Data Management Using R

Myo Minn Oo, MD, PhD

Last updated on 2021-12-12

Contents

Welcome to `dmur`!

This is a book for anyone who are interested in manipulating and processing medical data. This book introduces different aspects of data management and how to implement these in R using RStudio. While there are a plethora of great R books covering a variety of data management topics, I hope this book would serve as a self-learning guide to avoid roadblocks and frustrations before becoming fully comfortable with using R. Many beginners find themselves wanting to develop data management skills in R, but lose their patience after they encounter a steep learning curve of R and several months of frustration. If you feel nostalgic about this, this book is for you.

It is intended for non-technical audience and Stata users to:

- Serve as a guidebook to R code for data management
- Serve as a R code reference manual for `mStats` package
- Provide task-centered examples addressing common data management problems
- Assist people in transitioning to R

Under construction

This book is still **UNDER CONSTRUCTION**. If you have any comments or suggestions, feel free to contact me at dr.myominnoo@gmail.com. Thank you!

if you have a dataset that you think would be suitable for inclusion in this text (as an example or for an exercise), I would love to hear about it.

Inspirations

Tutorials and books that provided ideas and knowledge of this book are credited within their respective pages. More generally, the following sources provided inspiration for this book:

- Data Management Using Stata: A Practical Handbook
- UCLA's Stata tutorials
- R for applied epidemiology and public health

- R for Data Science book (R4DS)

- bookdown: Authoring Books and Technical Documents with R Markdown
- Netlify hosts this website

The design of this book is based on the codes obtained from Peter Higgins. Kudos to Peter!

How to use this handbook

- Browse the pages in the Table of Contents, or use the search box
- Click the "copy" icons to copy code

- You can follow along with the example datasets we will download in the next chapter 1.8.

Software versions

The knitr package (Xie 2015) and the bookdown package (Xie 2021) were used to compile this text. The R session information is shown below:

```
xfun::session_info()
```

```
## R version 4.1.0 (2021-05-18)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Locale: en_US.UTF-8 / en_US.UTF-8 / en_US.UTF-8 / C / en_US.UTF-8 / en_US.UTF-8
##
## Package version:
##   base64enc_0.1.3 bookdown_0.24.4 compiler_4.1.0  digest_0.6.29
##   evaluate_0.14   fastmap_1.1.0   glue_1.5.1      graphics_4.1.0
##   grDevices_4.1.0 highr_0.9       htmltools_0.5.2 jquerylib_0.1.4
##   jsonlite_1.7.2  knitr_1.36      magrittr_2.0.1  methods_4.1.0
##   rlang_0.4.12    rmarkdown_2.11  stats_4.1.0     stringi_1.7.6
##   stringr_1.4.0   tinytex_0.35    tools_4.1.0     utils_4.1.0
##   xfun_0.28       yaml_2.2.1
```

```
packageVersion("tidyverse")
```

```
## [1] '1.3.1'
```

```
packageVersion("dplyr")
```

```
## [1] '1.0.7'
```

```
packageVersion("magrittr")
```

```
## [1] '2.0.1'
```

```
packageVersion("mStats")
```

```
## [1] '4.0.0'
```

Terms of Use and Contribution

License

Academic courses and training programs are welcome to use this handbook with their students, but please send us an email to let me know. If you have questions about your intended use, email **dr.myominnoo@gmail.com**.

Citation

Oo, Myo Minn. Data Management Using R. 2021.

Contribution

If you would like to make a content contribution, please contact with us first via Github issues or by email. We are implementing a schedule for updates and are creating a contributor guide.

Please note that the epiRhandbook project is released with a Contributor Code of Conduct. By contributing to this project, you agree to abide by its terms.

Chapter 1

Getting Started

R is an **free** software for statistics and graphics. It is widely used among statisticians and data scientists for developing statistical software and data analysis.

R is primarily developed in three programming languages: C, Fortan and R itself. Although It itself can be used with command line interface, there are several third-party integrated development environment (IDE) with nice graphical user interface, including RStudio and Jupyter Notebook.

R comes from `S` programming language. `S` was created by John Chambers in 1976 at Bell Labs. Later, two statisticians, Ross Ihaka and Robert Gentleman, developed R that is currently maintained by the R Development Core Team. R is named partly after the first names of the first two R authors and partly as a play on the name of S.

R can be downloaded from http://cran.r-project.org/.

1.1 RStudio

`Rstudio` is an integraded environment for R. It comes in two versions: Desktop version is a desktop application and server version runs on a web browser. Regular `RStudio` for personal usage is free for both desktop and server version.

> If R is a car engine, then RStudio is the structure of that care, body frame, interior design and every other thing that help us operate it safely, efficiently and comfotably.

RStudio can be downloaded from their official website.

1.2 Start screen

When you open RStudio, you will see a screen that looks like figure 1.1.

