**Chapter 1: Importing data from flat files with utils**

* 1. **read.csv**

The utils package, which is automatically loaded in your R session on startup, can import CSV files with the [**read.csv()**](http://www.rdocumentation.org/packages/utils/functions/read.table) function.

In this exercise, you'll be working with [**swimming\_pools.csv**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/swimming_pools.csv); it contains data on swimming pools in Brisbane, Australia (Source: [**data.gov.au**](https://data.gov.au/dataset/swimming-pools-brisbane-city-council)). The file contains the column names in the first row. It uses a comma to separate values within rows.

Type **[dir()](http://www.rdocumentation.org/packages/base/functions/list.files" \t "_blank)** in the console to list the files in your working directory. You'll see that it contains swimming\_pools.csv, so you can start straight away.

**1.2 stringsAsFactor**

With stringsAsFactors, you can tell R whether it should convert strings in the flat file to factors.

For all importing functions in the utils package, this argument is TRUE, which means that you import strings as factors. This only makes sense if the strings you import represent categorical variables in R. If you set stringsAsFactors to FALSE, the data frame columns corresponding to strings in your text file will be character.

You'll again be working with the [**swimming\_pools.csv**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/swimming_pools.csv) file. It contains two columns (Name and Address), which shouldn't be factors.

**1.3 Read.delim**

Aside from .csv files, there are also the .txt files which are basically text files. You can import these functions with **[read.delim()](http://www.rdocumentation.org/packages/utils/functions/read.table" \t "_blank)**. By default, it sets the sep argument to "\t" (fields in a record are delimited by tabs) and the header argument to TRUE (the first row contains the field names).

In this exercise, you will import [**hotdogs.txt**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/hotdogs.txt), containing information on sodium and calorie levels in different hotdogs (Source: [**UCLA**](http://wiki.stat.ucla.edu/socr/index.php/SOCR_012708_ID_Data_HotDogs)). The dataset has 3 variables, but the variable names are not available in the first line of the file. The file uses tabs as field separators.

**1,4 Read.table**

If you're dealing with more exotic flat file formats, you'll want to use **[read.table()](http://www.rdocumentation.org/packages/utils/functions/read.table" \t "_blank)**. It's the most basic importing function; you can specify tons of different arguments in this function. Unlike [**read.csv()**](http://www.rdocumentation.org/packages/utils/functions/read.table) and **[read.delim()](http://www.rdocumentation.org/packages/utils/functions/read.table" \t "_blank)**, the header argument defaults to FALSE and the sep argument is "" by default.

Up to you again! The data is still [**hotdogs.txt**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/hotdogs.txt). It has no column names in the first row, and the field separators are tabs. This time, though, the file is in the data folder inside your current working directory. A variable path with the location of this file is already coded for you.

**1.5 Arguments**

Lily and Tom are having an argument because they want to share a hot dog but they can't seem to agree on which one to choose. After some time, they simply decide that they will have one each. Lily wants to have the one with the fewest calories while Tom wants to have the one with the most sodium.

Next to calories and sodium, the hotdogs have one more variable: type. This can be one of three things: Beef, Meat, or Poultry, so a categorical variable: a factor is fine.

**1.6 Column classes**

Next to column names, you can also specify the column types or column classes of the resulting data frame. You can do this by setting the colClasses argument to a vector of strings representing classes:

read.delim("my\_file.txt",

colClasses = c("character",

"numeric",

"logical"))

This approach can be useful if you have some columns that should be factors and others that should be characters. You don't have to bother with stringsAsFactors anymore; just state for each column what the class should be.

If a column is set to "NULL" in the colClasses vector, this column will be skipped and will not be loaded into the data frame.

**Chapter 2: readr & data.table**

**2.1 Read\_csv**

CSV files can be imported with **[read\_csv()](http://www.rdocumentation.org/packages/readr/versions/1.0.0/topics/read_delim" \t "_blank)**. It's a wrapper function around **[read\_delim()](http://www.rdocumentation.org/packages/readr/versions/1.0.0/topics/read_delim" \t "_blank)** that handles all the details for you. For example, it will assume that the first row contains the column names.

The dataset you'll be working with here is [**potatoes.csv**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/potatoes.csv). It gives information on the impact of storage period and cooking on potatoes' flavor. It uses commas to delimit fields in a record, and contains column names in the first row. The file is available in your workspace. Remember that you can inspect your workspace with dir().

