl>, 43889 <dbl>, 43890 <dbl>, 43891 <dbl>, 43892 <dbl>,  
## # 43893 <dbl>, 43894 <dbl>, 43895 <dbl>, 43896 <dbl>, 43897 <dbl>,  
## # 43898 <dbl>, 43899 <dbl>, 43900 <dbl>, 43901 <dbl>, 43902 <dbl>,  
## # 43903 <dbl>, 43904 <dbl>, 43905 <dbl>, 43906 <dbl>, 43907 <dbl>,  
## # 43908 <dbl>, 43909 <dbl>, 43910 <dbl>, 43911 <dbl>, 43912 <dbl>,  
## # 43913 <dbl>, 43914 <dbl>, 43915 <dbl>, 43916 <dbl>, 43917 <dbl>,  
## # 43918 <dbl>, 43919 <dbl>, 43920 <dbl>, 43921 <dbl>, 43922 <dbl>,  
## # 43923 <dbl>, 43924 <dbl>, 43925 <dbl>, 43926 <dbl>, 43927 <dbl>,  
## # 43928 <dbl>, 43929 <dbl>, 43930 <dbl>, 43931 <dbl>, 43932 <dbl>,  
## # 43933 <dbl>, 43934 <dbl>, 43935 <dbl>, …

ISO - 3 letter code assigned to each country

COUNTRY\_NAME - Name of the country

AFRICAN\_REGION - African region

43831, 43832, 43833 - This looks like a date format used by Excel. It is the number of days since January 1, 1970.

## Working with data - other ways to look at data

Show the first 5 rows

head(africa\_covid\_cases, n=5)

## # A tibble: 5 x 492  
## ISO COUNTRY\_NAME AFRICAN\_REGION `43831` `43832` `43833` `43834` `43835`  
## <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 DZA Algeria Northern Africa 0 0 0 0 0  
## 2 AGO Angola Southern Africa 0 0 0 0 0  
## 3 BEN Benin Western Africa 0 0 0 0 0  
## 4 BWA Botswana Southern Africa 0 0 0 0 0  
## 5 BFA Burkina Faso Western Africa 0 0 0 0 0  
## # … with 484 more variables: 43836 <dbl>, 43837 <dbl>, 43838 <dbl>,  
## # 43839 <dbl>, 43840 <dbl>, 43841 <dbl>, 43842 <dbl>, 43843 <dbl>,  
## # 43844 <dbl>, 43845 <dbl>, 43846 <dbl>, 43847 <dbl>, 43848 <dbl>,  
## # 43849 <dbl>, 43850 <dbl>, 43851 <dbl>, 43852 <dbl>, 43853 <dbl>,  
## # 43854 <dbl>, 43855 <dbl>, 43856 <dbl>, 43857 <dbl>, 43858 <dbl>,  
## # 43859 <dbl>, 43860 <dbl>, 43861 <dbl>, 43862 <dbl>, 43863 <dbl>,  
## # 43864 <dbl>, 43865 <dbl>, 43866 <dbl>, 43867 <dbl>, 43868 <dbl>,  
## # 43869 <dbl>, 43870 <dbl>, 43871 <dbl>, 43872 <dbl>, 43873 <dbl>,  
## # 43874 <dbl>, 43875 <dbl>, 43876 <dbl>, 43877 <dbl>, 43878 <dbl>,  
## # 43879 <dbl>, 43880 <dbl>, 43881 <dbl>, 43882 <dbl>, 43883 <dbl>,  
## # 43884 <dbl>, 43885 <dbl>, 43886 <dbl>, 43887 <dbl>, 43888 <dbl>,  
## # 43889 <dbl>, 43890 <dbl>, 43891 <dbl>, 43892 <dbl>, 43893 <dbl>,  
## # 43894 <dbl>, 43895 <dbl>, 43896 <dbl>, 43897 <dbl>, 43898 <dbl>,  
## # 43899 <dbl>, 43900 <dbl>, 43901 <dbl>, 43902 <dbl>, 43903 <dbl>,  
## # 43904 <dbl>, 43905 <dbl>, 43906 <dbl>, 43907 <dbl>, 43908 <dbl>,  
## # 43909 <dbl>, 43910 <dbl>, 43911 <dbl>, 43912 <dbl>, 43913 <dbl>,  
## # 43914 <dbl>, 43915 <dbl>, 43916 <dbl>, 43917 <dbl>, 43918 <dbl>,  
## # 43919 <dbl>, 43920 <dbl>, 43921 <dbl>, 43922 <dbl>, 43923 <dbl>,  
## # 43924 <dbl>, 43925 <dbl>, 43926 <dbl>, 43927 <dbl>, 43928 <dbl>,  
## # 43929 <dbl>, 43930 <dbl>, 43931 <dbl>, 43932 <dbl>, 43933 <dbl>,  
## # 43934 <dbl>, 43935 <dbl>, …

Show the last 7 rows

tail(africa\_covid\_cases, n=7)

