Exercise 2

First Assigment: Introduction to Importing Data in R

Andreas Merckel, 00746397

17.03.2021

read.csv

In this exercise we learned the usage of the R function read.csv(), which is used to import comma-separated values into R. str() gives us an overview over the imported data.

```
# Import swimming_pools.csv: pools
pools <- read.csv("data/swimming_pools.csv")
# Print the structure of pools
str(pools)
```

```
## 'data.frame':    20 obs. of  4 variables:
##  $ Name     : Factor w/ 20 levels "Acacia Ridge Leisure Centre",..: 1 2 3 4 5 6 19 7 8 9 ...
##  $ Address  : Factor w/ 20 levels "1 Fairlead Crescent, Manly",..: 5 20 18 10 9 11 6 15 12 17 ...
##  $ Latitude : num  -27.6 -27.6 -27.6 -27.5 -27.4 ...
##  $ Longitude: num  153 153 153 153 153 ...
```

stringsAsFactors

In this exercise we learned how we can tell read.csv() to convert any strings in the underlying imported file to factors or not with the corresponding option. Importing strings as factors makes sense if the strings you import are supposed to represent categorical variables in R.

```
# Import swimming_pools.csv correctly: pools
pools <- read.csv("data/swimming_pools.csv", stringsAsFactors = FALSE)

# Check the structure of pools
str(pools)
```

```
## 'data.frame':    20 obs. of  4 variables:
##  $ Name     : chr  "Acacia Ridge Leisure Centre" "Bellbowrie Pool" "Carole Park" "Centenary Pool (in
##  $ Address  : chr  "1391 Beaudesert Road, Acacia Ridge" "Sugarwood Street, Bellbowrie" "Cnr Boundary
##  $ Latitude : num  -27.6 -27.6 -27.6 -27.5 -27.4 ...
##  $ Longitude: num  153 153 153 153 153 ...
```

Any changes?

A multiple choice question regarding the stringsAsFactors option, the correct answer is `Two variables: Name and Address.`

read.delim

read.delim can be used when you need to customize the delimiter.

```
# Import hotdogs.txt: hotdogs
hotdogs <- read.delim(header=FALSE, "data/hotdogs.txt")

# Summarize hotdogs
summary(hotdogs)
```

```
##        V1           V2              V3
##  Beef   :20   Min.   : 86.0   Min.   :144.0
##  Meat   :17   1st Qu.:132.0   1st Qu.:362.5
##  Poultry:17   Median :145.0   Median :405.0
##               Mean   :145.4   Mean   :424.8
##               3rd Qu.:172.8   3rd Qu.:503.5
##               Max.   :195.0   Max.   :645.0
```

read.table

read.table can be used when there is the need to have even more options regarding the underlying file format separators.

```
# Path to the hotdogs.txt file: path
path <- file.path("data", "hotdogs.txt")

# Import the hotdogs.txt file: hotdogs
hotdogs <- read.table(path, head=FALSE,
                      sep = '\t',
                      col.names = c("type", "calories", "sodium"))

# Call head() on hotdogs
head(hotdogs)
```

```
##   type calories sodium
## 1 Beef      186    495
## 2 Beef      181    477
## 3 Beef      176    425
## 4 Beef      149    322
## 5 Beef      184    482
## 6 Beef      190    587
```

Arguments

This exercise is putting the above together: We are using the possible arguments in read.delim() to read in real-world data. We learn that we can give custom names to eventual columns in our dataset, by providing an appropriate vector.

```
# Finish the read.delim() call
hotdogs <- read.delim("data/hotdogs.txt", header = FALSE, col.names = c("type", "calories", "sodium"))

# Select the hot dog with the least calories: lily
lily <- hotdogs[which.min(hotdogs$calories), ]

# Select the observation with the most sodium: tom
tom <- hotdogs[which.max(hotdogs$sodium), ]

# Print lily and tom
lily
```

```
##       type calories sodium
## 50 Poultry       86    358
```

```
tom
```

```
##    type calories sodium
## 15 Beef      190    645
```

Column classes

Additionally to naming columns, we can also decide the types, with another vector called colClasses.

```
# Previous call to import hotdogs.txt
hotdogs <- read.delim("data/hotdogs.txt", header = FALSE, col.names = c("type", "calories", "sodium"))

# Display structure of hotdogs
str(hotdogs)
```

```
## 'data.frame':    54 obs. of  3 variables:
##  $ type    : Factor w/ 3 levels "Beef","Meat",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ calories: int  186 181 176 149 184 190 158 139 175 148 ...
##  $ sodium  : int  495 477 425 322 482 587 370 322 479 375 ...
```

```
# Edit the colClasses argument to import the data correctly: hotdogs2
hotdogs2 <- read.delim("data/hotdogs.txt", header = FALSE,
                       col.names = c("type", "calories", "sodium"),
                       colClasses = c("factor", "NULL", "numeric"))

# Display structure of hotdogs2
str(hotdogs2)
```

```
## 'data.frame':    54 obs. of  2 variables:
##  $ type  : Factor w/ 3 levels "Beef","Meat",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ sodium: num  495 477 425 322 482 587 370 322 479 375 ...
```

