Practical: Data I/O

BaselRBootcamp 2017

Slides

Here a link to the lecture slides for this session: **LINK** (https://therbootcamp.github.io/_sessions/D1S3_DataIO/Data_IO.html)

Overview

In this practical you'll learn how to read and save data. By the end of this practical you will know how to:

1. Create a project
2. Identify the location of a file
3. Read in data of various types
4. Use R's file connections
5. Scrape the internet (if you get there)

Functions

Here are the main read-in functions:

Function	Description
`read_csv()`	Read flat csv file
`read_sas()`	Read SAS file
`read_sav()`	Read SPSS file
`readRDS()`	Read RDS file
`file(...,'r'), readLines`	Read from file conection

Here are the main export functions:

Function	Description
`write_csv()`	Write flat csv file
`write_sas()`	Write SAS file
`write_sav()`	Write SPSS file
`saveRDS()`	Save RDS file
`file(...,'w'), readLines`	Write to file conection

Tasks

The tutorial begins with the titanic data set, which contains records of 1313 passengers on their name (`Name`), their passenger class (`PClass`), their age (`Age`), their sex (`Sex`), whether they survived (`Survived`), and finally a numeric coding of their sex (`SexCode`).

Later you will be working with a randomly generated artificial data set of mine, which will help demonstrate differences in speed and file size.

Please download the following data files and save them to an appropriate location on the disc:

titanic.csv
(https://raw.githubusercontent.com/therbootcamp/therbootcamp.github.io/master/_sessions/_data/titanic.csv)
titanic.sav
(https://raw.githubusercontent.com/therbootcamp/therbootcamp.github.io/master/_sessions/_data/titanic.sav)
titanic.sas7bdat
(https://raw.githubusercontent.com/therbootcamp/therbootcamp.github.io/master/_sessions/_data/titanic.sas7bdat)
my_data.csv
(https://raw.githubusercontent.com/therbootcamp/therbootcamp.github.io/master/_sessions/_data/my_data.csv)

If you can, please verify whether the `.sav` and `sas7bdat` are appropriate SPSS and SAS files, respectively.

Read to `tibble`

1. Create a new **project** either in the location of the downloaded data files or its parent folder (one folder higher). To do this go to *File/New Project…*, select existing directory, and enter the desired folder. This step will ease reading and saving data, as it sets the working directory to the proximity of the data files. Confirm this using `getwd()`.

2. Identify the filepath (incl. filename) for each of the three data files. RStudio makes it really easy. Simply write quotation marks in your script editor, e.g., `""`, move the (text) curser between the marks, and press the tab key. This opens a file menu that allows you to conveniently select the path to the file. The result is a character string that exactly defines the location of the file relative to the current working directory. Oftentimes, it is convenient to do this directly within the read function, e.g., `read_csv("")`.