## # A tibble: 7 x 492  
## ISO COUNTRY\_NAME AFRICAN\_REGION `43831` `43832` `43833` `43834` `43835`  
## <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 SDN Sudan Eastern Africa 0 0 0 0 0  
## 2 TZA Tanzania Eastern Africa 0 0 0 0 0  
## 3 TGO Togo Western Africa 0 0 0 0 0  
## 4 TUN Tunisia Northern Africa 0 0 0 0 0  
## 5 UGA Uganda Eastern Africa 0 0 0 0 0  
## 6 ZMB Zambia Southern Africa 0 0 0 0 0  
## 7 ZWE Zimbabwe Southern Africa 0 0 0 0 0  
## # … with 484 more variables: 43836 <dbl>, 43837 <dbl>, 43838 <dbl>,  
## # 43839 <dbl>, 43840 <dbl>, 43841 <dbl>, 43842 <dbl>, 43843 <dbl>,  
## # 43844 <dbl>, 43845 <dbl>, 43846 <dbl>, 43847 <dbl>, 43848 <dbl>,  
## # 43849 <dbl>, 43850 <dbl>, 43851 <dbl>, 43852 <dbl>, 43853 <dbl>,  
## # 43854 <dbl>, 43855 <dbl>, 43856 <dbl>, 43857 <dbl>, 43858 <dbl>,  
## # 43859 <dbl>, 43860 <dbl>, 43861 <dbl>, 43862 <dbl>, 43863 <dbl>,  
## # 43864 <dbl>, 43865 <dbl>, 43866 <dbl>, 43867 <dbl>, 43868 <dbl>,  
## # 43869 <dbl>, 43870 <dbl>, 43871 <dbl>, 43872 <dbl>, 43873 <dbl>,  
## # 43874 <dbl>, 43875 <dbl>, 43876 <dbl>, 43877 <dbl>, 43878 <dbl>,  
## # 43879 <dbl>, 43880 <dbl>, 43881 <dbl>, 43882 <dbl>, 43883 <dbl>,  
## # 43884 <dbl>, 43885 <dbl>, 43886 <dbl>, 43887 <dbl>, 43888 <dbl>,  
## # 43889 <dbl>, 43890 <dbl>, 43891 <dbl>, 43892 <dbl>, 43893 <dbl>,  
## # 43894 <dbl>, 43895 <dbl>, 43896 <dbl>, 43897 <dbl>, 43898 <dbl>,  
## # 43899 <dbl>, 43900 <dbl>, 43901 <dbl>, 43902 <dbl>, 43903 <dbl>,  
## # 43904 <dbl>, 43905 <dbl>, 43906 <dbl>, 43907 <dbl>, 43908 <dbl>,  
## # 43909 <dbl>, 43910 <dbl>, 43911 <dbl>, 43912 <dbl>, 43913 <dbl>,  
## # 43914 <dbl>, 43915 <dbl>, 43916 <dbl>, 43917 <dbl>, 43918 <dbl>,  
## # 43919 <dbl>, 43920 <dbl>, 43921 <dbl>, 43922 <dbl>, 43923 <dbl>,  
## # 43924 <dbl>, 43925 <dbl>, 43926 <dbl>, 43927 <dbl>, 43928 <dbl>,  
## # 43929 <dbl>, 43930 <dbl>, 43931 <dbl>, 43932 <dbl>, 43933 <dbl>,  
## # 43934 <dbl>, 43935 <dbl>, …

How many unique countries are in the data?

unique(africa\_covid\_cases$COUNTRY\_NAME)

## [1] "Algeria" "Angola"   
## [3] "Benin" "Botswana"   
## [5] "Burkina Faso" "Burundi"   
## [7] "Cameroon" "Central African Republic"   
## [9] "Chad" "Comoros"   
## [11] "Congo" "Cote d'Ivoire"   
## [13] "Democratic Republic of the Congo" "Djibouti"   
## [15] "Egypt" "Equatorial Guinea"   
## [17] "Eritrea" "Eswatini"   
## [19] "Ethiopia" "Gabon"   
## [21] "Gambia" "Ghana"   
## [23] "Guinea" "Guinea-Bissau"   
## [25] "Kenya" "Lesotho"   
## [27] "Liberia" "Libya"   
## [29] "Madagascar" "Malawi"   
## [31] "Mali" "Mauritania"   
## [33] "Mauritius" "Mayotte"   
## [35] "Morocco" "Mozambique"   
## [37] "Namibia" "Niger"   
## [39] "Nigeria" "Rwanda"   
## [41] "Sao Tome and Principe" "Senegal"   
## [43] "Sierra Leone" "Somalia"   
## [45] "South Africa" "South Sudan"   
## [47] "Sudan" "Tanzania"   
## [49] "Togo" "Tunisia"   
## [51] "Uganda" "Zambia"   
## [53] "Zimbabwe"

There are 53 unique country values. This is helpful as there are also 53 rows so we can say that each row represents a country.

We can assign the list of unique countries to an object for future reference

country\_list <- unique(africa\_covid\_cases$COUNTRY\_NAME)

## Working with data - looking at one variable

In the previous step, the following command was used

unique(africa\_covid\_cases$COUNTRY\_NAME)

What does “$” do in R?

It allows us to look at a specific variable within the dataset

unique(africa\_covid\_cases$AFRICAN\_REGION)

## [1] "Northern Africa" "Southern Africa" "Western Africa" "Central Africa"   
## [5] "Eastern Africa"

And again we can assign this to an object

region\_list <- unique(africa\_covid\_cases$AFRICAN\_REGION)

## The tidyverse

When using R, there are many approaches you can use to reach the same result.

There are thousands of packages with many functions and sometimes these packages can overlap.

This can be confusing when you are starting to learn R.

