Lecture 13 - Importing/Exporting Data

Packages

```
# install packages
install.packages("dslabs")
install.packages("readxl")   # imports excel spreadsheets
install.packages("writexl")  # exports excel spreadsheets
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
```

```
# load libraries
library(dslabs)
library(readxl)
library(writexl)
```

Importing-Exporting Data

- Thus far, we have learned a lot about how we can use `R` to process data
  - Descriptive statistics
  - Plotting
  - Subsetting
  - Functions
  - Loops

- The datasets we used are either preloaded in a package or I have provided code to load data
- Obviously, this is not the case in real-world data analysis!

- In today's lecture, we'll learn how to
  - import data into `R` from various file formats
  - export data out of `R` onto our computers
  - navigate a file directories using `R` code

Importing Data

- Importing data is the process of bringing data that is stored in a separate environment into your own `R` working environment
- The way we import data depends how the desired data is stored externally
- Most often, the data you will encounter in your career will either be in a text file format or spreadsheet format

Text Files - `.csv`

- The majority of text file formats you will encounter will have the extension
  - .csv
  - .txt

- The file extension `.csv` means "comma separated value"

- Data within `.csv` files is stored as a table with rows and columns
- Each row of data is a separate line
- Each column of data is separated by a comma (",")

- In text files, the character string that defines a separate column, such as the "," in `.csv` is called a "***delimiter***"
- Another way to refer to `.csv` files is "comma delimited files"

- Our familiar `murders` data frame in `.csv` format can be found at the following web address:

  https://raw.githubusercontent.com/khasenst/datasets_teaching/refs/heads/main/murders.csv
- Note how each column in the `murders` dataset is separated by a comma

Importing a `.csv` file

- In `R`, `.csv` files can be imported using the `read.csv()` function

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

```
read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)
```

- `file` - filepath to the dataset
- `header` - Set to `TRUE` if the first line in your data contains the variable names
- `sep` - The delimiter in your file. This is by default a "," for comma separated files

- The script below loads the `murders` dataset from the `.csv` format into `R` as a data frame

```
# load the murders dataset as a csv file
murders_csv <- read.csv(file = "https://raw.githubusercontent.com/khasenst/datasets_te
                        header = TRUE,
                        sep = ",")
```

```
# verify it was loaded correctly
head(murders_csv)
```

A data.frame: 6 × 5

	state	abb	region	population	total
	<chr>	<chr>	<chr>	<int>	<int>
1	Alabama	AL	South	4779736	135
2	Alaska	AK	West	710231	19
3	Arizona	AZ	West	6392017	232
4	Arkansas	AR	South	2915918	93
5	California	CA	West	37253956	1257
6	Colorado	CO	West	5029196	65

```
# check its class
class(murders_csv)
```

'data.frame'

- It's as easy as that!

Importing a `.txt` file

- In `R`, `.txt` files can be imported using the `read.table()` function

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables

to fields in the file.

Usage

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
           row.names, col.names, as.is = !stringsAsFactors, tryLogical = TRUE,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = FALSE,
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

- `file` - filepath to the dataset
- `header` - Set to `TRUE` if the first line in your data contains the variable names
- `sep` - The delimiter in your file. This is by default an empty string.

- Since the default delimiter for the `read.table()` function is an empty string, which is not very useful, we must specify a delimiter
- Other than a comma, the tab is the most common delimiter when storing data as a `.txt` file
- To specify a "tab" in `R`, we use the delimiter `"\t"`

- The script below loads the `murders` dataset from the `.txt` format into `R` as a data frame from the following url

  https://raw.githubusercontent.com/khasenst/datasets_teaching/refs/heads/main/murders_tab.txt

```
# load the murders dataset as a csv file
murders_tab <- read.table(file = "https://raw.githubusercontent.com/khasenst/datasets_
                          header = TRUE,
                          sep = "\t")


# verify it was loaded correctly
head(murders_tab)
```

A data frame: 6 × 5

A data.frame: 6 × 5

	state	abb	region	population	total
	<chr>	<chr>	<chr>	<int>	<int>
1	Alabama	AL	South	4779736	135
2	Alaska	AK	West	710231	19
3	Arizona	AZ	West	6392017	232
4	Arkansas	AR	South	2915918	93
5	California	CA	West	37253956	1257
6	Colorado	CO	West	5029196	65

```
# check its class
class(murders_tab)
```

'data.frame'

- Other delimiters include period `"."`, `"|"`, `";"` etc.
- Below is an example script that imports a text file into `R` with the `"|"` delimiter from the following url

  https://raw.githubusercontent.com/khasenst/datasets_teaching/refs/heads/main/murders_pipe.txt

```
# load the murders dataset as a csv file
murders_pipe <- read.table(file = "https://raw.githubusercontent.com/khasenst/datasets
                           header = TRUE,
                           sep = "|")
```

```
# verify it was loaded correctly
head(murders_pipe)
```