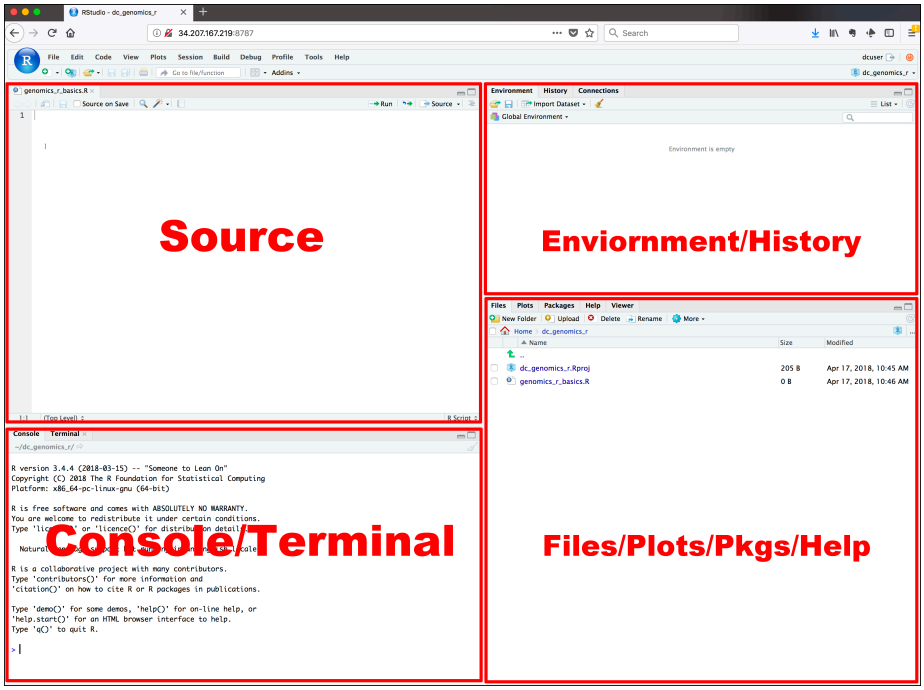

Figure 1.1: RStudio's start screen

The interface has four window panes. Each window pane may have several tabs or sub-windows. By default, **Source Pane** is on the top-left corner, **Console Pane** on bottom-left, **Environment Pane** on top-right, and **Files Pane** on bottom-right.

1. **Source Pane** is a text editor that will be referred to as *R Script* later on. You can write commands and save them, which is the main point of reproducibility. Anyone who has this R Script can review and edit it in the future.

2. **Console Pane** is the place you write your line-by-line command. It means you can only write a single command or a long paragraph of commands. After you close the RStudio, the commands will not be saved unless you specified to do so. However, it is the best way to saving **R Script** in **Source Pane**to store the commands you desire.

    - Symbol `>` called prompt
    - Type 3 + 4, and press Enter.

3. **Environment Pane** has several sub-windows. For data management and beginner, you only have to know Environment and History.

    - `Environment` is where R works.
    - `Global Environment` is the place where your data will be after importing data.
    - `History` saves the commands you run in R console.
    - `Connections` is where you connect to external databases.

4. **Files Pane** also has several sub-windows.

    - `Files` is like a folder manager on your phone. You can manage files and folders as well as set the working directory.
    - `Plots` is where your plots will appear.
    - `Packages` is where you manage your R packages. You can install it from CRAN or other repositories. You can also install locally stored R packages.
    - `Help` is where R stores documentations. You can open the help or introduction page of the respective packages as well as individual functions.

You can arrange these windows as you like. I think putting console on the right upper window will give you a lot of freedom as you will mostly be using **Source Pane** and **Console Pane** simultaneously. You can change this by pressing the four window icon below the menu bar and choosing the option **Console on the Right** as shown in figure 1.2.

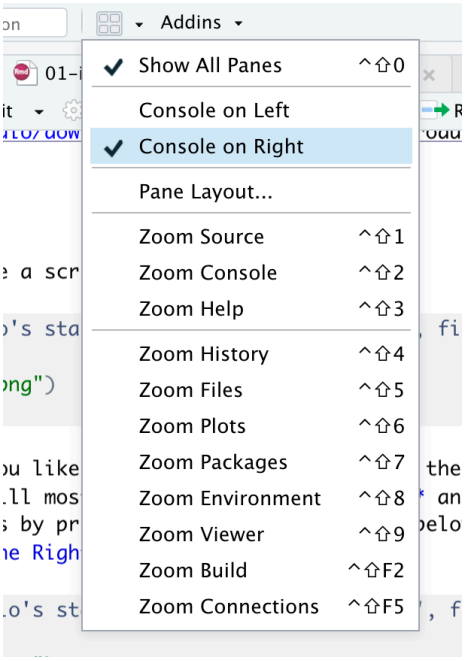

Figure 1.2: Changing console to the right upper window

1.3 R Packages

`Base R` is what you get after installing R. R is powerful because of its tens of thousands of packages. The number is still growing. This also creates the problem of confusion. Even no single user can check or use all packages R can offer. A solution is to stick to certain packages that work well for you. In this book, I will show you a few packages that works for me.

1.3.1 Installing a package from CRAN

To install a package, you can use the function `install.packages("package_name")` if it is published on `CRAN`. `CRAN` is the comprehensive R Archive Network, web servers that store R packages.

Packages installed directly from Github are usually under development, but up-to-date and often free from previous issues and bugs.

Let's install the packages used in this book.

```
package_name <- c("tidyverse", "mStats", "flextable")
install.packages(package_name)
```

1.3.2 Installing a package from GitHub

For packages published on GitHub, use either `devtools::install_github("repo")` or `remotes::install_github("repo")`.

Let's illustrate this by installing `remotes` and then `mStats` from GitHub.

```
install.packages("remotes")
remotes::install_github("myominnoo/mStats")
```

1.3.3 Essential packages for this book

- dplyr
- mStats
- magrittr

1.4 mStats

R has a steep learning curve. For physicians and health professionals, they just want to get started with data management and analysis, rather than spending a lot of time to learn and master R. There might be an advantage to mastering R; however unlike statisticians and programmers, it seems a very daunting process for them to become comfortable with coding in R.

The `mStats` package was developed to facilitate the data management and analysis in R while requiring only a few basic of R. It takes some principles from one of the most popular statistical software in public health settings, `STATA`. `STATA` has been in existence for a long time now that most of its commands are stable, powerful, and intuitive. These features attract power-users and help them accomplish their data tasks.

In a nutshell, `mStats` is a R package that facilitates data management and analysis of health research in R environment. It includes three core sets of functions that are designed to help data management, data analysis, and reporting streamline for health research projects.