There is a collection of packages with many of the most commonly used packages and this is called the [tidyverse](https://www.tidyverse.org/).

tidyverse::tidyverse\_packages()

## [1] "broom" "cli" "crayon" "dbplyr"   
## [5] "dplyr" "dtplyr" "forcats" "googledrive"   
## [9] "googlesheets4" "ggplot2" "haven" "hms"   
## [13] "httr" "jsonlite" "lubridate" "magrittr"   
## [17] "modelr" "pillar" "purrr" "readr"   
## [21] "readxl" "reprex" "rlang" "rstudioapi"   
## [25] "rvest" "stringr" "tibble" "tidyr"   
## [29] "xml2" "tidyverse"

We will use functions from some of these packages over the next few sessions.

## The tidyverse: Tidy data

The key concept when working with packages from the tidyverse is the concept of “tidy data”.

[R for Epidemiologist handbook 4.1 From Excel - Tidy data](https://epirhandbook.com/transition-to-r.html?q=tidy#tidy-data)

Principles of “[tidy data](https://r4ds.had.co.nz/tidy-data.html)”:

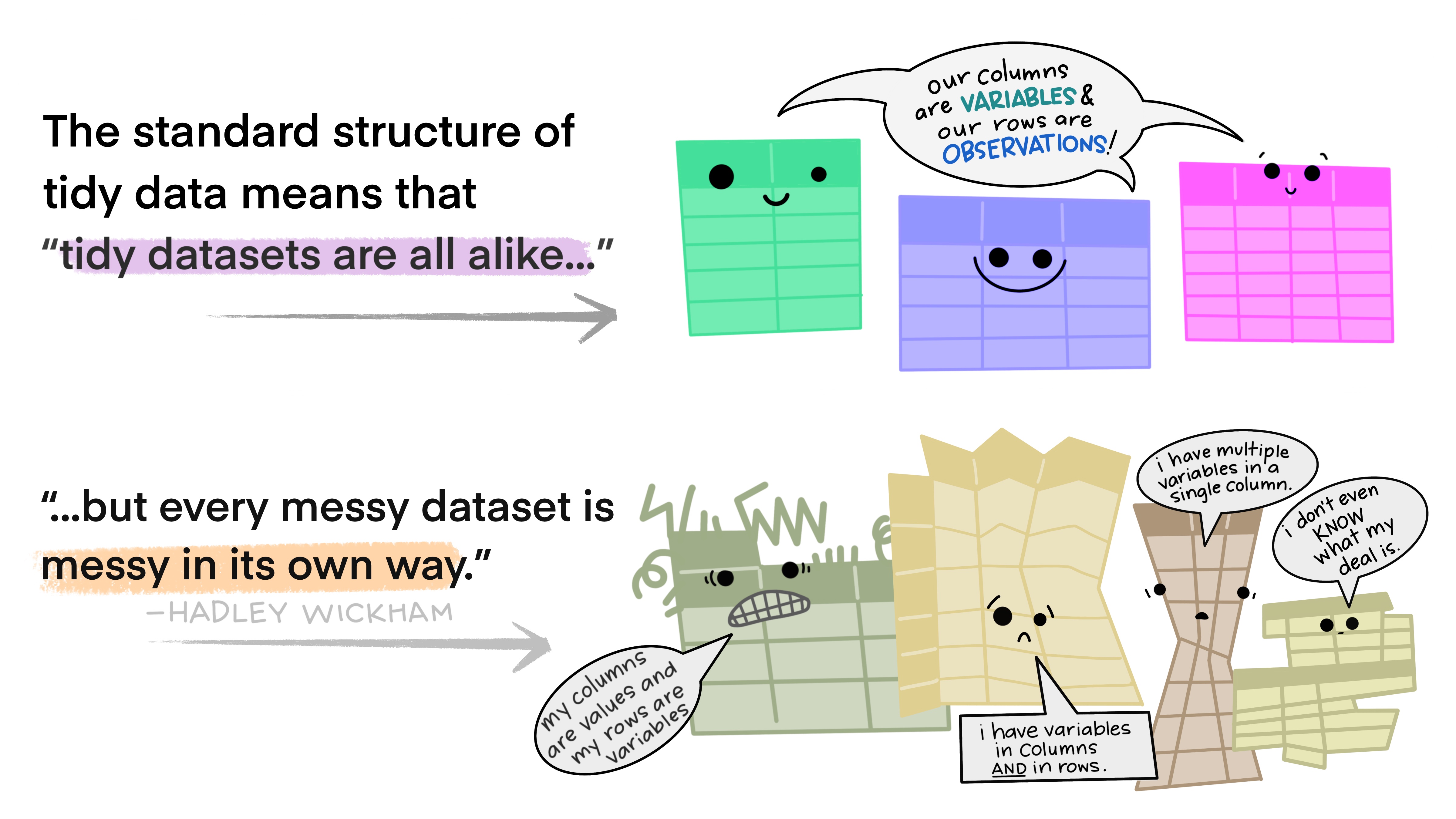
1. Each variable must have its own column
2. Each observation must have its own row
3. Each value must have its own cell

## The tidyverse: Why is this important?

Functions from the tidyverse packages are set up to work with tidy data.

If your data are not tidy, then you will have to restructure the data to a tidy format.

Restructuring can take a lot of time if the data are stored in Excel spreadsheets with a lot of formatting/merged columns.