3. Load the packages `readr` and `haven` using `library()`, and read in the three titanic data files usig `read_csv()`, `read_sas()`, and `read_sav()`. Call them `df_csv`, `df_sas`, and `df_sav`, respectively. Inspect each of the imported objects using `print()` and `str()`. What kind of object are they?

```
df_csv <- read_csv('data/titanic.csv')
```

```
Warning: Missing column names filled in: 'X1' [1]
```

```
Parsed with column specification:
cols(
  X1 = col_integer(),
  Name = col_character(),
  PClass = col_character(),
  Age = col_double(),
  Sex = col_character(),
  Survived = col_integer(),
  SexCode = col_integer()
)
```

```
df_sas <- read_sas('data/titanic.sas7bdat')
df_sav <- read_sav('data/titanic.sav')
df_csv ; df_sas ; df_sav
```

```
# A tibble: 1,313 x 7
      X1                                          Name PClass   Age    Sex
   <int>                                         <chr>  <chr> <dbl>  <chr>
 1     1                 Allen, Miss Elisabeth Walton    1st 29.00 female
 2     2                  Allison, Miss Helen Loraine    1st  2.00 female
 3     3          Allison, Mr Hudson Joshua Creighton    1st 30.00   male
 4     4 Allison, Mrs Hudson JC (Bessie Waldo Daniels)    1st 25.00 female
 5     5                Allison, Master Hudson Trevor    1st  0.92   male
 6     6                           Anderson, Mr Harry    1st 47.00   male
 7     7             Andrews, Miss Kornelia Theodosia    1st 63.00 female
 8     8                       Andrews, Mr Thomas, jr    1st 39.00   male
 9     9  Appleton, Mrs Edward Dale (Charlotte Lamson)    1st 58.00 female
10    10                       Artagaveytia, Mr Ramon    1st 71.00   male
# ... with 1,303 more rows, and 2 more variables: Survived <int>,
#   SexCode <int>
```

```
# A tibble: 1,313 x 7
      X1                                          Name PClass   Age    Sex
   <dbl>                                         <chr>  <chr> <dbl>  <chr>
 1     1                 Allen, Miss Elisabeth Walton    1st 29.00 female
 2     2                  Allison, Miss Helen Loraine    1st  2.00 female
 3     3          Allison, Mr Hudson Joshua Creighton    1st 30.00   male
 4     4 Allison, Mrs Hudson JC (Bessie Waldo Daniels)    1st 25.00 female
 5     5                Allison, Master Hudson Trevor    1st  0.92   male
 6     6                           Anderson, Mr Harry    1st 47.00   male
 7     7             Andrews, Miss Kornelia Theodosia    1st 63.00 female
 8     8                       Andrews, Mr Thomas, jr    1st 39.00   male
 9     9  Appleton, Mrs Edward Dale (Charlotte Lamson)    1st 58.00 female
10    10                       Artagaveytia, Mr Ramon    1st 71.00   male
# ... with 1,303 more rows, and 2 more variables: Survived <dbl>,
#   SexCode <dbl>
```

```
# A tibble: 1,313 x 7
      X1                                          Name PClass   Age    Sex
   <dbl>                                         <chr>  <chr> <dbl>  <chr>
 1     1                 Allen, Miss Elisabeth Walton    1st 29.00 female
 2     2                  Allison, Miss Helen Loraine    1st  2.00 female
 3     3          Allison, Mr Hudson Joshua Creighton    1st 30.00   male
 4     4 Allison, Mrs Hudson JC (Bessie Waldo Daniels)    1st 25.00 female
 5     5                Allison, Master Hudson Trevor    1st  0.92   male
 6     6                           Anderson, Mr Harry    1st 47.00   male
 7     7             Andrews, Miss Kornelia Theodosia    1st 63.00 female
 8     8                       Andrews, Mr Thomas, jr    1st 39.00   male
 9     9  Appleton, Mrs Edward Dale (Charlotte Lamson)    1st 58.00 female
10    10                       Artagaveytia, Mr Ramon    1st 71.00   male
# ... with 1,303 more rows, and 2 more variables: Survived <dbl>,
#   SexCode <dbl>
```

```
str(df_csv) ; str(df_sas) ; str(df_sav)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   1313 obs. of  7 variables:
 $ X1      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Name    : chr  "Allen, Miss Elisabeth Walton" "Allison, Miss Helen Loraine" "Allison,
Mr Hudson Joshua Creighton" "Allison, Mrs Hudson JC (Bessie Waldo Daniels)" ...
 $ PClass  : chr  "1st" "1st" "1st" "1st" ...
 $ Age     : num  29 2 30 25 0.92 47 63 39 58 71 ...
 $ Sex     : chr  "female" "female" "male" "female" ...
 $ Survived: int  1 0 0 0 1 1 1 0 1 0 ...
 $ SexCode : int  1 1 0 1 0 0 1 0 1 0 ...
 - attr(*, "spec")=List of 2
  ..$ cols   :List of 7
  .. ..$ X1      : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  .. ..$ Name    : list()
  .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
  .. ..$ PClass  : list()
  .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
  .. ..$ Age     : list()
  .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
  .. ..$ Sex     : list()
  .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
  .. ..$ Survived: list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  .. ..$ SexCode : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  ..$ default: list()
  .. ..- attr(*, "class")= chr  "collector_guess" "collector"
  ..- attr(*, "class")= chr "col_spec"
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   1313 obs. of  7 variables:
 $ X1      : num  1 2 3 4 5 6 7 8 9 10 ...
 $ Name    : chr  "Allen, Miss Elisabeth Walton" "Allison, Miss Helen Loraine" "Allison,
Mr Hudson Joshua Creighton" "Allison, Mrs Hudson JC (Bessie Waldo Daniels)" ...
 $ PClass  : chr  "1st" "1st" "1st" "1st" ...
 $ Age     : num  29 2 30 25 0.92 47 63 39 58 71 ...
 $ Sex     : chr  "female" "female" "male" "female" ...
 $ Survived: num  1 0 0 0 1 1 1 0 1 0 ...
 $ SexCode : num  1 1 0 1 0 0 1 0 1 0 ...
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   1313 obs. of  7 variables:
 $ X1      : atomic  1 2 3 4 5 6 7 8 9 10 ...
  ..- attr(*, "format.spss")= chr "F8.0"
 $ Name    : atomic  Allen, Miss Elisabeth Walton Allison, Miss Helen Loraine Allison, M
r Hudson Joshua Creighton Allison, Mrs Hudson | __truncated__ ...
  ..- attr(*, "format.spss")= chr "A62"
 $ PClass  : atomic  1st 1st 1st 1st ...
  ..- attr(*, "format.spss")= chr "A3"
 $ Age     : atomic  29 2 30 25 0.92 47 63 39 58 71 ...
  ..- attr(*, "format.spss")= chr "F8.2"
 $ Sex     : atomic  female female male female ...
  ..- attr(*, "format.spss")= chr "A6"
 $ Survived: atomic  1 0 0 0 1 1 1 0 1 0 ...
  ..- attr(*, "format.spss")= chr "F8.0"
 $ SexCode : atomic  1 1 0 1 0 0 1 0 1 0 ...
  ..- attr(*, "format.spss")= chr "F8.0"
```