For epidemiology and public health, there are already a number of good packages available for `R`, including but not limited to `epiCalc`, `Epi`, `epibasix`, `epiDisplay`, `epiR` and `epitools`. `mStats` does not intend to replace any of them, but to only contribute as much as I can to data management and analysis of health research projects.

1.4.1 How it started

The `mStats` package came into life when I started my doctoral life in Thailand in 2018. The first version of `mStats` was released in late 2018. Initially it was intended to prove a concept of what I have learnt and how much I could make

use of what I have learnt so far. In addition, I would like something open source unlike `STATA` and something different from base R.

There are two packages that deeply inspired me to do this: `dplyr` under the universe of `tidyverse` by **Hadley Wickham** and `epiCalc` by my advisor **Virasakdi Chongsuvivatwong**. The first version of `mStats` made use of `ggplot2` and included functions for generating ready-made plots. I released several small patch versions with addition of a few functions that were interesting to me at that time.

In mid 2019, I started another package called `stats2`. In this package, I tried to simplify the concept of one function doing one task with a few or no options to tweak the task. Although this might be in contrast with the generous flexibility of `R`, I hardly doubt beginners would mind this limitation. In fact, such simplicity might make users' life easier to get their jobs in R done. The package was on CRAN and released on March 31, 2020.

As times progress, I hope this package will find its way to some people in their works.

1.4.2 Guiding principles

Most of the guiding principles behind `dplyr`, `tidyverse`, and `STATA` inspire the creation and development of the `mStats` package:

- **Compose simple functions with the pipe**:
  - `data masking` for tidy evaluation: it is very common to use datasets as in the form of `data frame` in as R analysis project. `R` functions are mostly very generic and created for basic data structures of R. The [tidyverse] uses tidy evaluation and implements data masking in its `dplyr` package. This allows manipulation of **data-variables** as if they were "programming" variables that live in an environment. Basically, it allows you to type, as an example, `generate(airquality, lnOzone = log(Ozone))` instead of `airquality$lnOzone <- log(airquality$Ozone)`.
  - The principle of pipes changes the paradigm of coding in `R`. The use of the pipe function `%>%` from the `magrittr` package allows for cleaner code and enhances readability of the codes. Most functions in `mStats` is compatible with the usage of the pipe function.
- **Designed for humans**:
  - `KISS`: Keep `functions` short and simple - one function does only one task with few options to change the nature of the task. `mStats` intentionally limits options of arguments in functions and attempts to balance between having too many argument options and having no options at all.

  - `Labels`: `mStats` consistently uses the concept of **labelling variables and datasets**. It also streamlines `labels` for the purpose of report generation. Though it is not necessary to label values, this can be done using certain packages like `labelled` and `sjlabelled`.
  - Outputs: outputs in the console are well-formatted for the ease of interpretation.

- **Reproducible research**
  - Making tables is One of the most common tasks in health research analysis where most researchers waste much time. Reproducibility in health research is becoming crucially important and depends on streamlining this process. `mStats` provides one of many workflows used for tables generation and link to `R` codes for getting the job done.

1.5 Package masking

Packages including `mStats` might contain functions that are in conflicts with other packages. I suggest to explicitly use the syntax to call the desired package, `package::function`.

For example, the `cut` function from `mStats` will be masked from the `cut` function from `base`. When you want to use the function from base, you need to write this: `base::cut()`.

1.6 Piping your workflow

The pipe function `%>%` from the `magrittr` package can connect several lines of codes in a single workflow. It allows cleaner codes and enhances human readability. As an example, if you have three generic functions, you can write codes like this: `function3(function32(function31(..., data), ...), ...)`. But it is confusing. By using `%>%`, we can arrange codes from top to bottom and left to right as follow:

```
data %>%
  function1(...) %>%
  function2(...) %>%
  function3(...)
```

1.7 Searching Help

1.7.1 General

If you get stuck, try to google it or search on virtual forums like `Stack Overflow`. Most of what you want to know are already answered or solved in

one of those forums.

1.7.2 Getting help

If you encounter a clear bug, please file an issue with a minimal reproducible example on GitHub. For questions and other discussion, please directly email me dr.myominnoo@gmail.com or use the mStats mailing list.

1.8 Example datasets

In this book, we will use datasets from several sources. The code chunks below download these datasets into a `data` folder under your current working directory.

Let's create a function that takes a URL link for zipped files, download, and unzip the files into specified directory.

```
download_zipfile <- function(url, exdir = "data/", file = tempfile(), junkpaths = TRUE)
    for (link in url) {
        ## download the dmus2 zipped file into temporary file
        download.file(link, file)

        ## extract the dmus2 datasets into the data folder
        unzip(file, exdir = exdir, junkpaths = TRUE)
    }
}
```

1.8.1 dmus2

The following code chunk downloads datasets from one of the most popular data management books for Stata, Data Management Using Stata: A Practical Handbook, Second Edition

```
## Download dmus2 zipped file
download_zipfile("https://www.stata-press.com/data/dmus2/dmus2.zip",
                 exdir = "data/dmus2")
```

1.8.1.1 WHO Mortality Data

THe following code chunks download datasets from WHO webpage.

```
## 1. A list of countries-years available for the mortality and population data.
## 2. Reference populations and live births.
## 3. Country codes
file <- c("https://cdn.who.int/media/docs/default-source/world-health-data-platform/mor
          "https://cdn.who.int/media/docs/default-source/world-health-data-platform/mor
          "https://cdn.who.int/media/docs/default-source/world-health-data-platform/mor
download_zipfile(file, exdir = "data/who")
```