read_csv

This function comes from the readr package and is a wrapper function for read.csv(), so i.e. it is used for common use-cases instead of having to fiddle around with all the options in read.csv().

```
# Load the readr package
library(readr)

# Import potatoes.csv with read_csv(): potatoes
potatoes <- read_csv("data/potatoes.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   area = col_double(),
##   temp = col_double(),
##   size = col_double(),
##   storage = col_double(),
##   method = col_double(),
##   texture = col_double(),
##   flavor = col_double(),
##   moistness = col_double()
## )
```

read_tsv

Like read_csv(), read_tsv() is a wrapper function for tab-separated values.

```
# readr is already loaded

# Column names
properties <- c("area", "temp", "size", "storage", "method",
                "texture", "flavor", "moistness")

# Import potatoes.txt: potatoes
potatoes <- read_tsv("data/potatoes.txt", col_names=properties)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   area = col_double(),
##   temp = col_double(),
##   size = col_double(),
##   storage = col_double(),
##   method = col_double(),
##   texture = col_double(),
##   flavor = col_double(),
##   moistness = col_double()
## )
```

```
# Call head() on potatoes
head(potatoes)
```

```
## # A tibble: 6 x 8
##    area  temp  size storage method texture flavor moistness
##   <dbl> <dbl> <dbl>   <dbl>  <dbl>   <dbl>  <dbl>     <dbl>
## 1     1     1     1       1      1     2.9    3.2       3
## 2     1     1     1       1      2     2.3    2.5       2.6
## 3     1     1     1       1      3     2.5    2.8       2.8
## 4     1     1     1       1      4     2.1    2.9       2.4
## 5     1     1     1       1      5     1.9    2.8       2.2
## 6     1     1     1       2      1     1.8    3         1.7
```

read_delim

As the above, read_delim comes from the readr package and is corresponding to read.table() in its functionality.

```
# readr is already loaded

# Column names
properties <- c("area", "temp", "size", "storage", "method",
                "texture", "flavor", "moistness")

# Import potatoes.txt using read_delim(): potatoes
potatoes <- read_delim("data/potatoes.txt", delim="\t", col_names=properties)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   area = col_double(),
##   temp = col_double(),
```

```
##   size = col_double(),
##   storage = col_double(),
##   method = col_double(),
##   texture = col_double(),
##   flavor = col_double(),
##   moistness = col_double()
## )
```

```
# Print out potatoes
potatoes
```

```
## # A tibble: 160 x 8
##     area  temp  size storage method texture flavor moistness
##    <dbl> <dbl> <dbl>   <dbl>  <dbl>   <dbl>  <dbl>     <dbl>
##  1     1     1     1       1      1     2.9    3.2       3
##  2     1     1     1       1      2     2.3    2.5       2.6
##  3     1     1     1       1      3     2.5    2.8       2.8
##  4     1     1     1       1      4     2.1    2.9       2.4
##  5     1     1     1       1      5     1.9    2.8       2.2
##  6     1     1     1       2      1     1.8    3         1.7
##  7     1     1     1       2      2     2.6    3.1       2.4
##  8     1     1     1       2      3     3      3         2.9
##  9     1     1     1       2      4     2.2    3.2       2.5
## 10     1     1     1       2      5     2      2.8       1.9
## # ... with 150 more rows
```

skip and n_max

These options for read_tsv help us to specify which part of our data we want to import.

```
# readr is already loaded

# Column names
properties <- c("area", "temp", "size", "storage", "method",
                "texture", "flavor", "moistness")

# Import 5 observations from potatoes.txt: potatoes_fragment
potatoes_fragment <- read_tsv("data/potatoes.txt", skip = 6, n_max = 5, col_names = properties)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   area = col_double(),
##   temp = col_double(),
##   size = col_double(),
##   storage = col_double(),
##   method = col_double(),
##   texture = col_double(),
##   flavor = col_double(),
##   moistness = col_double()
## )
```

col_types

col_types from the readr package is used to specify the types of the columns.

```
# readr is already loaded

# Column names
properties <- c("area", "temp", "size", "storage", "method",
                "texture", "flavor", "moistness")

# Import all data, but force all columns to be character: potatoes_char
potatoes_char <- read_tsv("data/potatoes.txt", col_types = "cccccccc", col_names = properties)

# Print out structure of potatoes_char
str(potatoes_char)
```

```
## spec_tbl_df [160 x 8] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ area     : chr [1:160] "1" "1" "1" "1" ...
##  $ temp     : chr [1:160] "1" "1" "1" "1" ...
##  $ size     : chr [1:160] "1" "1" "1" "1" ...
##  $ storage  : chr [1:160] "1" "1" "1" "1" ...
##  $ method   : chr [1:160] "1" "2" "3" "4" ...
##  $ texture  : chr [1:160] "2.9" "2.3" "2.5" "2.1" ...
##  $ flavor   : chr [1:160] "3.2" "2.5" "2.8" "2.9" ...
##  $ moistness: chr [1:160] "3.0" "2.6" "2.8" "2.4" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   area = col_character(),
##   ..   temp = col_character(),
##   ..   size = col_character(),
##   ..   storage = col_character(),
##   ..   method = col_character(),
##   ..   texture = col_character(),
##   ..   flavor = col_character(),
##   ..   moistness = col_character()
##   .. )
```