[](https://www.openscapes.org/blog/2020/10/12/tidy-data/)

## The tidyverse: Checking if data are tidy

In a previous step, we imported COVID case data from an Excel spreadsheet.  
But how do we know if the data are “tidy”

Remember there are 3 principles

1. Each variable must have its own column
2. Each observation must have its own row
3. Each value must have its own cell

head(africa\_covid\_cases, n=3)

## # A tibble: 3 x 492  
## ISO COUNTRY\_NAME AFRICAN\_REGION `43831` `43832` `43833` `43834` `43835`  
## <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 DZA Algeria Northern Africa 0 0 0 0 0  
## 2 AGO Angola Southern Africa 0 0 0 0 0  
## 3 BEN Benin Western Africa 0 0 0 0 0  
## # … with 484 more variables: 43836 <dbl>, 43837 <dbl>, 43838 <dbl>,  
## # 43839 <dbl>, 43840 <dbl>, 43841 <dbl>, 43842 <dbl>, 43843 <dbl>,  
## # 43844 <dbl>, 43845 <dbl>, 43846 <dbl>, 43847 <dbl>, 43848 <dbl>,  
## # 43849 <dbl>, 43850 <dbl>, 43851 <dbl>, 43852 <dbl>, 43853 <dbl>,  
## # 43854 <dbl>, 43855 <dbl>, 43856 <dbl>, 43857 <dbl>, 43858 <dbl>,  
## # 43859 <dbl>, 43860 <dbl>, 43861 <dbl>, 43862 <dbl>, 43863 <dbl>,  
## # 43864 <dbl>, 43865 <dbl>, 43866 <dbl>, 43867 <dbl>, 43868 <dbl>,  
## # 43869 <dbl>, 43870 <dbl>, 43871 <dbl>, 43872 <dbl>, 43873 <dbl>,  
## # 43874 <dbl>, 43875 <dbl>, 43876 <dbl>, 43877 <dbl>, 43878 <dbl>,  
## # 43879 <dbl>, 43880 <dbl>, 43881 <dbl>, 43882 <dbl>, 43883 <dbl>,  
## # 43884 <dbl>, 43885 <dbl>, 43886 <dbl>, 43887 <dbl>, 43888 <dbl>,  
## # 43889 <dbl>, 43890 <dbl>, 43891 <dbl>, 43892 <dbl>, 43893 <dbl>,  
## # 43894 <dbl>, 43895 <dbl>, 43896 <dbl>, 43897 <dbl>, 43898 <dbl>,  
## # 43899 <dbl>, 43900 <dbl>, 43901 <dbl>, 43902 <dbl>, 43903 <dbl>,  
## # 43904 <dbl>, 43905 <dbl>, 43906 <dbl>, 43907 <dbl>, 43908 <dbl>,  
## # 43909 <dbl>, 43910 <dbl>, 43911 <dbl>, 43912 <dbl>, 43913 <dbl>,  
## # 43914 <dbl>, 43915 <dbl>, 43916 <dbl>, 43917 <dbl>, 43918 <dbl>,  
## # 43919 <dbl>, 43920 <dbl>, 43921 <dbl>, 43922 <dbl>, 43923 <dbl>,  
## # 43924 <dbl>, 43925 <dbl>, 43926 <dbl>, 43927 <dbl>, 43928 <dbl>,  
## # 43929 <dbl>, 43930 <dbl>, 43931 <dbl>, 43932 <dbl>, 43933 <dbl>,  
## # 43934 <dbl>, 43935 <dbl>, …

So are the data “tidy”?

The data from the spreadsheet are not “tidy”.

The columns “43831, 43832, 43833…” represent different dates. Therefore, this does meet the second argument of “tidy data” - “Each observation must have its own row”.

But we can fix this using functions from the packages included in the tidyverse

Remember, first we must install the packages from the tidyverse

install.packages("tidyverse")

## The tidyverse: Tidying data

Now that the tidyverse has been installed, we can use the functions from the packages to “tidy” the data.

One package which is very helpful for this is called “tidyr”

Instead of loading individual packages, we can load the core tidyverse packages with one command

library(tidyverse)

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──

## ✓ ggplot2 3.3.4 ✓ purrr 0.3.4  
## ✓ tibble 3.1.2 ✓ dplyr 1.0.7  
## ✓ tidyr 1.1.3 ✓ stringr 1.4.0  
## ✓ readr 1.4.0 ✓ forcats 0.5.1

## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

The core packages contain powerful functions we can use to process, analyse and visualise data.

Remember to look at the documentation for a package type “?[name of package]”

Example -

?tidyr

To look at the functions within a package, type [name of package]::

Example

tidyr::

To reformat the data to a tidy format, we need to transform the data from wide to long.