1.8.2 UCLA Stata Tutorial

The following code chunks download datasets from UCLA's Stata tutorials.

```
if (!dir.exists("data/ucla"))
    dir.create("data/ucla") ## this will create a directory if it's not there

download.file("http://stats.idre.ucla.edu/stat/data/patient_pt1_stata_dm.dta",
              destfile = "data/ucla/patient_pt1_stata_dm.dta")
download.file("http://stats.idre.ucla.edu/stat/data/patient_pt2_stata_dm.dta",
              destfile = "data/ucla/patient_pt2_stata_dm.dta")
download.file("http://stats.idre.ucla.edu/stat/data/doctor_stata_dm.dta",
              destfile = "data/ucla/doctor_stata_dm.dta")

download.file("http://stats.idre.ucla.edu/stat/data/hsb2.csv",
              destfile = "data/ucla/hsb2.csv")
download.file("http://stats.idre.ucla.edu/stat/data/hsb2.xls",
              destfile = "data/ucla/hsb2.xls")
```

1.8.3 Miliary Tuberculosis

The following dataset was used to evaluated a scoring system for central nervous tuberculosis among patients with miliary tuberculosis in a Chinese hospital. Read their study here.

We put this dataset into a separate path `data/others/`.

```
if (!dir.exists("data/others"))
    dir.create("data/others") ## this will create a directory if it's not there

download.file("https://journals.plos.org/plosone/article/file?type=supplementary&id=info:doi/10.1
              destfile = "data/others/miliary_tb.xlsx")
```

Chapter 2

Using existing data files

You have some data that you are eager to analyze using R. Before you can analyze the data, you must first read the data into R. This chapter describes how you can read common types of data files into R.

2.1 Working directories

I would recommend against using absolute pathway with the `setwd()` function for your works. One main reason is that other people don't have the same pathways as yours, and it makes replicating codes in other people's devices extremely cumbersome. There are two ways I find more efficient and productive than just setting up your directory with `setwd()`.

1. Open RStudio by doublie-clicking the `.R` file.
2. Create an R project and open RStudio by double-clicking the `.Rproj` file.

Both methods will automatically set up your working directory to the folder where your `.R` or `.Rproj` file exists. Then you use relative pathway to navigate to other folders and files.

If you have followed the section on downloading `dmus2` datasets, you might already have a `data` folder. You can put this data folder under your project folder, whatever you may call it. For the sake of demonstration, let's create a folder named `dmur` and put this `data` folder under `dmur`. You can start create necessary `.R` and `.Rproj` files as follows. Essentially, you will have the following structures under the`dmur` folder.

```
- dmur              ## this is your project or root folder
    |
    - data          ## this is data folder that contains `dmus2` datasets.
        |
        - many datasets ...     ## these are `dmus2` datasets
```

```
    |
    - dmur.R  ## You can double-click this to open RStudio
    |
    - dmur.Rproj  ## You can also double-click this to open RStudio
```

[To add how to create .R or .Rproj files]

2.2 Reading data files into R

There are several data files that you can read and import into R. This chapter begins by illustrating how you can read several common types of datasets into R. As you would expect, you can certainly do this in multiple ways; however I will show the ways I think simplest and most efficient.

In previous section 1.8, we have downloaded several datasets into respective folders under a `data` folder of your current working directory. It should look like the following structure.

```
- Your current working directory
    |
    _ data
        |
        _ dmus2     ## <<== This is where dmus2 datasets were downloaded ==>>
        |
        _ who       ## <<== This is where who datasets were downloaded ==>>
```

When we specify a path later, we will use this folder structure. If we want to import a dataset from `dmus2` folder, we will specify the file in this format `./data/dmus2/filename`.

2.2.1 CSV and text data files

Comma-separated values files, also known as `CSV` files, are a common format for storing raw data. They have a filename extension of `.csv`. As the name suggests, `CSV` files use `commas` to separate the variables (columns) of the data. Data files in `CSV` format often include the names of the variables in the first row, also separated by commas.

We illustrate this below by using a `momkid1.csv` data file, which includes identification number of mom, month, day and year of mom's birthday, and kid's birthday.

While there are several ways to import a raw data file, we use `read_csv()` function from the `readr` package. As we use this function probably once at this time when we import datasets, we write `readr::read_csv()` to indicate the function from `readr` package without loading the whole package. We will use

this pattern in the whole book whenever we need one particular function from a package at one time.

```
momkid1 <- readr::read_csv("data/dmus2/momkid1.csv")
```

```
## Rows: 4 Columns: 5
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): kidbday
## dbl (4): momid, momm, momd, momy
```

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

```
momkid1
```

```
## # A tibble: 4 x 5
##   momid  momm  momd  momy kidbday
##   <dbl> <dbl> <dbl> <dbl> <chr>
## 1     1    11    28  1972 1/5/1998
## 2     2     4     3  1973 4/11/2002
## 3     3     6    13  1968 5/15/1996
## 4     4     1     5  1960 1/4/2004
```

The imported dataset can be displayed in the console just by typing in the assigned object name `momkid1`.

Text files are also quite popular in storing simple raw data. In the example below, the `dentists1.txt` data file contains five rows of data regarding five dentists. The four variables reflect the name of the dentist, the years she or he has been practicing, whether she or he works full time, and whether she or he recommends Quaddent gum.

```
dentists1 <- readr::read_csv("data/dmus2/dentists1.txt")
```

```
## Rows: 5 Columns: 4
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): name
## dbl (3): years, fulltime, recom
```

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

```
dentists1
```

```
## # A tibble: 5 x 4
##   name              years fulltime recom
```

```
##   <chr>             <dbl>    <dbl> <dbl>
## 1 Y. Don Uflossmore  7.25        0     1
## 2 Olive Tu'Drill    10.2         1     1
## 3 Isaac O'Yerbreath 32.8         1     1
## 4 Ruth Canaale      22           1     1
## 5 Mike Avity         8.5         0     0
```

See `?read_csv` for more details on their optional arguments.