col_types with collectors

The argument col_types can also be called with a list to set the types.

```
# readr is already loaded

# Import without col_types
hotdogs <- read_tsv("data/hotdogs.txt", col_names = c("type", "calories", "sodium"))
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   type = col_character(),
##   calories = col_double(),
##   sodium = col_double()
## )
```

```
# Display the summary of hotdogs
summary(hotdogs)
```

```
##      type              calories         sodium
##  Length:54          Min.   : 86.0   Min.   :144.0
##  Class :character   1st Qu.:132.0   1st Qu.:362.5
```

```
##  Mode  :character   Median :145.0   Median :405.0
##                     Mean   :145.4   Mean   :424.8
##                     3rd Qu.:172.8   3rd Qu.:503.5
##                     Max.   :195.0   Max.   :645.0
```

```
# The collectors you will need to import the data
fac <- col_factor(levels = c("Beef", "Meat", "Poultry"))
int <- col_integer()

# Edit the col_types argument to import the data correctly: hotdogs_factor
hotdogs_factor <- read_tsv("data/hotdogs.txt",
                           col_names = c("type", "calories", "sodium"),
                           col_types = list(fac, int, int))

# Display the summary of hotdogs_factor
summary(hotdogs_factor)
```

```
##       type       calories         sodium
##  Beef   :20   Min.   : 86.0   Min.   :144.0
##  Meat   :17   1st Qu.:132.0   1st Qu.:362.5
##  Poultry:17   Median :145.0   Median :405.0
##               Mean   :145.4   Mean   :424.8
##               3rd Qu.:172.8   3rd Qu.:503.5
##               Max.   :195.0   Max.   :645.0
```

fread

fread is a function from the data.table package and corresponds to read.table() from above. It is better at predicting the correct types and values and faster than read.table()

```
# load the data.table package
library(data.table);

# Import potatoes.csv with fread(): potatoes
potatoes <- fread("data/potatoes.csv");

# Print out potatoes
potatoes
```

```
##       area temp size storage method texture flavor moistness
##    1:    1    1    1       1      1     2.9    3.2       3.0
##    2:    1    1    1       1      2     2.3    2.5       2.6
##    3:    1    1    1       1      3     2.5    2.8       2.8
##    4:    1    1    1       1      4     2.1    2.9       2.4
##    5:    1    1    1       1      5     1.9    2.8       2.2
##   ---
## 156:    2    2    2       4      1     2.7    3.3       2.6
## 157:    2    2    2       4      2     2.6    2.8       2.3
## 158:    2    2    2       4      3     2.5    3.1       2.6
## 159:    2    2    2       4      4     3.4    3.3       3.0
## 160:    2    2    2       4      5     2.5    2.8       2.3
```

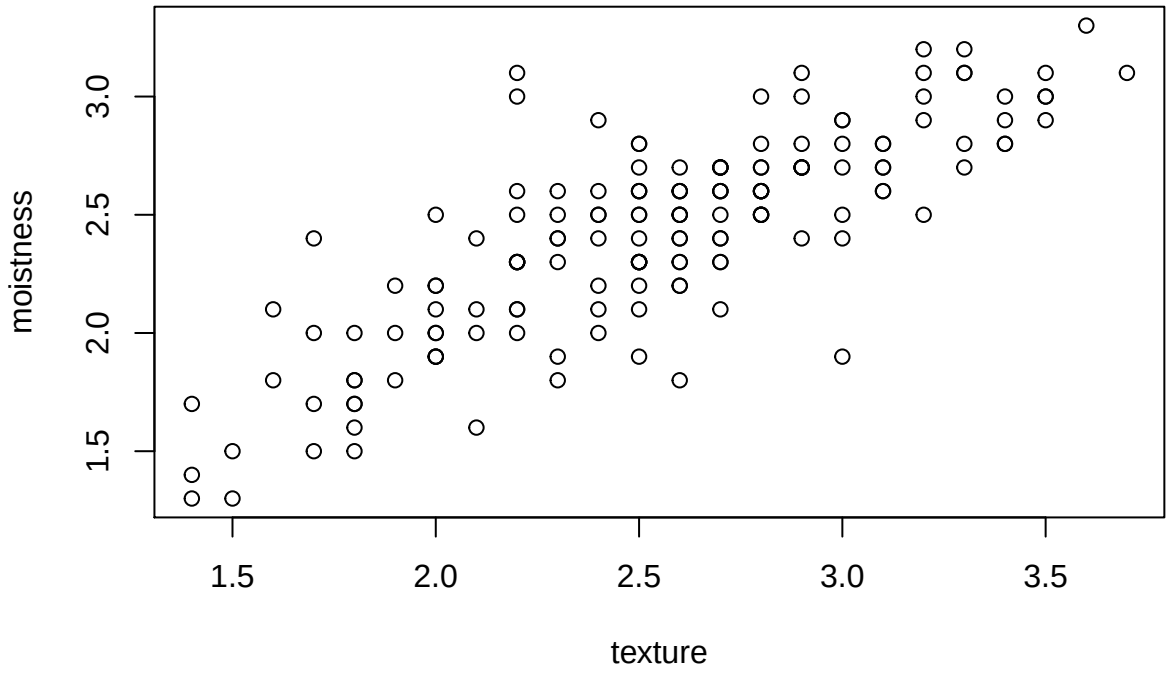
fread: more advanced use

fread() can also take additional arguments for deciding how our data should be imported. Additionally, we plot() the data in this exercise.

```
# fread is already loaded

# Import columns 6 and 8 of potatoes.csv: potatoes
potatoes <- fread("data/potatoes.csv", select = c(6, 8));

# Plot texture (x) and moistness (y) of potatoes
plot(potatoes$texture, potatoes$moistness, xlab="texture", ylab="moistness")
```



Dedicated classes

A multiple choice question regarding classes and fread(), the correct answer is: The class of the result of `fread()` is both data.table and data.frame. read_csv() creates an object with three classes: tbl_df, tbl and data.frame.