The Epidemiologist R handbook has an excellent section describing how to do this

12 - [Pivoting data](https://epirhandbook.com/pivoting-data.html?q=pivot_long#pivoting-data)

## The tidyverse: Wide to long

## [From the Epidemiologist R handbook](https://epirhandbook.com/pivoting-data.html?q=pivot_long#wide-to-long)

africa\_covid\_cases\_long <- africa\_covid\_cases %>%   
 pivot\_longer(cols=4:492, names\_to="excel\_date", values\_to="cases")

Transforming data from wide to long usually requires a few attempts to ensure you have the correct outcome!

head(africa\_covid\_cases\_long, n=3)

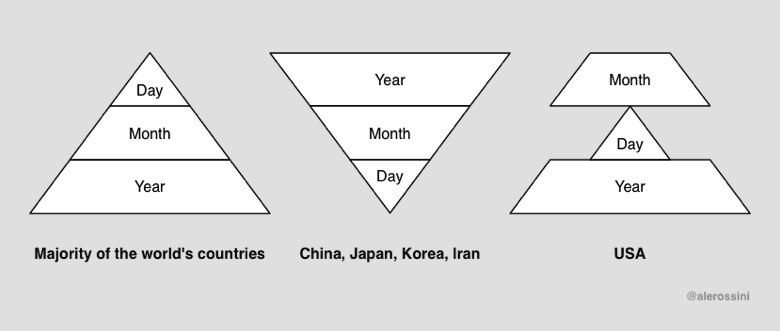
## # A tibble: 3 x 5  
## ISO COUNTRY\_NAME AFRICAN\_REGION excel\_date cases  
## <chr> <chr> <chr> <chr> <dbl>  
## 1 DZA Algeria Northern Africa 43831 0  
## 2 DZA Algeria Northern Africa 43832 0  
## 3 DZA Algeria Northern Africa 43833 0

This looks correct!

You can add comments to code to show other people (and remind yourself!) why you wrote the code in a particular way

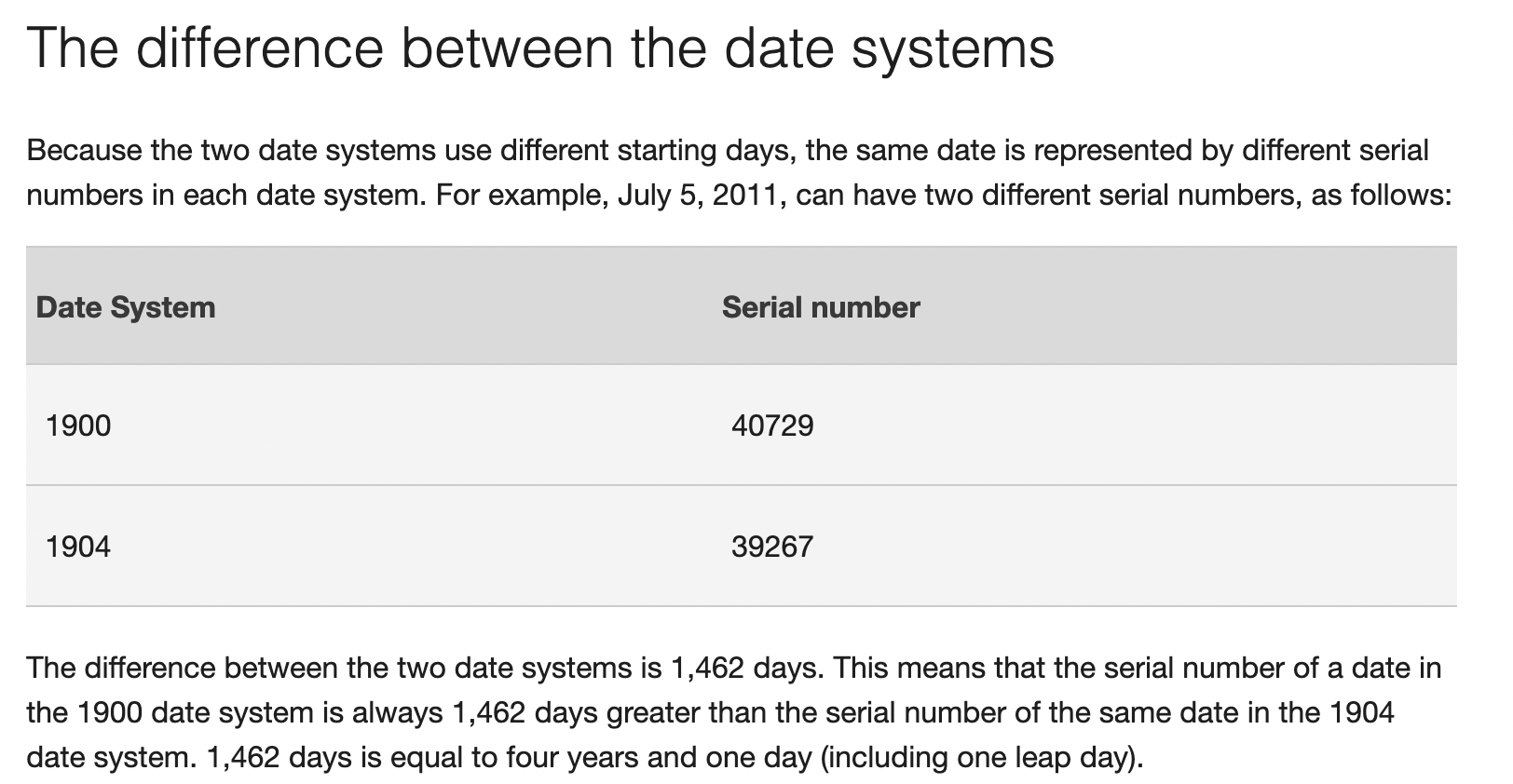
africa\_covid\_cases\_long <-  
 africa\_covid\_cases %>% #tell R to use this dataset  
 pivot\_longer(cols = 4:492,#select the columns you want  
 names\_to = "excel\_date", #name the new date column  
 values\_to = "cases") #name the new cases column