4. Try to verify that the three data sets are identical. To do this use the *is-equal-to* operator `==` introduced in the last tutorial and the arithmetic mean function `mean()`. Remember, that logical values can be coerced to 0 and 1. Note, that the data sets contain NA's, that is missing values. One way to see this is via the summary function `summary()`. Specifically, the variable `Age` conains missing values. Use the help file `?mean` to find out how to compute the mean while ignoring missing values and then use `==` to verify the equality between the three data sets.

```
mean(df_csv == df_sas, na.rm=T) == 1
```

```
[1] TRUE
```

```
mean(df_csv == df_sav, na.rm=T) == 1
```

```
[1] TRUE
```

```
mean(df_sas == df_sav, na.rm=T) == 1
```

```
[1] TRUE
```

Write to disk

5. The first column in the titanic set contains row numbers. Older read and write functions in R would by default add a row number column to the data, when writing to disk. Newer functions, however, do not show this behavior. ELiminate the first column from `df_csv` using a negative index. Negtive indices mean *omit* rather than *select*. E.g., c(1, 2, 3)[-2] returns the vector without the second value. Then write the reduced data frame to disk using `write_csv()`. Use the exact same file path. Observe whether R gives you a warning before (over-)writing the data?

```
df_csv = df_csv[,-1]
write_csv(df_csv,"data/titanic_without_col1.csv")
```

6. Determine the file path to `my_data.csv` and read it in using `read_csv()`. Now write the data back to disk using `write_sas`, `write_sav`, and also `saveRDS`. Each of these functions take as the first argument the data frame and as the second argumen the file path. To obtain the latter adapt the filepath of `my_data.csv` by hand to make sure that the file path has the correct file ending, i.e., `.sas7bdat`, `.sav`, and `.RDS`, respectively. While you write the individual data sets pay attention to the speed of execution.

```
my_data = read_csv("data/my_data.csv")
```

```
Parsed with column specification:
cols(
  id = col_character(),
  var_1 = col_double(),
  var_2 = col_double()
)
```

```
write_sas(my_data,"data/my_data.sas7bdat")
write_sav(my_data,"data/my_data.sav")
saveRDS(my_data,"data/my_data.RDS")
```