List the sheets of an Excel file

We learn the usage of excel_sheets(), provided from the readxl package, which makes it possible to read in excel sheets in a convenient way.

```
# Load the readxl package
library(readxl)

# Print the names of all worksheets
excel_sheets("data/urbanpop.xlsx")
```

```
## [1] "1960-1966" "1967-1974" "1975-2011"
```

Import an Excel sheet

We can name the sheets and combine them into a list.

```
# The readxl package is already loaded

# Read the sheets, one by one
pop_1 <- read_excel("data/urbanpop.xlsx", sheet = 1);
pop_2 <- read_excel("data/urbanpop.xlsx", sheet = 2);
pop_3 <- read_excel("data/urbanpop.xlsx", sheet = 3);

# Put pop_1, pop_2 and pop_3 in a list: pop_list
pop_list = list(pop_1, pop_2, pop_3);

# Display the structure of pop_list
str(pop_list)
```

```
## List of 3
##  $ : tibble [209 x 8] (S3: tbl_df/tbl/data.frame)
##   ..$ country: chr [1:209] "Afghanistan" "Albania" "Algeria" "American Samoa" ...
##   ..$ 1960   : num [1:209] 769308 494443 3293999 NA NA ...
##   ..$ 1961   : num [1:209] 814923 511803 3515148 13660 8724 ...
##   ..$ 1962   : num [1:209] 858522 529439 3739963 14166 9700 ...
##   ..$ 1963   : num [1:209] 903914 547377 3973289 14759 10748 ...
##   ..$ 1964   : num [1:209] 951226 565572 4220987 15396 11866 ...
##   ..$ 1965   : num [1:209] 1000582 583983 4488176 16045 13053 ...
##   ..$ 1966   : num [1:209] 1058743 602512 4649105 16693 14217 ...
##  $ : tibble [209 x 9] (S3: tbl_df/tbl/data.frame)
##   ..$ country: chr [1:209] "Afghanistan" "Albania" "Algeria" "American Samoa" ...
##   ..$ 1967   : num [1:209] 1119067 621180 4826104 17349 15440 ...
##   ..$ 1968   : num [1:209] 1182159 639964 5017299 17996 16727 ...
##   ..$ 1969   : num [1:209] 1248901 658853 5219332 18619 18088 ...
##   ..$ 1970   : num [1:209] 1319849 677839 5429743 19206 19529 ...
##   ..$ 1971   : num [1:209] 1409001 698932 5619042 19752 20929 ...
##   ..$ 1972   : num [1:209] 1502402 720207 5815734 20263 22406 ...
##   ..$ 1973   : num [1:209] 1598835 741681 6020647 20742 23937 ...
##   ..$ 1974   : num [1:209] 1696445 763385 6235114 21194 25482 ...
##  $ : tibble [209 x 38] (S3: tbl_df/tbl/data.frame)
##   ..$ country: chr [1:209] "Afghanistan" "Albania" "Algeria" "American Samoa" ...
##   ..$ 1975   : num [1:209] 1793266 785350 6460138 21632 27019 ...
##   ..$ 1976   : num [1:209] 1905033 807990 6774099 22047 28366 ...
##   ..$ 1977   : num [1:209] 2021308 830959 7102902 22452 29677 ...
##   ..$ 1978   : num [1:209] 2142248 854262 7447728 22899 31037 ...
##   ..$ 1979   : num [1:209] 2268015 877898 7810073 23457 32572 ...
##   ..$ 1980   : num [1:209] 2398775 901884 8190772 24177 34366 ...
##   ..$ 1981   : num [1:209] 2493265 927224 8637724 25173 36356 ...
##   ..$ 1982   : num [1:209] 2590846 952447 9105820 26342 38618 ...
##   ..$ 1983   : num [1:209] 2691612 978476 9591900 27655 40983 ...
##   ..$ 1984   : num [1:209] 2795656 1006613 10091289 29062 43207 ...
##   ..$ 1985   : num [1:209] 2903078 1037541 10600112 30524 45119 ...
##   ..$ 1986   : num [1:209] 3006983 1072365 11101757 32014 46254 ...
##   ..$ 1987   : num [1:209] 3113957 1109954 11609104 33548 47019 ...
##   ..$ 1988   : num [1:209] 3224082 1146633 12122941 35095 47669 ...
##   ..$ 1989   : num [1:209] 3337444 1177286 12645263 36618 48577 ...
##   ..$ 1990   : num [1:209] 3454129 1198293 13177079 38088 49982 ...
```

```
##   ..$ 1991   : num [1:209] 3617842 1215445 13708813 39600 51972 ...
##   ..$ 1992   : num [1:209] 3788685 1222544 14248297 41049 54469 ...
##   ..$ 1993   : num [1:209] 3966956 1222812 14789176 42443 57079 ...
##   ..$ 1994   : num [1:209] 4152960 1221364 15322651 43798 59243 ...
##   ..$ 1995   : num [1:209] 4347018 1222234 15842442 45129 60598 ...
##   ..$ 1996   : num [1:209] 4531285 1228760 16395553 46343 60927 ...
##   ..$ 1997   : num [1:209] 4722603 1238090 16935451 47527 60462 ...
##   ..$ 1998   : num [1:209] 4921227 1250366 17469200 48705 59685 ...
##   ..$ 1999   : num [1:209] 5127421 1265195 18007937 49906 59281 ...
##   ..$ 2000   : num [1:209] 5341456 1282223 18560597 51151 59719 ...
##   ..$ 2001   : num [1:209] 5564492 1315690 19198872 52341 61062 ...
##   ..$ 2002   : num [1:209] 5795940 1352278 19854835 53583 63212 ...
##   ..$ 2003   : num [1:209] 6036100 1391143 20529356 54864 65802 ...
##   ..$ 2004   : num [1:209] 6285281 1430918 21222198 56166 68301 ...
##   ..$ 2005   : num [1:209] 6543804 1470488 21932978 57474 70329 ...
##   ..$ 2006   : num [1:209] 6812538 1512255 22625052 58679 71726 ...
##   ..$ 2007   : num [1:209] 7091245 1553491 23335543 59894 72684 ...
##   ..$ 2008   : num [1:209] 7380272 1594351 24061749 61118 73335 ...
##   ..$ 2009   : num [1:209] 7679982 1635262 24799591 62357 73897 ...
##   ..$ 2010   : num [1:209] 7990746 1676545 25545622 63616 74525 ...
##   ..$ 2011   : num [1:209] 8316976 1716842 26216968 64817 75207 ...
```

