Data import and export, connection to SQL databases

Didzis Elferts

Interreg
Estonia-Latvia
European Regional Development Fund

EUROPEAN UNION

WaterAct
Joint actions for more efficient management of common groundwater resources

Data import

- Base R provides functions to import .txt, .csv and other ASCI type data
- With additional libraries it is possible to import other formats - Excel, SPSS, SAS, json, shp, etc.

Working directory

If you do not specify the full path to this file when you import a data file, R searches for it in the working directory.

```
getwd()
```

```
## [1] "/Users/didzis/Dropbox/DidzaDati/2021/R_kursi/Presentations"
```

Working directory can be changed with function **`setwd()`** or in RStudio selecting **`Session/Set working directory`**

RStudio project

Way to divide your work in independent parts with its own:

- working directory
- workspace
- history
- source documents

Data import

You can also use package **`readr`** to import data files (tables). This package contains functions that allow you to import different types of files, depending on the type of column separator. The benefits are much higher speeds, the ability to properly recognize the data format in most cases, without automatically creating factor columns.

- **`read_csv()`**: comma separated files (CSV)
- **`read_tsv()`**: tab delimited
- **`read_delim()`**: general function
- **`read_fwf()`**: fixed width files
- **`read_table()`**: white space separated
- **`read_log()`**: web log files

Excel import

You can use package **`readxl`** function **`read_excel ()`** to import Excel files. In the function, you can specify both the page to be imported and import only part tables. The function is automatically able to determine column types (including dates).

```
library(readxl)
df <- read_excel("Data/Didzis.xlsx")
df
```

```
## # A tibble: 6 × 2
##   Height Weight
##    <dbl>  <dbl>
## 1    175     70
## 2    180     86
## 3    165     61
## 4    163     69
## 5    172     72
## 6    169     68
```

SPSS file import

With R package `foreign` you can import Minitab, S, SAS, SPSS, Stata, Systat, Weka data files (but there could be some problems due to different versions of files).

Example shows import of SPSS data file (source of file: http://spss.allenandunwin.com.s3-website-ap-southeast-2.amazonaws.com/data-files.html#.WtRSetYuDIE)

SPSS file import

```
## re-encoding from CP1252
```

```
library(foreign)
survey <- read.spss("Data/survey.sav", to.data.frame = TRUE)
```

```
str(survey)
```

```
## 'data.frame':    439 obs. of  134 variables:
##  $ id        : num  415 9 425 307 440 484 341 300 61 24 ...
##  $ sex       : Factor w/ 2 levels "MALES","FEMALES": 2 1 2 1 1 2 2 1 2 1 ...
##  $ age       : num  24 39 48 41 23 31 30 23 18 23 ...
##  $ marital   : Factor w/ 8 levels "SINGLE","STEADY RELATIONSHIP",..: 4 3 4 5 1 4 6 2 2 1 ...
##  $ child     : Factor w/ 2 levels "YES","NO": 1 1 1 1 2 1 2 2 2 2 ...
##  $ educ      : Factor w/ 6 levels "PRIMARY","SOME SECONDARY",..: 5 5 2 2 5 5 4 5 2 6 ...
##  $ source    : Factor w/ 9 levels "WORK","SPOUSE OR PARTNER",..: 7 1 4 1 1 7 8 1 2 NA ...
##  $ smoke     : Factor w/ 2 levels "YES","NO": 2 1 2 2 2 2 2 1 1 2 ...
##  $ smokenum  : num  NA 2 NA 0 0 NA 0 100 40 0 ...
##  $ op1       : num  3 2 3 3 3 2 3 4 3 1 ...
##  $ op2       : num  2 3 1 1 2 2 5 1 4 3 ...
##  $ op3       : num  3 4 3 5 3 2 1 3 2 1 ...
##  $ op4       : num  2 3 3 3 2 2 4 1 4 4 ...
##  $ op5       : num  4 5 3 5 1 3 1 4 2 1 ...
##  $ op6       : num  2 4 4 1 3 4 4 2 4 5 ...
##  $ mast1     : num  2 2 3 2 1 1 4 2 3 3 ...
##  $ mast2     : num  4 4 3 4 4 3 2 4 2 1 ...
##  $ mast3     : num  2 2 2 1 2 2 4 1 3 3 ...
##  $ mast4     : num  2 3 3 1 1 2 4 2 3 3 ...
##  $ mast5     : num  4 4 3 4 2 3 2 3 2 2 ...
```