2.2.1.1 Exercises

We have downloaded a few mortality datasets from WHO. Try to import the following datasets without looking at the answer:

- `pop` which contains reference population* and live birth data.
- `country_codes` which contains country codes

2.2.1.2 Solutions

As you might realize, there is no file extension for these files. Try opening these files with a text editor. You will see that these files are stored in comma-separated format. Hence, we will read using `readr::read_csv()`

```
pop <- readr::read_csv("data/who/pop")
```

```
## Rows: 9719 Columns: 33
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): SubDiv, Frmat
## dbl (31): Country, Admin1, Year, Sex, Pop1, Pop2, Pop3, Pop4, Pop5, Pop6, Po...
```

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

```
pop
```

```
## # A tibble: 9,719 x 33
##    Country Admin1 SubDiv  Year   Sex Frmat    Pop1   Pop2   Pop3  Pop4  Pop5
##      <dbl>  <dbl> <chr>  <dbl> <dbl> <chr>   <dbl>  <dbl>  <dbl> <dbl> <dbl>
##  1    1060     NA <NA>    1980     1 07     137100   3400  15800    NA    NA
##  2    1060     NA <NA>    1980     2 07     159000   4000  18400    NA    NA
##  3    1125     NA <NA>    1955     1 02    5051500 150300 543400    NA    NA
##  4    1125     NA <NA>    1955     2 02    5049400 145200 551000    NA    NA
##  5    1125     NA <NA>    1956     1 02    5353700 158700 576600    NA    NA
##  6    1125     NA <NA>    1956     2 02    5351400 153600 584800    NA    NA
##  7    1125     NA <NA>    1957     1 02    5403000 160300 580800    NA    NA
##  8    1125     NA <NA>    1957     2 02    5392000 155300 589400    NA    NA
##  9    1125     NA <NA>    1958     1 02    5506900 162800 592000    NA    NA
```

```
## 10    1125     NA <NA>    1958     2 02    5494400 157800 600100    NA    NA
## # ... with 9,709 more rows, and 22 more variables: Pop6 <dbl>, Pop7 <dbl>,
## #   Pop8 <dbl>, Pop9 <dbl>, Pop10 <dbl>, Pop11 <dbl>, Pop12 <dbl>, Pop13 <dbl>,
## #   Pop14 <dbl>, Pop15 <dbl>, Pop16 <dbl>, Pop17 <dbl>, Pop18 <dbl>,
## #   Pop19 <dbl>, Pop20 <dbl>, Pop21 <dbl>, Pop22 <dbl>, Pop23 <dbl>,
## #   Pop24 <dbl>, Pop25 <dbl>, Pop26 <dbl>, Lb <dbl>
```

```
country_codes <- readr::read_csv("data/who/country_codes")
```

```
## Rows: 227 Columns: 2
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): name
## dbl (1): country
```

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

```
country_codes
```

```
## # A tibble: 227 x 2
##    country name
##      <dbl> <chr>
##  1    1010 Algeria
##  2    1020 Angola
##  3    1025 Benin
##  4    1030 Botswana
##  5    1035 Burkina Faso
##  6    1040 Burundi
##  7    1045 Cameroon
##  8    1060 Cape Verde
##  9    1070 Central African Republic
## 10    1080 Chad
## # ... with 217 more rows
```

2.2.2 Excel spreadsheets

You frequently receive data files stored as Excel spreadsheets. For example, the spreadsheet file named `dentists.xls` contains information about five dentists. You can import Excel spreadsheets into R by using `read_excel()` from the `readxl` package.

```
dentists <- readxl::read_excel("data/dmus2/dentists.xls")
dentists
```

```
## # A tibble: 5 x 4
##   name              years fulltime recom
```

```
##   <chr>             <dbl>    <dbl> <dbl>
## 1 Y. Don Uflossmore  7.25        0     1
## 2 Olive Tu'Drill    10.2         1     1
## 3 Isaac O'Yerbreath 32.8         1     1
## 4 Ruth Canaale      22           1     1
## 5 Mike Avity         8.5         0     0
```

As we did with `dentists.xls`, let's import another file named `dentists2.xls` using the `read_excel()` function.

```
dentists2 <- readxl::read_excel("data/dmus2/dentists2.xls")
dentists2
```

```
## # A tibble: 4 x 2
##   name       years
##   <chr>      <dbl>
## 1 I. Sue Yoo     3
## 2 A. Dewey       8
## 3 B. Cheetem    11
## 4 C. Howe       21
```

Only 2 variables and 4 observations were imported. These are the names of lawyers. If you open the file with Excel or a spreadsheet program, you will realize that there are three sheets in the file, namely `lawyers`, `dentists`, and `Sheet2`. When there are multiple sheets in an Excel file, the default behavior for the `read_excel()` function is to import the first sheet. As such, the results from the list command above is showing a listing of lawyers because the first sheet contained information about lawyers.

We can import the contents of the sheet named `dentists` by adding the optional argument `sheet = "dentists"`, as illustrated below. This tells the function that it should specifically import the sheet named `dentists`.

```
dentists2 <- readxl::read_excel("data/dmus2/dentists2.xls", sheet = "dentists")
dentists2
```

```
## # A tibble: 5 x 4
##   name              years fulltime recom
##   <chr>             <dbl>    <dbl> <dbl>
## 1 Y. Don Uflossmore  7.25        0     1
## 2 Olive Tu'Drill    10.2         1     1
## 3 Isaac O'Yerbreath 32.8         1     1
## 4 Ruth Canaale      22           1     1
## 5 Mike Avity         8.5         0     0
```

As we would expect, the imported dataset has 4 variables and 5 observations which matches the contents of the `dentists` sheet.