Reading a workbook

lapply() allows us to read in and merge our excel sheets in a convenient way.

```
# The readxl package is already loaded

# Read all Excel sheets with lapply(): pop_list
pop_list <- lapply(excel_sheets("data/urbanpop.xlsx"),
                   read_excel, path = "data/urbanpop.xlsx");

# Display the structure of pop_list
str(pop_list)
```

```
## List of 3
##  $ : tibble [209 x 8] (S3: tbl_df/tbl/data.frame)
##   ..$ country: chr [1:209] "Afghanistan" "Albania" "Algeria" "American Samoa" ...
##   ..$ 1960   : num [1:209] 769308 494443 3293999 NA NA ...
##   ..$ 1961   : num [1:209] 814923 511803 3515148 13660 8724 ...
##   ..$ 1962   : num [1:209] 858522 529439 3739963 14166 9700 ...
##   ..$ 1963   : num [1:209] 903914 547377 3973289 14759 10748 ...
##   ..$ 1964   : num [1:209] 951226 565572 4220987 15396 11866 ...
##   ..$ 1965   : num [1:209] 1000582 583983 4488176 16045 13053 ...
##   ..$ 1966   : num [1:209] 1058743 602512 4649105 16693 14217 ...
##  $ : tibble [209 x 9] (S3: tbl_df/tbl/data.frame)
##   ..$ country: chr [1:209] "Afghanistan" "Albania" "Algeria" "American Samoa" ...
##   ..$ 1967   : num [1:209] 1119067 621180 4826104 17349 15440 ...
##   ..$ 1968   : num [1:209] 1182159 639964 5017299 17996 16727 ...
##   ..$ 1969   : num [1:209] 1248901 658853 5219332 18619 18088 ...
##   ..$ 1970   : num [1:209] 1319849 677839 5429743 19206 19529 ...
##   ..$ 1971   : num [1:209] 1409001 698932 5619042 19752 20929 ...
##   ..$ 1972   : num [1:209] 1502402 720207 5815734 20263 22406 ...
##   ..$ 1973   : num [1:209] 1598835 741681 6020647 20742 23937 ...
##   ..$ 1974   : num [1:209] 1696445 763385 6235114 21194 25482 ...
```

```
##  $ : tibble [209 x 38] (S3: tbl_df/tbl/data.frame)
##   ..$ country: chr [1:209] "Afghanistan" "Albania" "Algeria" "American Samoa" ...
##   ..$ 1975   : num [1:209] 1793266 785350 6460138 21632 27019 ...
##   ..$ 1976   : num [1:209] 1905033 807990 6774099 22047 28366 ...
##   ..$ 1977   : num [1:209] 2021308 830959 7102902 22452 29677 ...
##   ..$ 1978   : num [1:209] 2142248 854262 7447728 22899 31037 ...
##   ..$ 1979   : num [1:209] 2268015 877898 7810073 23457 32572 ...
##   ..$ 1980   : num [1:209] 2398775 901884 8190772 24177 34366 ...
##   ..$ 1981   : num [1:209] 2493265 927224 8637724 25173 36356 ...
##   ..$ 1982   : num [1:209] 2590846 952447 9105820 26342 38618 ...
##   ..$ 1983   : num [1:209] 2691612 978476 9591900 27655 40983 ...
##   ..$ 1984   : num [1:209] 2795656 1006613 10091289 29062 43207 ...
##   ..$ 1985   : num [1:209] 2903078 1037541 10600112 30524 45119 ...
##   ..$ 1986   : num [1:209] 3006983 1072365 11101757 32014 46254 ...
##   ..$ 1987   : num [1:209] 3113957 1109954 11609104 33548 47019 ...
##   ..$ 1988   : num [1:209] 3224082 1146633 12122941 35095 47669 ...
##   ..$ 1989   : num [1:209] 3337444 1177286 12645263 36618 48577 ...
##   ..$ 1990   : num [1:209] 3454129 1198293 13177079 38088 49982 ...
##   ..$ 1991   : num [1:209] 3617842 1215445 13708813 39600 51972 ...
##   ..$ 1992   : num [1:209] 3788685 1222544 14248297 41049 54469 ...
##   ..$ 1993   : num [1:209] 3966956 1222812 14789176 42443 57079 ...
##   ..$ 1994   : num [1:209] 4152960 1221364 15322651 43798 59243 ...
##   ..$ 1995   : num [1:209] 4347018 1222234 15842442 45129 60598 ...
##   ..$ 1996   : num [1:209] 4531285 1228760 16395553 46343 60927 ...
##   ..$ 1997   : num [1:209] 4722603 1238090 16935451 47527 60462 ...
##   ..$ 1998   : num [1:209] 4921227 1250366 17469200 48705 59685 ...
##   ..$ 1999   : num [1:209] 5127421 1265195 18007937 49906 59281 ...
##   ..$ 2000   : num [1:209] 5341456 1282223 18560597 51151 59719 ...
##   ..$ 2001   : num [1:209] 5564492 1315690 19198872 52341 61062 ...
##   ..$ 2002   : num [1:209] 5795940 1352278 19854835 53583 63212 ...
##   ..$ 2003   : num [1:209] 6036100 1391143 20529356 54864 65802 ...
##   ..$ 2004   : num [1:209] 6285281 1430918 21222198 56166 68301 ...
##   ..$ 2005   : num [1:209] 6543804 1470488 21932978 57474 70329 ...
##   ..$ 2006   : num [1:209] 6812538 1512255 22625052 58679 71726 ...
##   ..$ 2007   : num [1:209] 7091245 1553491 23335543 59894 72684 ...
##   ..$ 2008   : num [1:209] 7380272 1594351 24061749 61118 73335 ...
##   ..$ 2009   : num [1:209] 7679982 1635262 24799591 62357 73897 ...
##   ..$ 2010   : num [1:209] 7990746 1676545 25545622 63616 74525 ...
##   ..$ 2011   : num [1:209] 8316976 1716842 26216968 64817 75207 ...
```