A data.frame: 6 × 5

	state	abb	region	population	total
	<chr>	<chr>	<chr>	<dbl>	<int>
1	Alabama	AL	South	4779736	135
2	Alaska	AK	West	710231	19
3	Arizona	AZ	West	6392017	232
4	Arkansas	AR	South	2915918	93
5	California	CA	West	37253956	1257

6	Colorado	CO	West	5029196	65

```
# check its class
class(murders_tab)
```

```
'data.frame'
```

Excel Files and Spreadsheets

- To import data from spreadsheets, namely files in Microsoft Excel (`.xlsx`) format, we can use the `read_excel()` function from the `readxl` library

- The `read_excel()` function has many optional function arguments, but most of the time, we only need two
  - `path` - filepath to the dataset
  - `sheet` - An optional arugment specifying the sheet number you would like to import

- The script below loads the `murders` dataset from the `.xlsx` format into `R` as a data frame
- Unfortunately, we aren't able to load the excel file directly from a url
- Therefore, we must download the file and upload it to Google Colaboratory https://github.com/khasenst/datasets_teaching/blob/main/murders.xlsx

```
# import the first sheet
murders_sheet1 <- read_excel(path = "murders.xlsx",
                             sheet = 1)
```

```
head(murders_sheet1)
```

```
Error: `path` does not exist: ‘murders.xlsx’
Traceback:

1. check_file(path)
2. stop("`path` does not exist: ", sQuote(path), call. = FALSE)
3. .handleSimpleError(function (cnd)
 . {
```

```
.     watcher$capture_plot_and_output()
.     cnd <- sanitize_call(cnd)
.     watcher$push(cnd)
.     switch(on_error, continue = invokeRestart("eval_continue"),
.         stop = invokeRestart("eval_stop"), error = NULL)
. }, "`path` does not exist: ‘murders.xlsx’", base::quote(NULL))
```

Étapes suivantes : Expliquer l'erreur

```
# import the first sheet
murders_sheet2 <- read_excel(path = "murders.xlsx",
                             sheet = 2)

head(murders_sheet2)
```

```
Error: `path` does not exist: ‘murders.xlsx’
Traceback:

1. check_file(path)
2. stop("`path` does not exist: ", sQuote(path), call. = FALSE)
3. .handleSimpleError(function (cnd)
 . {
 .     watcher$capture_plot_and_output()
 .     cnd <- sanitize_call(cnd)
 .     watcher$push(cnd)
 .     switch(on_error, continue = invokeRestart("eval_continue"),
 .         stop = invokeRestart("eval_stop"), error = NULL)
 . }, "`path` does not exist: ‘murders.xlsx’", base::quote(NULL))
```

Étapes suivantes : Expliquer l'erreur

```
# combine (if you want)
murders_excel <- data.frame(murders_sheet1,
                            total = murders$total)

head(murders_excel)
```

```
Error: object 'murders_sheet1' not found
Traceback:

1. .handleSimpleError(function (cnd)
 . {
 .     watcher$capture_plot_and_output()
 .     cnd <- sanitize_call(cnd)
 .     watcher$push(cnd)
 .     switch(on_error, continue = invokeRestart("eval_continue"),
 .         stop = invokeRestart("eval_stop"), error = NULL)
 . }, "object 'murders_sheet1' not found", base::quote(eval(expr,
 .     envir)))
```

Étapes suivantes : Expliquer l'erreur

Other File Formats

- The formats discussed thus far are mostly meant to store structured data
- That is, data that takes the form of a table where rows represent observations and columns represent variables
- Next lecture, we will discuss other that can store either structured or unstructured data
  - JSON
  - html
  - xml
- There are many other more file formats relevant to special types of data, but these are outside scope of this course
  - Shapefiles for spatial data
  - `.png` or `.jpg` for images
  - `.dcm` or `.nii` for medical images

Exporting Data

- After importing data and performing your analysis, you may want to
  - export a new version of your dataset that might be more organized or contain additional data
  - export a data summary for external reference
- We can export our data outside of `R` using very similar functions!
- The majority of the time, you will want to export your data as a `.csv` file

Exporting a `.csv` file

- We can export an `R` data frame as a `.csv` file using the function `write.csv()`
- There are three arguments that are commonly used
  - `x` - the `R` data frame to exported
  - `file` - the filename to where the data frame will be exported
  - `row.names` - Whether or not you would like the row names of the data frame to also be exported. I often set this to `FALSE`, unless there is a special reason to include the row names
- The script below exports the `murders` data frame as a `.csv` file

```
# export data as a csv file
write.csv(x    = murders,
          file = "murders_as_csv.csv",
          row.names = FALSE)
```

Exporting a `.txt` file

- We can export an `R` data frame as a `.txt` file using the function `write.table()`
- There are four arguments that are commonly used
  - `x` - the `R` data frame to exported
  - `file` - the filename to where the data frame will be exported
  - `sep` - the delimiter when exporting the data frame
  - `row.names` - Whether or not you would like the row names of the data frame to also be exported. I often set this to `FALSE`, unless there is a special reason to include the row names

```
# export data as a txt file with tab delimiter
write.table(x         = murders,
            file      = "murders_as_tab.txt",
            sep       = "\t",
            row.names = FALSE)
```

```
# export data as a txt file with | delimiter
write.table(x         = murders,
            file      = "murders_as_pipe.txt",
            sep       = "|",
            row.names = FALSE)
```

Exporting a `.xlsx` file

- We can export an `R` data frame as a `.xlsx` file using the function `write_xlsx()`
  - `x` - the `R` data frame to exported
  - `path` - the filename to where the data frame will be exported

```
write_xlsx(x    = murders,
           path = "murders_as_excel.xlsx")
```

Navigating File Directories

- Thus far, we have been loading data available from the internet
- ...but what if your data is stored locally or in a special server?