**2.2 Read\_tsv**

Where you use read\_csv() to easily read in CSV files, you use **[read\_tsv()](http://www.rdocumentation.org/packages/readr/versions/1.0.0/topics/read_delim" \t "_blank)** to easily read in TSV files. TSV is short for tab-separated values.

This time, the potatoes data comes in the form of a tab-separated values file; [**potatoes.txt**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/potatoes.txt) is available in your workspace. In contrast to potatoes.csv, this file does **not** contain columns names in the first row, though.

There's a vector properties that you can use to specify these column names manually.

**2.3 Read\_delim**

Just as **[read.table()](http://www.rdocumentation.org/packages/utils/functions/read.table" \t "_blank)** was the main utils function, **[read\_delim()](http://www.rdocumentation.org/packages/readr/versions/1.0.0/topics/read_delim" \t "_blank)** is the main readr function.

[**read\_delim()**](http://www.rdocumentation.org/packages/readr/versions/1.0.0/topics/read_delim) takes two mandatory arguments:

* file: the file that contains the data
* delim: the character that separates the values in the data file

You'll again be working [**potatoes.txt**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/potatoes.txt); the file uses tabs ("\t") to delimit values and does **not** contain column names in its first line. It's available in your working directory so you can start right away. As before, the vector properties is available to set the col\_names.

**2.4 Skip and n\_max**

Through skip and n\_max you can control *which part* of your flat file you're actually importing into R.

* skip specifies the number of lines you're ignoring in the flat file before actually starting to import data.
* n\_max specifies the number of lines you're actually importing.

Say for example you have a CSV file with 20 lines, and set skip = 2 and n\_max = 3, you're only reading in lines 3, 4 and 5 of the file.

Watch out: Once you skip some lines, you also skip the first line that can contain column names!

[**potatoes.txt**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/potatoes.txt), a flat file with tab-delimited records and without column names, is available in your workspace.

**2.5 Col\_types**

You can also specify which types the columns in your imported data frame should have. You can do this with col\_types. If set to NULL, the default, functions from the readr package will try to find the correct types themselves. You can manually set the types with a string, where each character denotes the class of the column: character, double, integer and logical. \_ skips the column as a whole.

[**potatoes.txt**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/potatoes.txt), a flat file with tab-delimited records and without column names, is again available in your workspace.

**2.6 Col\_types with collectors**

Another way of setting the types of the imported columns is using **collectors**. Collector functions can be passed in a [**list()**](http://www.rdocumentation.org/packages/base/functions/list) to the col\_types argument of read\_ functions to tell them how to interpret values in a column.

For a complete list of collector functions, you can take a look at the [**collector**](https://www.rdocumentation.org/packages/readr/topics/collector) documentation. For this exercise you will need two collector functions:

* col\_integer(): the column should be interpreted as an integer.
* col\_factor(levels, ordered = FALSE): the column should be interpreted as a factor with levels.

In this exercise, you will work with [**hotdogs.txt**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/hotdogs.txt), which is a tab-delimited file without column names in the first row.

**2.7 Fread**

You still remember how to use **[read.table()](http://www.rdocumentation.org/packages/utils/functions/read.table" \t "_blank)**, right? Well, **[fread()](http://www.rdocumentation.org/packages/data.table/functions/fread" \t "_blank)** is a function that does the same job with very similar arguments. It is extremely easy to use and blazingly fast! Often, simply specifying the path to the file is enough to successfully import your data.

Don't take our word for it, try it yourself! You'll be working with the [**potatoes.csv**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/potatoes.csv) file, that's available in your workspace. Fields are delimited by commas, and the first line contains the column names.

**2.8 Fread: more advanced use**

Now that you know the basics about **[fread()](http://www.rdocumentation.org/packages/data.table/functions/fread" \t "_blank)**, you should know about two arguments of the function: drop and select, to drop or select variables of interest.

Suppose you have a dataset that contains 5 variables and you want to keep the first and fifth variable, named "a" and "e". The following options will all do the trick:

fread("path/to/file.txt", drop = 2:4)

fread("path/to/file.txt", select = c(1, 5))

fread("path/to/file.txt", drop = c("b", "c", "d"))

fread("path/to/file.txt", select = c("a", "e"))

Let's stick with potatoes since we're particularly fond of them here at DataCamp. The data is again available in the file [**potatoes.csv**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/potatoes.csv), containing comma-separated records.