The col_names argument

This argument allows us to specify the header and the column names for the dataframe. If set to FALSE, R will choose the names, if set to TRUE, the first row in the excel sheet will be used for naming.

```
# The readxl package is already loaded

# Import the first Excel sheet of urbanpop_nonames.xlsx (R gives names): pop_a
pop_a <- read_excel("data/urbanpop_nonames.xlsx", col_names = FALSE)
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
```

```
## * `` -> ...5
## * ...
```

```
# Import the first Excel sheet of urbanpop_nonames.xlsx (specify col_names): pop_b
cols <- c("country", paste0("year_", 1960:1966))
pop_b <- read_excel("data/urbanpop_nonames.xlsx", col_names = cols)

# Print the summary of pop_a
summary(pop_a)
```

```
##      ...1                ...2                ...3                ...4
##  Length:209         Min.   :     3378   Min.   :     1028   Min.   :     1090
##  Class :character   1st Qu.:    88978   1st Qu.:    70644   1st Qu.:    74974
##  Mode  :character   Median :   580675   Median :   570159   Median :   593968
##                     Mean   :  4988124   Mean   :  4991613   Mean   :  5141592
##                     3rd Qu.:  3077228   3rd Qu.:  2807280   3rd Qu.:  2948396
##                     Max.   :126469700   Max.   :129268133   Max.   :131974143
##                     NA's   :11
##       ...5                ...6                ...7
##  Min.   :     1154   Min.   :     1218   Min.   :     1281
##  1st Qu.:    81870   1st Qu.:    84953   1st Qu.:    88633
##  Median :   619331   Median :   645262   Median :   679109
##  Mean   :  5303711   Mean   :  5468966   Mean   :  5637394
##  3rd Qu.:  3148941   3rd Qu.:  3296444   3rd Qu.:  3317422
##  Max.   :134599886   Max.   :137205240   Max.   :139663053
##
##       ...8
##  Min.   :     1349
##  1st Qu.:    93638
##  Median :   735139
##  Mean   :  5790281
##  3rd Qu.:  3418036
##  Max.   :141962708
##
```

```
# Print the summary of pop_b
summary(pop_b)
```

```
##    country            year_1960           year_1961           year_1962
##  Length:209         Min.   :     3378   Min.   :     1028   Min.   :     1090
##  Class :character   1st Qu.:    88978   1st Qu.:    70644   1st Qu.:    74974
##  Mode  :character   Median :   580675   Median :   570159   Median :   593968
##                     Mean   :  4988124   Mean   :  4991613   Mean   :  5141592
##                     3rd Qu.:  3077228   3rd Qu.:  2807280   3rd Qu.:  2948396
##                     Max.   :126469700   Max.   :129268133   Max.   :131974143
##                     NA's   :11
##    year_1963           year_1964           year_1965
##  Min.   :     1154   Min.   :     1218   Min.   :     1281
##  1st Qu.:    81870   1st Qu.:    84953   1st Qu.:    88633
##  Median :   619331   Median :   645262   Median :   679109
##  Mean   :  5303711   Mean   :  5468966   Mean   :  5637394
##  3rd Qu.:  3148941   3rd Qu.:  3296444   3rd Qu.:  3317422
##  Max.   :134599886   Max.   :137205240   Max.   :139663053
##
##    year_1966
```

```
##  Min.   :     1349
##  1st Qu.:    93638
##  Median :   735139
##  Mean   :  5790281
##  3rd Qu.:  3418036
##  Max.   :141962708
##
```