## Working with dates

[](https://iso.mit.edu/americanisms/date-format-in-the-united-states/)

To add to the confusion, Excel has 2 additional date systems:

1. 1900 date system
2. 1904 date system

* 
* <https://support.microsoft.com/en-us/office/date-systems-in-excel-e7fe7167-48a9-4b96-bb53-5612a800b487>

In the data set we are using, the dates are in this format:

head(africa\_covid\_cases\_long$excel\_date)

## [1] "43831" "43832" "43833" "43834" "43835" "43836"

We can use a function from another package to convert this to a standard date format.

install.packages("janitor")

library(janitor)

##   
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':  
##   
## chisq.test, fisher.test

The package [janitor](https://cran.r-project.org/web/packages/janitor/vignettes/janitor.html) has many helpful functions for cleaning data

africa\_covid\_cases\_long <- africa\_covid\_cases\_long %>%   
 mutate(date\_format=excel\_numeric\_to\_date(as.numeric(excel\_date)))  
  
head(africa\_covid\_cases\_long$date\_format)

## [1] "2020-01-01" "2020-01-02" "2020-01-03" "2020-01-04" "2020-01-05"  
## [6] "2020-01-06"

The new variable created “date\_format” is in the format YEAR-MONTH-DATE.

We can also check if the values in the new variable look correct

min(africa\_covid\_cases\_long$date\_format) #minimum date

## [1] "2020-01-01"

max(africa\_covid\_cases\_long$date\_format) #maximum date

## [1] "2021-05-03"

We know this is a data set of COVID cases so the date range (from the start of 2020 through to May of 2021) looks to be correct.

## Your analysis dataset

We have created 2 data objects - *africa\_covid\_cases and africa\_covid\_cases\_long*

Before analysing the data, it is a good idea to generate a new dataset which only contains the variables you need to analyse.

So what variables do we have in *africa\_covid\_cases\_long*

names(africa\_covid\_cases\_long)

## [1] "ISO" "COUNTRY\_NAME" "AFRICAN\_REGION" "excel\_date"   
## [5] "cases" "date\_format"

We can select the variables we want to keep using the select function from the dplyr package

dplyr is a core part of the tidyverse so it is loaded when you write library(tidyverse)

analysis\_dataset <- africa\_covid\_cases\_long %>%   
 select(date\_format,AFRICAN\_REGION, COUNTRY\_NAME, cases)

We can look at the first few rows of the dataset we have created to check we have selected the desired variables.

head(analysis\_dataset)

## # A tibble: 6 x 4  
## date\_format AFRICAN\_REGION COUNTRY\_NAME cases  
## <date> <chr> <chr> <dbl>  
## 1 2020-01-01 Northern Africa Algeria 0  
## 2 2020-01-02 Northern Africa Algeria 0  
## 3 2020-01-03 Northern Africa Algeria 0  
## 4 2020-01-04 Northern Africa Algeria 0  
## 5 2020-01-05 Northern Africa Algeria 0  
## 6 2020-01-06 Northern Africa Algeria 0

The select function from the dplyr package is very useful.

It can also be used to rename selected variables

analysis\_dataset <- africa\_covid\_cases\_long %>%   
 select(date=date\_format,region=AFRICAN\_REGION, country=COUNTRY\_NAME, cases)

We have renamed AFRICAN\_REGION and COUNTRY\_NAME as region and country

head(analysis\_dataset)

## # A tibble: 6 x 4  
## date region country cases  
## <date> <chr> <chr> <dbl>  
## 1 2020-01-01 Northern Africa Algeria 0  
## 2 2020-01-02 Northern Africa Algeria 0  
## 3 2020-01-03 Northern Africa Algeria 0  
## 4 2020-01-04 Northern Africa Algeria 0  
## 5 2020-01-05 Northern Africa Algeria 0  
## 6 2020-01-06 Northern Africa Algeria 0

## Answering questions with data 1

So far we have:

* Imported the data from an Excel worksheet
* Reshaped the data into a “tidy” format
* Changed the format of a variable to a date
* Selected only the variables we want to use for the analysis

Now we can start to use the dataset to answer questions

*How many confirmed cases of COVID-19 have been recorded in Africa?*

analysis\_dataset %>%   
 summarise(total\_covid\_cases=sum(cases))

## # A tibble: 1 x 1  
## total\_covid\_cases  
## <dbl>  
## 1 NA

The answer is “NA”, which stands for “Not Available”

This is a good example of how R deals with missing data

* There may be dates in our dataset where there were no confirmed cases of COVID-19 recorded
* When data are missing, R will display “NA” for the variable
* If you try to run a calculation on data where there is one or more “NA” values, the results will be “NA”

There are several options for dealing with missing values in R

1. Complete case analysis
   * Remove rows with any missing data

full\_dataset <- na.omit(analysis\_dataset)

1. Exclude “NA” values from calculations
   * Add an additional argument to the function to remove “NA”

analysis\_dataset %>%   
 summarise(total\_covid\_cases=sum(cases, na.rm=TRUE))

## # A tibble: 1 x 1  
## total\_covid\_cases  
## <dbl>  
## 1 4561465

This command has now excluded NA values and has provided us with an answer for the number of confirmed COVID-19 cases in Africa - 4,561,465