- A useful skill is the ability to navigate to different files on your own system using scripts/code

- This is analogous to clicking through folders, except we do this with code!

- Below are functions that can help you navigate your directories/file system

`getwd()`

- The `getwd()` function "gets" your current working directory
- That is, `getwd()` shows the location that `R` is currently importing/exporting data from/to

- By default, Google Colaboratory sets our working directory to the folder `"/content"`

```
# get our current working directory
getwd()
```

```
'/content'
```

`list.files()`

- The `list.files()` function lists the files and folders in a directory
- If no arguments are given, the `list.files()` function simply lists the files in your current working directory

```
# list files in the current working directory
print(list.files())
```

```
 [1] "murders_as_csv.csv"    "murders_as_excel.xlsx" "murders_as_pipe.txt"
 [4] "murders_as_tab.txt"    "sample_data"
```

```
# list files in a specified working directory
print(list.files("/content"))
```

```
     [1] "murders_as_csv.csv"    "murders_as_excel.xlsx" "murders_as_pipe.txt"
     [4] "murders_as_tab.txt"    "sample_data"
```

dir.create()

- The `dir.create()` function can create a new folder from a specified filepath
- The script below creates a folder called `"my_data"` in the existing folder `"content"`
- To do so, we specify the filepath `"/content/my_data"`

```
# create a new folder within a directory
dir.create("/content/my_data")
```

- Note that attempting to creating a directory that already exists produces a warning message

```
# create a new folder within a directory when it already exists
dir.create("/content/my_data")
```

```
     Warning message in dir.create("/content/my_data"):
     “'/content/my_data' already exists”
```

setwd()

- The `setwd()` function "sets" your working directory to a specified working directory
- We can use `setwd()` to navigate to the recently created `"my_data"` folder

```
# set our current working directory
setwd("/content/my_data")
```

```
# check our current working directory
getwd()
```

'/content/my_data'

```
# list files in the current directory
list.files()

# print the number of
print(length(list.files()))
```

[1] 0

- Currently, there are no files in the directory!

file.path()

- Notice how we separate each folder and file location using a forward slash `/`
- We can create filepaths manually by typing the entire character string with forward slashes
- Or we can use the `file.path()` function to do this for us!

```
# type it out ourselves
my_path <- "/content/my_data"
my_path
```

'/content/my_data'

```
# using list.files()
my_path <- file.path("/content", "my_data")
my_path
```

'/content/my_data'

Example

- Let's store our `mpg` data frames in different file formats into their own respective

- Let's store our "murders" data frames in different file formats into their own respective folders

```
# check our current working directory
getwd()
```

'/content/my_data'

```
# set our working directory
setwd("/content")
```

```
# check we are in the correct directory
getwd()
```

'/content'

- Use `dir.create()` to create folders for each type file format (`.csv`, `.txt`, `.xlsx`)

```
# csv folder
dir.create(file.path("/content", "csv"))

# txt folder
dir.create(file.path("/content", "txt"))

# xlsx folder
dir.create(file.path("/content", "xlsx"))
```

```
# check our work
print(list.files())
```

```
[1] "csv"                   "murders_as_csv.csv"    "murders_as_excel.xlsx"
[4] "murders_as_pipe.txt"   "murders_as_tab.txt"    "my_data"
[7] "sample_data"           "txt"                   "xlsx"
```

- Write files to the Colaboratory computer

```
# create csv path
csv_path <- file.path("/content", "csv")
csv_path
```

'/content/csv'

```
# export data to file
```

```
    .
write.csv(x    = murders,
          file = file.path(csv_path, "murders_as_csv.csv"),
          row.names = FALSE)
```

```
# create txt path
txt_path <- file.path("/content", "txt")

# export data to tab delimited file
write.table(x         = murders,
            file      = file.path(txt_path, "murders_as_tab.txt"),
            sep       = "\t",
            row.names = FALSE)

# export data to pipe delimited file
write.table(x         = murders,
            file      = file.path(txt_path, "murders_as_pipe.txt"),
            sep       = "|",
            row.names = FALSE)
```

```
# create xlsx path
xlsx_path <- file.path("/content", "xlsx")

# export data to file
write_xlsx(x    = murders,
           path = file.path(xlsx_path, "xlsx_file.xlsx"))
```