The skip argument

This argument allows us to specify if we want to exclude a certain amount of entries in our underlying data.

```
# The readxl package is already loaded

# Import the second sheet of urbanpop.xlsx, skipping the first 21 rows: urbanpop_sel
urbanpop_sel = read_excel("data/urbanpop.xlsx",
                          col_names = FALSE,
                          skip = 21, sheet = 2)
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
## * ...
```

```
# Print out the first observation from urbanpop_sel
head(urbanpop_sel, n = 1)
```

```
## # A tibble: 1 x 9
##   ...1     ...2    ...3    ...4    ...5    ...6    ...7    ...8    ...9
##   <chr>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Benin 382022. 411859. 443013. 475611. 515820. 557938. 602093. 648410.
```

Import a local file

The gdata package provides its own function to read in excel sheets. It takes arguments to specify the name of the specific sheet.

```
# Load the gdata package
library(gdata)
```

```
## gdata: read.xls support for 'XLS' (Excel 97-2004) files ENABLED.

##

## gdata: read.xls support for 'XLSX' (Excel 2007+) files ENABLED.

##
## Attaching package: 'gdata'

## The following objects are masked from 'package:data.table':
##
##     first, last

## The following object is masked from 'package:stats':
##
##     nobs
```

```
## The following object is masked from 'package:utils':
##
##     object.size
```

```
## The following object is masked from 'package:base':
##
##     startsWith
```

```
# Import the second sheet of urbanpop.xls: urban_pop
urban_pop <- read.xls("data/urbanpop.xls", sheet = "1967-1974")

# Print the first 11 observations using head()
head(urban_pop, n = 11)
```

```
##                country       X1967       X1968       X1969       X1970
## 1          Afghanistan  1119067.20  1182159.06  1248900.79  1319848.78
## 2              Albania   621179.85   639964.46   658853.12   677839.12
## 3              Algeria  4826104.22  5017298.60  5219331.87  5429743.08
## 4       American Samoa    17348.66    17995.51    18618.68    19206.39
## 5              Andorra    15439.62    16726.99    18088.32    19528.96
## 6               Angola   757496.32   798459.26   841261.96   886401.63
## 7  Antigua and Barbuda    22086.25    22149.39    22182.92    22180.87
## 8            Argentina 17753280.98 18124103.64 18510462.30 18918072.79
## 9              Armenia  1337032.09  1392892.13  1449641.49  1507619.77
## 10               Aruba    29414.72    29576.09    29737.87    29901.57
## 11           Australia  9934404.03 10153969.77 10412390.67 10664093.55
##          X1971       X1972       X1973       X1974
## 1   1409001.09  1502401.79  1598835.45  1696444.83
## 2    698932.25   720206.57   741681.04   763385.45
## 3   5619041.53  5815734.49  6020647.35  6235114.38
## 4     19752.02    20262.67    20741.97    21194.38
## 5     20928.73    22405.84    23937.05    25481.98
## 6    955010.09  1027397.35  1103829.78  1184486.23
## 7     22560.87    22907.76    23221.29    23502.92
## 8  19329718.16 19763078.00 20211424.85 20664728.90
## 9   1564367.60  1622103.53  1680497.75  1739063.02
## 10    30081.36    30279.76    30467.42    30602.87
## 11 11047706.39 11269945.50 11461120.68 11772934.25
```

read.xls() wraps around read.table()

Here we learn that we can specify arguments for read.xls() in the same way as read.table(), e.g. sheet, skip, header, stringsAsFactors, col.names.

```
# The gdata package is alreaded loaded

# Column names for urban_pop
columns <- c("country", paste0("year_", 1967:1974))

# Finish the read.xls call
urban_pop <- read.xls("data/urbanpop.xls", sheet = 2,
                      skip = 50, header = FALSE, stringsAsFactors = FALSE,
                      col.names = columns)

# Print first 10 observation of urban_pop
head(urban_pop, n = 10)
```

```
##                country   year_1967   year_1968   year_1969   year_1970
## 1               Cyprus   231929.74   237831.38   243983.34   250164.52
## 2       Czech Republic  6204409.91  6266304.50  6326368.97  6348794.89
## 3              Denmark  3777552.62  3826785.08  3874313.99  3930042.97
## 4             Djibouti    77788.04    84694.35    92045.77    99845.22
## 5             Dominica    27550.36    29527.32    31475.62    33328.25
## 6   Dominican Republic  1535485.43  1625455.76  1718315.40  1814060.00
## 7              Ecuador  2059355.12  2151395.14  2246890.79  2345864.41
## 8                Egypt 13798171.00 14248342.19 14703858.22 15162858.52
## 9          El Salvador  1345528.98  1387218.33  1429378.98  1472181.26
## 10  Equatorial Guinea    75364.50    77295.03    78445.74    78411.07
##       year_1971   year_1972   year_1973   year_1974
## 1    261213.21   272407.99   283774.90   295379.83
## 2   6437055.17  6572632.32  6718465.53  6873458.18
## 3   3981360.12  4028247.92  4076867.28  4120201.43
## 4    107799.69   116098.23   125391.58   136606.25
## 5     34761.52    36049.99    37260.05    38501.47
## 6   1915590.38  2020157.01  2127714.45  2238203.87
## 7   2453817.78  2565644.81  2681525.25  2801692.62
## 8  15603661.36 16047814.69 16498633.27 16960827.93
## 9   1527985.34  1584758.18  1642098.95  1699470.87
## 10    77055.29    74596.06    71438.96    68179.26
```