SPSS file import

Alternative to **foreign** is package **haven** that allows to import SAS, SPSS and Stata files, showing data in different way and does not make factors by default.

```
library(haven)
survey2 <- read_spss("Data/survey.sav")
```

```
str(survey2)
```

```
## tibble [439 × 134] (S3: tbl_df/tbl/data.frame)
##  $ id        : num [1:439] 415 9 425 307 440 484 341 300 61 24 ...
##   ..- attr(*, "format.spss")= chr "F3.0"
##  $ sex       : dbl+lbl [1:439] 2, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1, 2, 2, ...
##    ..@ label      : chr "sex"
##    ..@ format.spss: chr "F3.0"
##    ..@ labels     : Named num [1:2] 1 2
##    .. ..- attr(*, "names")= chr [1:2] "MALES" "FEMALES"
##  $ age       : num [1:439] 24 39 48 41 23 31 30 23 18 23 ...
##   ..- attr(*, "format.spss")= chr "F3.0"
##  $ marital   : dbl+lbl [1:439] 4, 3, 4, 5, 1, 4, 6, 2, 2, 1, 1, 4, 1, 4, 4, 4, 6, 7, ...
##    ..@ label      : chr "marital status"
##    ..@ format.spss: chr "F8.0"
##    ..@ labels     : Named num [1:8] 1 2 3 4 5 6 7 8
##    .. ..- attr(*, "names")= chr [1:8] "SINGLE" "STEADY RELATIONSHIP" "LIVING WITH PARTNER" "MARRIED FIRST TIME" ...
##  $ child     : dbl+lbl [1:439] 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2, 1, 1, ...
##    ..@ label      : chr "child"
##    ..@ format.spss: chr "F5.0"
##    ..@ labels     : Named num [1:2] 1 2
```

Factor

- Special type of data enabling the production of nominal and ordinal data for statistical analyses and graphics
- By default, a sequence of factor levels is created alphabetically
- You cannot simply add other data to a factor if it is not between the defined levels

Factor

```
vec_1 <- c("A", "B", "A", "C", "D")
vec_1
```

```
## [1] "A" "B" "A" "C" "D"
```

```
fact_1 <- factor(vec_1)
fact_1
```

```
## [1] A B A C D
## Levels: A B C D
```

Factor

```
vec_1[6] <- "E"
vec_1
```

```
## [1] "A" "B" "A" "C" "D" "E"
```

```
fact_1[6] <- "E"
```

```
## Warning in `[<-.factor`(`*tmp*`, 6, value = "E"): invalid factor level, NA
## generated
```

```
fact_1
```

```
## [1] A    B    A    C    D    <NA>
## Levels: A B C D
```

Factor

With function **`levels()`** you can add new levels to the factor

```
levels(fact_1)[5] <- "E"
levels(fact_1)
```

```
## [1] "A" "B" "C" "D" "E"
```

```
fact_1[6] <- "E"
fact_1
```

```
## [1] A B A C D E
## Levels: A B C D E
```

Factor

With argument **`levels =`** of function **`factor()`**, you can change levels of the factor

```
fact_1 <- factor(fact_1, levels = c("D","B","E", "A","C"))
fact_1
```

```
## [1] A B A C D E
## Levels: D B E A C
```

Connection to database

With an R program, you can connect to different databases. For example, a connection to SQLite database is made using an R package **DBI**.

Initially make connection to the database

```
library(DBI)
mydb <- dbConnect(RSQLite::SQLite(), "Data/db.sqlite")
```

Connection to database

Functions **`dblistTables()`** and **`dbListFields()`** respectivly show tables in database and variables in the table.

```
dbListTables(mydb)
```

```
## [1] "iris"   "mtcars"
```

```
dbListFields(mydb, "mtcars")
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

Connection to database

Whith function **`dbReadTable()`** whole table is imported

```
db_tabula <- dbReadTable(mydb, "mtcars")
head(db_tabula)
```

```
##    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## 4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## 5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## 6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Connection to database

Function **`dbGetQuery()`** allows to make data queries using SQL language

```
db_tabula <- dbGetQuery(mydb, 'SELECT * FROM mtcars WHERE mpg > 20')
head(db_tabula)
```

```
##    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## 2 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## 3 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 4 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## 5 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## 6 22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
```

Connection to database

When data import is finished, connection to the database should be closed.

```
dbDisconnect(mydb)
```

Data export

Data

As an example data from R object **`mtcars`** is used.

```
data("mtcars")
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

readr package

Package `readr` contains functions also for the data export

- `write_delim()`
- `write_csv()` - "," as delimiter
- `write_csv2()` - ";" as delimiter
- `write_excel_csv()`
- `write_tsv()`

readr package

```
library(readr)
write_csv(mtcars, "mtcars.csv")
write_csv2(mtcars, "mtcars2.csv")
write_tsv(mtcars, "mtcars.tsv")
```

xlsx package

One of the R packages to export Excel files is **`xlsx`**. This package depends on JAVA installation