*How many confirmed cases of COVID-19 have been recorded in Africa, by region?*

analysis\_dataset %>%   
 group\_by(region) %>%   
 summarise(total\_covid\_cases=sum(cases, na.rm=TRUE))

## # A tibble: 5 x 2  
## region total\_covid\_cases  
## <chr> <dbl>  
## 1 Central Africa 161353  
## 2 Eastern Africa 622537  
## 3 Northern Africa 1371469  
## 4 Southern Africa 1970137  
## 5 Western Africa 435969

group\_by is a very powerful function for summarising data

analysis\_dataset %>%   
 group\_by(region) %>%   
 summarise(total\_covid\_cases=sum(cases, na.rm=TRUE)) %>%   
 arrange(-total\_covid\_cases)

## # A tibble: 5 x 2  
## region total\_covid\_cases  
## <chr> <dbl>  
## 1 Southern Africa 1970137  
## 2 Northern Africa 1371469  
## 3 Eastern Africa 622537  
## 4 Western Africa 435969  
## 5 Central Africa 161353

The arrange function can be used to organise the results. In this case we have instructed R to sort the results by the total\_covid\_cases variable, from highest to lowest value.

We can add multiple variables to group\_by

If we add region and country to the group\_by command, sort from highest to lowest, we can see which countries reported the most confirmed COVID-19 cases

analysis\_dataset %>%   
 group\_by(region, country) %>%   
 summarise(total\_covid\_cases=sum(cases, na.rm=TRUE)) %>%   
 arrange(-total\_covid\_cases)

## `summarise()` has grouped output by 'region'. You can override using the `.groups` argument.

## # A tibble: 53 x 3  
## # Groups: region [5]  
## region country total\_covid\_cases  
## <chr> <chr> <dbl>  
## 1 Southern Africa South Africa 1584064  
## 2 Northern Africa Morocco 511856  
## 3 Northern Africa Tunisia 311743  
## 4 Eastern Africa Ethiopia 258353  
## 5 Northern Africa Egypt 228584  
## 6 Northern Africa Libya 178335  
## 7 Western Africa Nigeria 165199  
## 8 Eastern Africa Kenya 160559  
## 9 Northern Africa Algeria 122522  
## 10 Western Africa Ghana 92683  
## # … with 43 more rows

Another useful function is filter which can be used to apply filters to calculations

We can repeat the previous calculation, but then add a filter to only include results from countries in Northern Africa

analysis\_dataset %>%   
 group\_by(region, country) %>%   
 summarise(total\_covid\_cases=sum(cases, na.rm=TRUE)) %>%   
 arrange(-total\_covid\_cases) %>%   
 filter(region=="Northern Africa")

## `summarise()` has grouped output by 'region'. You can override using the `.groups` argument.

## # A tibble: 6 x 3  
## # Groups: region [1]  
## region country total\_covid\_cases  
## <chr> <chr> <dbl>  
## 1 Northern Africa Morocco 511856  
## 2 Northern Africa Tunisia 311743  
## 3 Northern Africa Egypt 228584  
## 4 Northern Africa Libya 178335  
## 5 Northern Africa Algeria 122522  
## 6 Northern Africa Mauritania 18429

The filter can be applied at any point within the calculation. For very complex calculations, it is helpful to apply the filter as early as possible. This reduces the number of records before the complex portion of the calculation occurs.

filter can also be used to make data frames

northern\_africa <- analysis\_dataset %>%   
 filter(region=="Northern Africa")

Using filters, we can answer additional questions.

*When was the first confirmed case of COVID-19 in Northern Africa?*

northern\_africa %>%   
 filter(cases>0) %>%   
 filter(date == min(date, na.rm=TRUE))

## # A tibble: 1 x 4  
## date region country cases  
## <date> <chr> <chr> <dbl>  
## 1 2020-02-14 Northern Africa Egypt 1

Here we have added 2 filters:

1. Only keep records where the value for cases is higher than 0
2. Only keep records where the value for date is equal to the minimum value for date. We have also added the na.rm=TRUE command from a previous step. If you don’t know the data very well, it is good practice to add this command.

When was the first confirmed case of COVID-19 in Northern Africa, by country?

northern\_africa %>%   
 group\_by(country) %>%   
 filter(cases>0) %>%   
 filter(date == min(date, na.rm=TRUE))

## # A tibble: 6 x 4  
## # Groups: country [6]  
## date region country cases  
## <date> <chr> <chr> <dbl>  
## 1 2020-02-25 Northern Africa Algeria 1  
## 2 2020-02-14 Northern Africa Egypt 1  
## 3 2020-03-24 Northern Africa Libya 1  
## 4 2020-03-13 Northern Africa Mauritania 1  
## 5 2020-03-02 Northern Africa Morocco 1  
## 6 2020-03-02 Northern Africa Tunisia 1

These results can be stored in an object for future use

first\_cases\_northern\_africa <- northern\_africa %>%   
 group\_by(country) %>%   
 filter(cases>0) %>%   
 filter(date == min(date, na.rm=TRUE))

On what date, was the 100th case of COVID-19 reported from each country in Northern Africa?

northern\_africa %>%   
 group\_by(country) %>%   
 mutate(cumulative\_cases=cumsum(cases)) %>%   
 filter(cumulative\_cases>=100) %>%   
 slice(1) %>%   
 pull(date, country)

## Algeria Egypt Libya Mauritania Morocco Tunisia   
## "2020-03-20" "2020-03-15" "2020-05-28" "2020-05-19" "2020-03-22" "2020-03-24"

Here we have introduced two new functions slice and pull

slice can be used to select certain rows from a dataset. In this case, we have added a column which is the cumulative number of cases, selected the first row after filtering the dataset to only include results where the value is greater than or equal to 100, and then selected the first row using the slice command.