Work that Excel data!

cbind() can be used to combine multiple sheets, na.omit() allows us to clean the data, by removing NA values.

```
# Add code to import data from all three sheets in urbanpop.xls
path <- "data/urbanpop.xls"
urban_sheet1 <- read.xls(path, sheet = 1, stringsAsFactors = FALSE)
urban_sheet2 <- read.xls(path, sheet = 2, stringsAsFactors = FALSE)
urban_sheet3 <- read.xls(path, sheet = 3, stringsAsFactors = FALSE)


# Extend the cbind() call to include urban_sheet3: urban
urban <- cbind(urban_sheet1, urban_sheet2[-1], urban_sheet3[-1])

# Remove all rows with NAs from urban: urban_clean
urban_clean <- na.omit(urban)
```

Connect to a workbook

The package XLConnect allows us to use functions that can again be used to work with Excel worksheets. For the initial connection, it provides us with the loadWorkbook() function.

```
# urbanpop.xlsx is available in your working directory

# Load the XLConnect package
library(XLConnect)
```

```
## XLConnect 1.0.2 by Mirai Solutions GmbH [aut],
##   Martin Studer [cre],
##   The Apache Software Foundation [ctb, cph] (Apache POI),
##   Graph Builder [ctb, cph] (Curvesapi Java library)
```

```
## https://mirai-solutions.ch
```

```
## https://github.com/miraisolutions/xlconnect
```

```
# Build connection to urbanpop.xlsx: my_book
my_book <- loadWorkbook("data/urbanpop.xlsx")

# Print out the class of my_book
class(my_book)
```

```
## [1] "workbook"
## attr(,"package")
## [1] "XLConnect"
```

List and read excel sheets

getSheets() and readWorksheets() from the XLConnect package can be used to list and load sheets, respectively.

```
# XLConnect is already available

# Build connection to urbanpop.xlsx
my_book <- loadWorkbook("data/urbanpop.xlsx")

# List the sheets in my_book
getSheets(my_book)
```

```
## [1] "1960-1966" "1967-1974" "1975-2011"
```

```
# Import the second sheet in my_book
sheet2 = readWorksheet(my_book, sheet = 2)
```

Customize readWorksheet

We can use startCol and endCol to decide which columns we want to import and cbind() to combine our selection together.

```
# XLConnect is already available

# Build connection to urbanpop.xlsx
my_book <- loadWorkbook("data/urbanpop.xlsx")

# Import columns 3, 4, and 5 from second sheet in my_book: urbanpop_sel
urbanpop_sel <- readWorksheet(my_book, sheet = 2, startCol = 3, endCol = 5)

# Import first column from second sheet in my_book: countries
countries <- readWorksheet(my_book, sheet = 2, startCol = 1, endCol = 1)

# cbind() urbanpop_sel and countries together: selection
selection <- cbind(countries, urbanpop_sel)
```

Add worksheet

Unlike the other approaches used so far, the connection to an actual workbook provided by XLConnect makes it possible to manipulate excel sheets from inside R.

```
# XLConnect is already available

# Build connection to urbanpop.xlsx
my_book <- loadWorkbook("data/urbanpop.xlsx")
```

```
# Add a worksheet to my_book, named "data_summary"
createSheet(my_book, "data_summary")

# Use getSheets() on my_book
getSheets(my_book)
```

```
## [1] "1960-1966"    "1967-1974"    "1975-2011"    "data_summary"
```

Populate worksheet

As described, we get an actual connection to a workbook from XLConnect. We can use this to populate our workbook from R with writeWorksheet() and save it to the worksheet with saveWorksheet().

```
# XLConnect is already available

# Build connection to urbanpop.xlsx
my_book <- loadWorkbook("data/urbanpop.xlsx")

# Add a worksheet to my_book, named "data_summary"
createSheet(my_book, "data_summary")

# Create data frame: summ
sheets <- getSheets(my_book)[1:3]
dims <- sapply(sheets, function(x) dim(readWorksheet(my_book, sheet = x)), USE.NAMES = FALSE)
summ <- data.frame(sheets = sheets,
                   nrows = dims[1, ],
                   ncols = dims[2, ])

# Add data in summ to "data_summary" sheet
writeWorksheet(my_book, summ, sheet = "data_summary")
# Save workbook as summary.xlsx
saveWorkbook(my_book, "data/summary.xlsx")
```

Renaming sheets

In the same manner as above, renameSheet() can be used to rename a worksheet.

```
# my_book is available

# Rename "data_summary" sheet to "summary"
renameSheet(my_book, sheet = 4, "summary")

# Print out sheets of my_book
getSheets(my_book)
```

```
## [1] "1960-1966" "1967-1974" "1975-2011" "summary"
```

```
# Save workbook to "renamed.xlsx"
saveWorkbook(my_book, "data/renamed.xlsx")
```

Removing sheets

In the same manner as above, renameSheet() can be used to remove a worksheet.

```
# Load the XLConnect package
library(XLConnect)
```

```
# Build connection to renamed.xlsx: my_book
my_book <- loadWorkbook("data/renamed.xlsx")

# Remove the fourth sheet
removeSheet(my_book, "summary")

# Save workbook to "clean.xlsx"
saveWorkbook(my_book, "data/clean.xlsx")
```