```
library(xlsx)
```

Table export

Main function for export tables is **`write.xlsx()`**. As arguments table name and file name are added (Excel file is made if not present)

```
write.xlsx(mtcars, "New_file.xlsx")
```

Adding arguments **`append = TRUE`** and **`sheet =`** allows to add table to existing file

```
write.xlsx(mtcars, "New_file.xlsx", append = TRUE, sheet = "Duplicate")
```

Placing table in particular place

It is possible to place exported table in particular row and column

```
wb = createWorkbook()

sheet = createSheet(wb, "Sheet 1")

addDataFrame(mtcars, sheet=sheet, startColumn=1)
addDataFrame(mtcars, sheet=sheet, startColumn=15, startRow = 7)

saveWorkbook(wb, "My_File.xlsx")
```

Placing table in particular place

It is possible to load existing Excel file and add additional table to it

```
wb = loadWorkbook("My_File.xlsx")

sheet <- getSheets(wb)
addDataFrame(mtcars, sheet=sheet$`Sheet 1`, startColumn=30)

saveWorkbook(wb, "My_File.xlsx")
```

Example with formating

Example from https://github.com/colearendt/xlsx/

```
wb <- createWorkbook()
s <- createSheet(wb,'test')

cs <- CellStyle(wb) +
  Font(wb,heightInPoints = 16, isBold = TRUE) +
  Alignment(horizontal='ALIGN_CENTER')


r <- createRow(s,1)
cell <- createCell(r,1:ncol(mtcars))

setCellValue(cell[[1]],'Title for mtcars')
for (i in cell) {
  setCellStyle(i,cs)
}

addMergedRegion(s, 1,1, 1,ncol(mtcars))

addDataFrame(mtcars, s, row.names=FALSE, startRow=3)

saveWorkbook(wb,'mtcars_pretty.xlsx')
```

Data selections

Selecting one variable

You can use `$` operator to select variable from the dataframe. First comes name of table and then name of variable. Square brackets also can be used.

```
df$Weight
```

```
## [1] 70 86 61 69 72 68
```

```
df["Weight"]
```

```
## # A tibble: 6 × 1
##   Weight
##    <dbl>
## 1     70
## 2     86
## 3     61
## 4     69
## 5     72
## 6     68
```

Selecting rows

Rows can be selected with square brackets after dataframe name.

```
df[1,]
```

```
## # A tibble: 1 × 2
##   Height Weight
##    <dbl>  <dbl>
## 1    175     70
```

```
df[c(2,6),]
```

```
## # A tibble: 2 × 2
##   Height Weight
##    <dbl>  <dbl>
## 1    180     86
## 2    169     68
```

```
df[1:3,]
```

```
## # A tibble: 3 × 2
##   Height Weight
##    <dbl>  <dbl>
## 1    175     70
## 2    180     86
## 3    165     61
```

Typical problems

Resources

- Common R Programming Errors Faced by Beginners - https://www.r-bloggers.com/2016/06/common-r-programming-errors-faced-by-beginners/
- Common R Error Messages - https://www.programmingr.com/r-error-messages/
- R Error Message Cheat Sheet - http://varianceexplained.org/courses/errors/
- Handling Errors & Warnings in R - https://statisticsglobe.com/errors-warnings-r

Floating poinr

```
sqrt(2) ^ 2 == 2
```

```
## [1] FALSE
```

```
1 / 49 * 49 == 1
```

```
## [1] FALSE
```

Floating point

As many programs also R has floating point problem. Solution is to use function **`near()`** from package **`dplyr`**.

```
library(dplyr)
near(sqrt(2) ^ 2,  2)
```

```
## [1] TRUE
```

```
near(1 / 49 * 49, 1)
```

```
## [1] TRUE
```

Could not find function

```
blom(rnorm(10))
```

```
## Error in blom(rnorm(10)): could not find function "blom"
```

Causes:

1. error in function name
2. packages is not added to the session

```
library(rcompanion)
blom(rnorm(10))
```

```
##  [1]  0.6580521  1.5593719 -0.3768262 -0.6580521  0.1230078  1.0053953
##  [7] -0.1230078  0.3768262 -1.5593719 -1.0053953
```

Error in x$x : $ operator is invalid for atomic vectors

```
x <- c(1, 2, 3)
x$x
```

```
## Error in x$x: $ operator is invalid for atomic vectors
```

Operator **$** used for the wrong object type

```
x[1]
```

```
## [1] 1
```

Error in x$V1 : $ operator is invalid for atomic vectors

```
y <- matrix(1:15, ncol = 3)
colnames(y) <- c("V1", "V2", "V3")
y$V1
```

```
## Error in y$V1: $ operator is invalid for atomic vectors
```

```
y[,1]
```

```
## [1] 1 2 3 4 5
```

Matrix also is as vector, so do not use **$** operator to select data

subscript out of bounds