**Chapter 3: Importing Excel data**

**3.1 List the sheets of an Excel file**

Before you can start importing from Excel, you should find out which sheets are available in the workbook. You can use the **[excel\_sheets()](https://cran.r-project.org/web/packages/readxl/readxl.pdf" \t "_blank)** function for this.

You will find the Excel file [**urbanpop.xlsx**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xlsx) in your working directory (type **[dir()](http://www.rdocumentation.org/packages/base/functions/list.files" \t "_blank)** to see it). This dataset contains urban population metrics for practically all countries in the world throughout time (Source: **[Gapminder](http://www.gapminder.org/" \t "_blank)**). It contains three sheets for three different time periods. In each sheet, the first row contains the column names.

**3.2 Import an Excel sheet**

Now that you know the names of the sheets in the Excel file you want to import, it is time to import those sheets into R. You can do this with the **[read\_excel()](https://cran.r-project.org/web/packages/readxl/readxl.pdf" \t "_blank)** function. Have a look at this recipe:

data <- read\_excel("data.xlsx", sheet = "my\_sheet")

This call simply imports the sheet with the name "my\_sheet" from the "data.xlsx" file. You can also pass a number to the sheet argument; this will cause **[read\_excel()](https://cran.r-project.org/web/packages/readxl/readxl.pdf" \t "_blank)** to import the sheet with the given sheet number. sheet = 1 will import the first sheet, sheet = 2 will import the second sheet, and so on.

In this exercise, you'll continue working with the [**urbanpop.xlsx**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xlsx) file.

**3.3 Reading a workbook**

In the previous exercise you generated a list of three Excel sheets that you imported. However, loading in every sheet manually and then merging them in a list can be quite tedious. Luckily, you can automate this with **[lapply()](http://www.rdocumentation.org/packages/base/functions/lapply" \t "_blank)**. If you have no experience with **[lapply()](http://www.rdocumentation.org/packages/base/functions/lapply" \t "_blank)**, feel free to take [**Chapter 4 of the Intermediate R course**](https://campus.datacamp.com/courses/intermediate-r/chapter-4-the-apply-family?ex=1).

Have a look at the example code below:

my\_workbook <- lapply(excel\_sheets("data.xlsx"),

read\_excel,

path = "data.xlsx")

The **[read\_excel()](https://cran.r-project.org/web/packages/readxl/readxl.pdf" \t "_blank)** function is called multiple times on the "data.xlsx" file and each sheet is loaded in one after the other. The result is a list of data frames, each data frame representing one of the sheets in data.xlsx.

You're still working with the [**urbanpop.xlsx**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xlsx) file.

**3.4 The col\_names argument**

Apart from path and sheet, there are several other arguments you can specify in **[read\_excel()](https://cran.r-project.org/web/packages/readxl/readxl.pdf" \t "_blank)**. One of these arguments is called col\_names.

By default it is TRUE, denoting whether the first row in the Excel sheets contains the column names. If this is not the case, you can set col\_names to FALSE. In this case, R will choose column names for you. You can also choose to set col\_names to a character vector with names for each column. It works exactly the same as in the readr package.

You'll be working with the [**urbanpop\_nonames.xlsx**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop_nonames.xlsx) file. It contains the same data as [**urbanpop.xlsx**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xlsx) but has no column names in the first row of the excel sheets.

**3.5 The skip argument**

Another argument that can be very useful when reading in Excel files that are less tidy, is skip. With skip, you can tell R to ignore a specified number of rows inside the Excel sheets you're trying to pull data from. Have a look at this example:

read\_excel("data.xlsx", skip = 15)

In this case, the first 15 rows in the first sheet of "data.xlsx" are ignored.

If the first row of this sheet contained the column names, this information will also be ignored by readxl. Make sure to set col\_names to FALSE or manually specify column names in this case!

The file [**urbanpop.xlsx**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xlsx) is available in your directory; it has column names in the first rows.

**3.6 Import a local file**

In this part of the chapter you'll learn how to import .xls files using the gdata package. Similar to the readxl package, you can import single Excel sheets from Excel sheets to start your analysis in R.