An additional step is the pull command. This is useful when you want to extract specific values from the result.

first\_100cases <- northern\_africa %>%   
 group\_by(country) %>%   
 mutate(cumulative\_cases=cumsum(cases)) %>%   
 filter(cumulative\_cases>=100) %>%   
 slice(1) %>%   
 pull(date, country)

## Visualising data

One of the key strengths of R is visualising data. There are many packages which have functions you can use to make graphs, tables, maps…the list is endless!

The first package of functions we will use for visualising data is another core tidyverse package called ggplot2. This is commonly referred to as ggplot

We have already loaded the package when we ran library(tidyverse)

You can also choose to only load the ggplot2 package by typing library(ggplot2)

library(ggplot2)

The Epidemiologist R handbook has 2 sections focused on ggplot

1. [ggplot basics](https://epirhandbook.com/ggplot-basics.html)
2. [ggplot tips](https://epirhandbook.com/ggplot-tips.html)

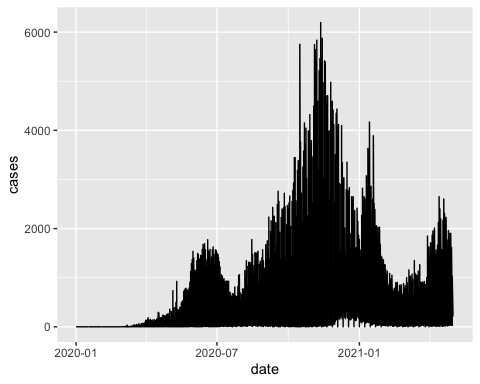
These sections contain very helpful explanations of many of the functions available with ggplot. There are also a number of excellent references for every type of graph you want to make.

We will walk through some common examples to teach some of the most common approaches

Make a graph of confirmed COVID-19 cases in Northern Africa

ggplot(northern\_africa, aes(x=date,y=cases)) +  
 geom\_line()

## Warning: Removed 7 row(s) containing missing values (geom\_path).



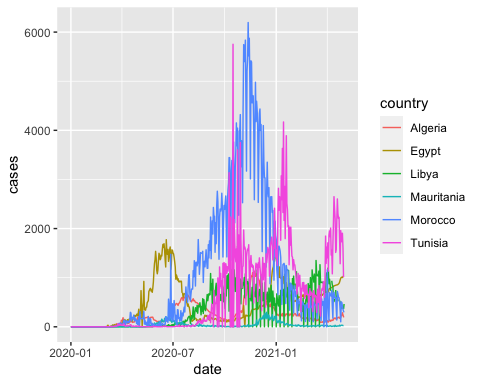
This command has generated a line graph of confirmed COVID-19 cases for countries in Northern Africa.

From earlier steps, we know that the dataset northern\_africa contains data from multiple countries: `r unique(northern\_africa$country’

We can add more information to the ggplot command to draw separate lines for each country

ggplot(northern\_africa, aes(x=date,y=cases, color=country)) +  
 geom\_line()

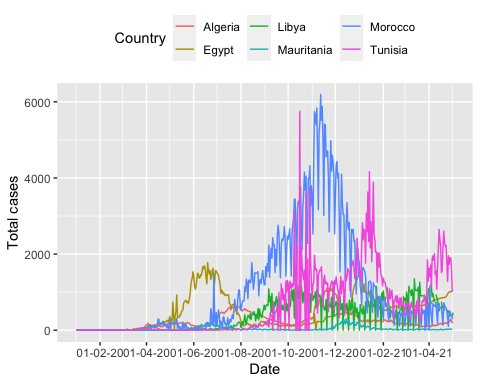
## Warning: Removed 9 row(s) containing missing values (geom\_path).



To make the graph more presentable, we can add more options to the ggplot command

ggplot(northern\_africa, aes(x=date,y=cases, color=country)) +  
 geom\_line() +  
 labs(x='Date', y='Total cases', color='Country') + #label axes  
 theme(legend.position='top') + #place legend at top of graph  
 scale\_x\_date(date\_breaks = '2 months', #set x axis to have 2 month breaks  
 date\_minor\_breaks = '1 month', #set x axis to have 1 month breaks  
 date\_labels = '%d-%m-%y') #change label for x axis

## Warning: Removed 9 row(s) containing missing values (geom\_path).



It is still difficult to see the data for each country. There is a helpful command to fix this

ggplot(northern\_africa, aes(x=date,y=cases, color=country)) +  
 geom\_line() +  
 labs(x='Date', y='Total cases', color='Country') + #label axes  
 theme(legend.position='none') + #remove legend by setting position to 'none'  
 scale\_x\_date(date\_breaks = '2 months', #set x axis to have 2 month breaks  
 date\_minor\_breaks = '1 month', #set x axis to have 1 month breaks  
 date\_labels = '%d-%m-%y') + #change label for x axis  
 facet\_wrap(~country) # this will create a separate graph for each country

## Warning: Removed 9 row(s) containing missing values (geom\_path).