```
matr <- matrix(1:20, ncol = 4)
matr
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

```
matr[,5]
```

```
## Error in matr[, 5]: subscript out of bounds
```

As matrix has only four columns, selecting fifth gives error

object of type 'closure' is not subsettable

```
rnorm[10]
```

```
## Error in rnorm[10]: object of type 'closure' is not subsettable
```

Square brackets used instead of parentheses.

```
rnorm(10)
```

```
##  [1] -1.04652770 -1.90273929 -0.80922454  0.42378619  0.14533866  0.06281033
##  [7] -1.51347022  0.60313983  0.44994704  0.77061790
```

Error in file(file, "rt") : cannot open the connection

```
dat <- read.csv("tb.csv")
```

```
## Warning in file(file, "rt"): cannot open file 'tb.csv': No such file or
## directory
```

```
## Error in file(file, "rt"): cannot open the connection
```

Causes:

1. wrong file name
2. wrong working directory or file missing
3. wrong path to the file

```
dat <- read.csv("Data/tb.csv")
```

Error in read.table(... object not found

```
dat <- read.csv(Data/tb.csv)
```

```
## Error in read.table(file = file, header = header, sep = sep, quote = quote, : object 'Data'
```

File name should be in quotes. Now functions searches for the object

Error in `[.data.frame`(mtcars, "Mpg") : undefined columns selected

```
data("mtcars")
mtcars["Mpg"]
```

```
## Error in `[.data.frame`(mtcars, "Mpg"): undefined columns selected
```

Wrong column selected (or mistake in the name)

```
mtcars["mpg"]
```

```
##                    mpg
## Mazda RX4         21.0
## Mazda RX4 Wag     21.0
## Datsun 710        22.8
## Hornet 4 Drive    21.4
## Hornet Sportabout 18.7
## Valiant           18.1
## Duster 360        14.3
## Merc 240D         24.4
## Merc 230          22.8
```

Error in plot.new() : figure margins too large

```
par(mar=c(0, 0, 10, 100))
plot(1:10)
```

```
## Error in plot.new(): figure margins too large
```

Causes

1. margins of the plot are too large
2. window for the plots in RStudio is too small

R error: vector memory exhausted (limit reached?)

```
## R error: vector memory exhausted (limit reached?)
## Error: cannot allocate vector of size 29.8 Gb
```

Size of the object is larger then RAM of computer or space allocated for R.

Error: unexpected ')' in

```
plot(1:10))
```

```
## Error: <text>:1:11: unexpected ')'
## 1: plot(1:10))
##               ^
```

Extra parentheses at the end of code (similar situation wit "}" and "]")

Error: unexpected symbol in

```
x = 1:10
y = 11:20
plot(x y)
```

```
## Error: <text>:3:8: unexpected symbol
## 2: y = 11:20
## 3: plot(x y
##           ^
```

Missing comma (or other symbol) between variables

```
x = 1:10
y = 11:20
plot(x, y)
```

Error: unexpected numeric constant in

```
plot(1:10 11:20)
```

```
## Error: <text>:1:11: unexpected numeric constant
## 1: plot(1:10 11
##               ^
```

Missing comma (or other symbol) between variables

```
plot(1:10, 11:20)
```

argument is not numeric or logical: returning NA

```
df <- data.frame(x = 1:10, y = 11: 20)
mean(df)
```

```
## Warning in mean.default(df): argument is not numeric or logical: returning NA
```

```
## [1] NA
```

Function used for wrong object type

```
apply(df, 2, mean)
```

```
##    x    y
##  5.5 15.5
```

Number of items to replace is not a multiple of replacement length

```
x <- 1:10
x[2:3] <- c(4,7,11)
```

```
## Warning in x[2:3] <- c(4, 7, 11): number of items to replace is not a multiple
## of replacement length
```

Number of items to replaace is smaller than number of provided values

```
x[2:3] <- c(4,7)
```

NAs introduced by coercion

```
vec <- c("50", "200", "1,000", "10", "1200", "2,100")  # Create example vector
vec
```

```
## [1] "50"    "200"   "1,000" "10"    "1200"  "2,100"
```

```
as.numeric(vec)
```

```
## Warning: NAs introduced by coercion
```

```
## [1]   50  200   NA   10 1200   NA
```

One of the items is not a number

```
vec <- gsub(",", "", vec)
vec
```

```
## [1] "50"   "200"  "1000" "10"   "1200" "2100"
```

```
as.numeric(vec)
```

```
## [1]   50  200 1000   10 1200 2100
```

Questions?