You'll be working with the [**urbanpop.xls**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xls) dataset, the .xls version of the Excel file you've been working with before. It's available in your current working directory.

**3.7 Read.xls() wraps around read.table()**

Remember how [**read.xls()**](http://www.rdocumentation.org/packages/gdata/functions/read.xls) actually works? It basically comes down to two steps: converting the Excel file to a .csv file using a Perl script, and then reading that .csv file with the [**read.csv()**](http://www.rdocumentation.org/packages/utils/functions/read.table) function that is loaded by default in R, through the utils package.

This means that all the options that you can specify in [**read.csv()**](http://www.rdocumentation.org/packages/utils/functions/read.table), can also be specified in [**read.xls()**](http://www.rdocumentation.org/packages/gdata/functions/read.xls).

The [**urbanpop.xls**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xls) dataset is already available in your workspace. It's still comprised of three sheets, and has column names in the first row of each sheet.

**3.8 Work that Excel data!**

Now that you can read in Excel data, let's try to clean and merge it. You already used the **[cbind()](http://www.rdocumentation.org/packages/base/functions/cbind" \t "_blank)** function some exercises ago. Let's take it one step further now.

The [**urbanpop.xls**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xls) dataset is available in your working directory. The file still contains three sheets, and has column names in the first row of each sheet.

**Chapter 4: Reproducible Excel work with XLConnect**

**4.1 Connect to a workbook**

When working with XLConnect, the first step will be to load a workbook in your R session with **[loadWorkbook()](http://www.rdocumentation.org/packages/XLConnect/functions/loadWorkbook" \t "_blank)**; this function will build a "bridge" between your Excel file and your R session.

In this and the following exercises, you will continue to work with [**urbanpop.xlsx**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xlsx), containing urban population data throughout time. The Excel file is available in your current working directory.

**4.2 List and read Excel sheets**

Just as readxl and gdata, you can use XLConnect to import data from Excel file into R.

To list the sheets in an Excel file, use **[getSheets()](http://www.rdocumentation.org/packages/XLConnect/functions/getSheets-methods" \t "_blank)**. To actually import data from a sheet, you can use **[readWorksheet()](http://www.rdocumentation.org/packages/XLConnect/functions/readWorksheet-methods" \t "_blank)**. Both functions require an XLConnect workbook object as the first argument.

You'll again be working with [**urbanpop.xlsx**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xlsx). The my\_book object that links to this Excel file has already been created.

**4.3 Customize readWorksheet**

To get a clear overview about [**urbanpop.xlsx**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xlsx) without having to open up the Excel file, you can execute the following code:

my\_book <- loadWorkbook("urbanpop.xlsx")

sheets <- getSheets(my\_book)

all <- lapply(sheets, readWorksheet, object = my\_book)

str(all)

Suppose we're only interested in urban population data of the years 1968, 1969 and 1970. The data for these years is in the columns 3, 4, and 5 of the second sheet. Only selecting these columns will leave us in the dark about the actual countries the figures belong to.

**4.4 Add worksheet**

Where readxl and gdata were only able to import Excel data, XLConnect's approach of providing an actual interface to an Excel file makes it able to edit your Excel files from inside R. In this exercise, you'll create a new sheet. In the next exercise, you'll populate the sheet with data, and save the results in a new Excel file.

You'll continue to work with [**urbanpop.xlsx**](http://s3.amazonaws.com/assets.datacamp.com/production/course_1477/datasets/urbanpop.xlsx). The my\_book object that links to this Excel file is already available.

**4.5 Populate worksheet**

The first step of creating a sheet is done; let's populate it with some data now! summ, a data frame with some summary statistics on the two Excel sheets is already coded so you can take it from there.

**4.6 Renaming sheets**

Come to think of it, "data\_summary" is not an ideal name. As the summary of these excel sheets is always data-related, you simply want to name the sheet "summary".

The workspace already contains a workbook, my\_book, that refers to an Excel file with 4 sheets: the three data sheets, and the "data\_summary" sheet.

**4.7 Removing sheets**

After presenting the new Excel sheet to your peers, it appears not everybody is a big fan. Why summarize sheets and store the info in Excel if all the information is implicitly available? To hell with it, just remove the entire fourth sheet!