Another file named `dentist3.xls` has additional information stored along with the data. In particular, column E contains notes about the dentists, and the

last row, row 7, contains column totals for some of the variables. Let's try to import this excel file using `read_excel()`.

```
dentists3 <- readxl::read_excel("data/dmus2/dentists3.xls")
```

```
## New names:
## * `` -> ...5
```

```
dentists3
```

```
## # A tibble: 6 x 5
##   name              years fulltime recom ...5
##   <chr>             <dbl>    <dbl> <dbl> <chr>
## 1 Y. Don Uflossmore  7.25        0     1 <NA>
## 2 Olive Tu'Drill    10.2         1     1 Good with children
## 3 Isaac O'Yerbreath 32.8         1     1 <NA>
## 4 Ruth Canaale      22           1     1 <NA>
## 5 Mike Avity         8.5         0     0 Has evening appointments
## 6 Total             80.8         3     4 <NA>
```

As you can see, the function imported all 5 variables and 6 observations including the `Total` row. We are only interested in the first four columns and five rows. Let's include the `range` optional argument. In Excel's terms, we want the data from cell `A1` to cell `D6`. The codes below illustrate how to import a specified cell range into R.

```
dentists3 <- readxl::read_excel("data/dmus2/dentists3.xls", range = "A1:D6")
dentists3
```

```
## # A tibble: 5 x 4
##   name              years fulltime recom
##   <chr>             <dbl>    <dbl> <dbl>
## 1 Y. Don Uflossmore  7.25        0     1
## 2 Olive Tu'Drill    10.2         1     1
## 3 Isaac O'Yerbreath 32.8         1     1
## 4 Ruth Canaale      22           1     1
## 5 Mike Avity         8.5         0     0
```

The console output now indicates that we have successfully imported the 4 variables and 5 observations we wanted.

See `?read_excel` for more details on their optional arguments.

2.2.2.1 Exercises

Import the following excel spreadsheets:

- `list_ctry_yrs_21June2021.xlsx` in `who` folder that contains A list of countries-years available for the mortality and population data.
- `miliary_tb.xlsx` in `others` folder that contains information about patients with miliary tuberculosis.

2.2.2.2 Solutions

Let's import `list_ctry_yrs_21June2021.xlsx` using `readxl::read_excel()`.

```
list_ctry_yrs <- readxl::read_excel("data/who/list_ctry_yrs_21June2021.xlsx")
```

```
## New names:
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
## * `` -> ...6
## * ...
```

```
list_ctry_yrs
```

```
## # A tibble: 5,919 x 8
##    `World Health Organization Mortal~ ...2   ...3  ...4  ...5  ...6  ...7  ...8
##    <chr>                              <chr>  <chr> <chr> <chr> <chr> <chr> <chr>
##  1 "Availability of countries-years ~ <NA>   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
##  2 "In  column \"ICD\",  \"Icd7\" da~ <NA>   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
##  3 "Consult file  \"Documentation.do~ <NA>   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
##  4 "Cells in orange  indicate the ad~ <NA>   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
##  5 "If you wish to be alerted of the~ <NA>   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
##  6  <NA>                              <NA>   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
##  7 "Country"                          name   Admi~ SubD~ Year  List  Icd   Upda~
##  8 "4005"                             Alban~ <NA>  <NA>  1987  09B   Icd9  <NA>
##  9 "4005"                             Alban~ <NA>  <NA>  1988  09B   Icd9  <NA>
## 10 "4005"                             Alban~ <NA>  <NA>  1989  09B   Icd9  <NA>
## # ... with 5,909 more rows
```

As you can see, this is not correct yet. Apparently, there are some texts in several rows above the data. If you open this file with a spreadsheet program and count them, you know there are exactly seven rows that we need to skip.

We specify another argument called `skip` in the function to illustrate this.

```
list_ctry_yrs <- readxl::read_excel("data/who/list_ctry_yrs_21June2021.xlsx", skip = 7)
```

```
list_ctry_yrs
```

```
## # A tibble: 5,912 x 8
##    Country name    Admin1 SubDiv Year  List  Icd   Update
##    <chr>   <chr>   <chr>  <lgl>  <chr> <chr> <chr> <chr>
##  1 4005    Albania <NA>   NA     1987  09B   Icd9  <NA>
##  2 4005    Albania <NA>   NA     1988  09B   Icd9  <NA>
##  3 4005    Albania <NA>   NA     1989  09B   Icd9  <NA>
##  4 4005    Albania <NA>   NA     1992  09B   Icd9  <NA>
##  5 4005    Albania <NA>   NA     1993  09B   Icd9  <NA>
##  6 4005    Albania <NA>   NA     1994  09B   Icd9  <NA>
```

```
##  7 4005    Albania <NA>   NA     1995  09B   Icd9  <NA>
##  8 4005    Albania <NA>   NA     1996  09B   Icd9  <NA>
##  9 4005    Albania <NA>   NA     1997  09B   Icd9  <NA>
## 10 4005    Albania <NA>   NA     1998  09B   Icd9  <NA>
## # ... with 5,902 more rows
```

Similary, we can do this by specifying an exact range.

```
list_ctry_yrs <- readxl::read_excel("data/who/list_ctry_yrs_21June2021.xlsx",
                                    range = "A8:H5920")
```

```
list_ctry_yrs
```

```
## # A tibble: 5,912 x 8
##    Country name    Admin1 SubDiv Year  List  Icd   Update
##    <chr>   <chr>   <chr>  <lgl>  <chr> <chr> <chr> <chr>
##  1 4005    Albania <NA>   NA     1987  09B   Icd9  <NA>
##  2 4005    Albania <NA>   NA     1988  09B   Icd9  <NA>
##  3 4005    Albania <NA>   NA     1989  09B   Icd9  <NA>
##  4 4005    Albania <NA>   NA     1992  09B   Icd9  <NA>
##  5 4005    Albania <NA>   NA     1993  09B   Icd9  <NA>
##  6 4005    Albania <NA>   NA     1994  09B   Icd9  <NA>
##  7 4005    Albania <NA>   NA     1995  09B   Icd9  <NA>
##  8 4005    Albania <NA>   NA     1996  09B   Icd9  <NA>
##  9 4005    Albania <NA>   NA     1997  09B   Icd9  <NA>
## 10 4005    Albania <NA>   NA     1998  09B   Icd9  <NA>
## # ... with 5,902 more rows
```

For the miliary tuberculosis, it's straightforward to import this dataset.

```
miliary_tb <- readxl::read_excel("./data/others/miliary_tb.xlsx")
miliary_tb
```

```
## # A tibble: 81 x 34
##      age gender diagnosis duration cough sputum fiver nightsweat fatigue
##    <dbl>  <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl>      <dbl>   <dbl>
##  1     4      2         1       10     1      0     0          0       0
##  2    43      1         1       60     1      1     1          1       0
##  3    20      2         1       45     0      0     1          0       0
##  4    22      2         1       30     0      0     1          0       0
##  5    67      1         1       40     1      0     1          0       0
##  6     6      1         1       13     0      0     1          0       0
##  7    21      2         1       14     1      1     1          0       0
##  8    22      1         1       13     0      0     1          0       0
##  9    30      1         1      120     1      1     1          0       0
## 10    36      1         1        3     1      1     1          0       0
## # ... with 71 more rows, and 25 more variables: emaciation <dbl>, nausea <dbl>,
## #   vomit <dbl>, chestpain <dbl>, shortbreath <dbl>, headache <dbl>, WBC <dbl>,
```

```
## #   RBC <dbl>, HGB <dbl>, PLT <dbl>, Perc Neutr <dbl>, ALT <dbl>, AST <dbl>,
## #   TP <dbl>, ALB <dbl>, GLB <dbl>, TBIL <dbl>, DBIL <dbl>, IBIL <dbl>,
## #   GGT <dbl>, BUN <dbl>, CREA <dbl>, ESR <dbl>, CRP <dbl>, ADA <dbl>
```

2.2.3 Stata datasets

Let's read the same dataset `dentists` using Stata `.dta` format. We use the `read_dta()` function from the `haven` package.

```
dentists <- haven::read_dta("data/dmus2/dentists.dta")
dentists
```

```
## # A tibble: 5 x 4
##   name              years fulltime recom
##   <chr>             <dbl>    <dbl> <dbl>
## 1 Y. Don Uflossmore  7.25        0     1
## 2 Olive Tu'Drill    10.2         1     1
## 3 Isaac O'Yerbreath 32.8         1     1
## 4 Ruth Canaale      22           1     1
## 5 Mike Avity         8.5         0     0
```

As you can see, we successfully read this dataset. The console output shows the information from the five dentists: their names, the years they have been practicing, whether they work full time, and whether they recommend Quaddent gum.

In addition to reading datasets from your computer, you can also read datasets stored on remote web servers. For example, `dentists.dta` is located on the Stata Press website, and you can use it by specifying its web URL address, as follows.

```
dentists <- haven::read_dta("https://www.stata-press.com/data/dmus2/dentists.dta")
```

Often our dataset might be enormous. Let's pretend that `dentists.dta` contains many variables, and we are only interested in importing just the variables `name` and `years`. We can import just these variables from `dentists.dta`, as shown below.

```
dentists <- haven::read_dta("data/dmus2/dentists.dta",
                            col_select = c("name", "years"))
dentists
```

```
## # A tibble: 5 x 2
##   name              years
##   <chr>             <dbl>
## 1 Y. Don Uflossmore  7.25
## 2 Olive Tu'Drill    10.2
## 3 Isaac O'Yerbreath 32.8
## 4 Ruth Canaale      22
```

```
## 5 Mike Avity         8.5
```

2.2.3.1 Exercises

Import all the Stata datasets under `ucla` folder. Check the folder using `dir()`.

2.2.3.2 Solutions

```
## list ulca folder
dir("data/ucla/")
```

```
## [1] "doctor_stata_dm.dta"      "hsb2.csv"
## [3] "hsb2.dta"                 "hsb2.xls"
## [5] "patient_pt1_stata_dm.dta" "patient_pt2_stata_dm.dta"
```

```
doctor_stata_dm <- haven::read_dta("data/ucla/doctor_stata_dm.dta")
doctor_stata_dm
```

```
## # A tibble: 40 x 5
##    docid experience school  lawsuits medicaid
##    <chr>      <dbl> <chr>      <dbl>    <dbl>
##  1 1-1           25 average        3    0.606
##  2 1-11          10 top            0    0.605
##  3 1-21          21 average        3    0.483
##  4 1-22          22 top            3    0.483
##  5 1-33          16 top            0    0.584
##  6 1-48          23 average        3    0.219
##  7 1-57          21 average        1    0.405
##  8 1-58          21 average        1    0.405
##  9 1-72          24 average        4    0.522
## 10 1-73          14 average        1    0.522
## # ... with 30 more rows
```

```
patient_pt1_stata_dm <- haven::read_dta("data/ucla/patient_pt1_stata_dm.dta")
patient_pt1_stata_dm
```

```
## # A tibble: 120 x 25
##    hospital hospid docid dis_date   tumorsize   co2  pain wound mobility ntumors
##    <chr>     <dbl> <chr> <date>         <dbl> <dbl> <dbl> <dbl>    <dbl>   <dbl>
##  1 UCLA          1 1-1   2009-09-06      68.0  1.53     4     4        2       0
##  2 UCLA          1 1-1   2011-01-07      64.7  1.68     2     3        2       0
##  3 UCLA          1 1-1   2010-09-04      86.4  1.45     3     3        2       0
##  4 UCLA          1 1-1   2010-06-25      53.4  1.57     3     4        2       0
##  5 UCLA          1 1-1   2009-07-01      51.7  1.42     4     5        2       0
##  6 UCLA          1 1-1   2009-03-06      78.9  1.71     3     4        2       0
##  7 UCLA          1 1-1   2010-04-15      62.9  1.54     4     4        3       2
##  8 UCLA          1 1-11  2010-07-25      73.2  1.45     4     5        9       9
##  9 UCLA          1 1-11  2009-07-12      81.2  1.55     5     5        9       0
```

```
## 10 UCLA          1 1-11  2009-08-19      61.3  1.49     7     7        8       2
## # ... with 110 more rows, and 15 more variables: nmorphine <dbl>,
## #   remission <dbl>, lungcapacity <dbl>, age <dbl>, married <dbl>,
## #   familyhx <chr>, smokinghx <chr>, sex <chr>, cancerstage <chr>,
## #   lengthofstay <dbl>, wbc <chr>, rbc <dbl>, bmi <dbl>, test1 <dbl>,
## #   test2 <dbl>
```

```
patient_pt2_stata_dm <- haven::read_dta("data/ucla/patient_pt2_stata_dm.dta")
patient_pt2_stata_dm
```

```
## # A tibble: 111 x 24
##    hospital        hospid docid dis_date   tumorsize    co2  pain wound mobility
##    <chr>            <dbl> <chr> <date>         <dbl>  <dbl> <dbl> <dbl>    <dbl>
##  1 "Cedars-Sinai"       3 3-227 2009-10-01      69.8   1.53     6     5        5
##  2 "Cedars-Sinai"       3 3-227 2010-02-18      68.0   1.69     8     4        5
##  3 "Cedars-Sinai"       3 3-227 2009-06-30      65.1   1.56     7     4        6
##  4 "Cedars-Sinai"       3 3-227 2009-11-15      71.4 -98        9     5        6
##  5 "Cedars-Sinai"       3 3-227 2010-02-17      69.5   1.68     4     5        6
##  6 "Cedars-Sinai"       3 3-227 2009-12-22      89.7   1.89     5     6        5
##  7 " Cedars-Sinai"      3 3-227 2010-04-15      73.1   1.78     6     4        5
##  8 "Cedars-Sinai"       3 3-241 2010-04-17      80.6   1.53     3     7        6
##  9 "Cedars-Sinai"       3 3-241 2010-10-12      55.1   1.71     5     7        6
## 10 "Cedars-Sinai"       3 3-241 2009-10-30      61.3   1.78     8     7        6
## # ... with 101 more rows, and 15 more variables: ntumors <dbl>,
## #   remission <dbl>, lungcapacity <dbl>, age <dbl>, married <dbl>,
## #   familyhx <chr>, smokinghx <chr>, sex <chr>, cancerstage <chr>,
## #   lengthofstay <dbl>, wbc <chr>, rbc <dbl>, bmi <dbl>, test1 <dbl>,
## #   test2 <dbl>
```

2.2.4 SPSS datasets

The `read_spss()` function from the `haven` package can import IBM SPSS Statistics `.sav` files. The examples below illustrate this by using an IBM SPSS Statistics file named `dentlab.sav`.

```
dentlab <- haven::read_spss("./data/dmus2/dentlab.sav")
dentlab
```

```
## # A tibble: 5 x 4
##   name              years      fulltime                recom
##   <chr>             <dbl>     <dbl+lbl>            <dbl+lbl>
## 1 Y. Don Uflossmore  7.25 0 [part time] 1 [recommend]
## 2 Olive Tu'Drill    10.2  1 [full time] 1 [recommend]
## 3 Isaac O'Yerbreath 32.8  1 [full time] 1 [recommend]
## 4 Ruth Canaale      22    1 [full time] 1 [recommend]
## 5 Mike Avity         8.5  0 [part time] 0 [do not recommend]
```

See `?labelled_spss` for how labelled variables in SPSS are handled by the

`haven` package in R.

Similar to `read_dta()`, we can import just the variables `name` and `years` from `dentlab.sav`, as shown below.

```
dentlab <- haven::read_spss("data/dmus2/dentlab.sav",
                            col_select = c("name", "years"))
dentlab
```

```
## # A tibble: 5 x 2
##   name              years
##   <chr>             <dbl>
## 1 Y. Don Uflossmore  7.25
## 2 Olive Tu'Drill    10.2
## 3 Isaac O'Yerbreath 32.8
## 4 Ruth Canaale      22
## 5 Mike Avity         8.5
```

Another function `read_sav()` can read both `.sav` and `.zsav` files from IBM SPSS. See `?read_spss` or `?read_sav` for more details on their optional arguments.

2.2.4.1 Exercises

[To add SPSS import exercises!]

2.2.5 Entering data directly

In R, you can construct a datasets using `as.data.frame()` or `data.frame()`. In any case, we can put small column blocks together to form a dataset as shown below.

```
name <- c("John Doe", "John Smith", "James Bond", "Jason Borne")
age <- runif(4, min = 35, max = 50) # this uses random prob distribution to generate age
ex_data <- data.frame(name, age)
ex_data
```

```
##          name      age
## 1    John Doe 45.49994
## 2  John Smith 44.06830
## 3  James Bond 35.53154
## 4 Jason Borne 41.07430
```

2.3 Saving data files

2.3.1 R Data files

2.3.2 CSV spreadsheets

2.3.3 Excel spreadsheets

2.3.4 Stata datasets

2.3.5 SPSS datasets

2.3.6 Other types

delimited types entering data directly