7. You may have noticed that the `saveRDS` function was the slowest. This is because `.RDS` generally leads to the highest degree of compression, i.e., the smalles data files. To verify this use the `file.info()` on each of the four files, `my_data.csv` , `my_data.sas7bdat` , `my_data.sav` , and `my_data.RDS` (using the correct path). Look out for the `size` element, which gives the size in `bytes` .

```
file.info("data/my_data.csv")
```

```
                     size isdir mode               mtime
data/my_data.csv 86932096 FALSE  644 2017-09-09 02:55:14
                               ctime               atime uid gid  uname
data/my_data.csv 2017-10-08 10:45:50 2017-10-08 11:05:42 502  20 dwulff
                 grname
data/my_data.csv  staff
```

```
file.info("data/my_data.sas7bdat")
```

```
                          size isdir mode               mtime
data/my_data.sas7bdat 40559616 FALSE  644 2017-10-08 11:05:45
                                    ctime               atime uid gid
data/my_data.sas7bdat 2017-10-08 11:05:45 2017-10-08 11:05:48 502  20
                       uname grname
data/my_data.sas7bdat dwulff  staff
```

```
file.info("data/my_data.sav")
```

```
                     size isdir mode               mtime
data/my_data.sav 48000465 FALSE  644 2017-10-08 11:05:46
                               ctime               atime uid gid  uname
data/my_data.sav 2017-10-08 11:05:46 2017-10-08 11:05:49 502  20 dwulff
                 grname
data/my_data.sav  staff
```

```
file.info("data/my_data.RDS")
```

```
                     size isdir mode               mtime
data/my_data.RDS 37462140 FALSE  644 2017-10-08 11:05:49
                               ctime               atime uid gid  uname
data/my_data.RDS 2017-10-08 11:05:49 2017-10-08 11:05:49 502  20 dwulff
                 grname
data/my_data.RDS  staff
```

File connections

8. The most basic way to handle files is via file connections. The first step of working with file connections is to establish a file connections using, e.g., `file()` . The main arguments to `file` are a location (a file path, url, etc.) and a mode indicator, e.g., `r` for `Open for reading in text mode` . Try opening a connection to the `titanic.csv` dataset. To do this, you must assign the output of `file()` to an object, which then is the connection, e.g., `con = file("my_path","r")` .

```
con = file('data/titanic.csv','r')
```

9. Now that the connection is open, you can read the file using, e.g., `readLines`. `readLines` iterates through the file line by line and returns each line as a character string. Store the output in an object. Try reading the file. When done close the connection using `close(my_con)` and inspect the data.

```
lin = readLines(con)
```

```
Warning in readLines(con): incomplete final line found on 'data/
titanic.csv'
```

```
close(con)
```

10. As you can see the read in data looks much messier than with using `read_csv()`. If you're up for the challenge you can try to nonetheless parse this data. A good way is to split each line using the `stri_split_fixed` function using the comma `,` (from the `stringi` package). This will return a list of vectors, where each vector is one row of the data frame. Next step is to bind the rows to a matrix using `do.call(rbind,my_list)`. What's then left is transform the matrix to a data frame and to take care of column types and names.

```
# read lines
con = file('data/titanic.csv','r')
lin <- readLines(con)
```

```
Warning in readLines(con): incomplete final line found on 'data/
titanic.csv'
```

```
close(con)

# load string operation package
require(stringi)
```

```
Loading required package: stringi
```

```
require(tibble)
```

```
Loading required package: tibble
```

```
Warning: package 'tibble' was built under R version 3.4.1
```

```
# split and recombine
spl <- stri_split_fixed(lin,',')
mat <- do.call(rbind,spl)
```

```
Warning in (function (..., deparse.level = 1) : number of columns of result
is not a multiple of vector length (arg 1)
```

```
df  <- as.tibble(mat)
# take care of types
```

Scraping the internet

11. Finally on to something rather different. Connections can also be established to data located outside one's own disk, such as, for instance, the world wide web. Speicifically, using `url()` we can establish a connection to a webpage, which we then can use to retrieve information. This is used, for instance, by the very convenient `rvest` package. To illustrate how easy it can be, consider the following code that downloads and parses Milestones table of R's Wikipedia page.

```
# load package
install.packages('rvest', repos = "https://stat.ethz.ch/CRAN/")
```

```


The downloaded binary packages are in
	/var/folders/4j/gkx0z2kn1b5djq50kwgl2wdc0000gp/T//RtmpntNjGJ/downloaded_packages
```

```
library(rvest)
```

```
Loading required package: xml2
```

```


Attaching package: 'rvest'
```

```
The following object is masked from 'package:readr':

    guess_encoding
```

```
library(magrittr)

# get html
url = 'https://en.wikipedia.org/wiki/R_(programming_language)'
page = read_html(url)

# get table
# use XPath from inspect page (e.g., in Chrome)
table = page %>% html_node(xpath = '//*[@id="mw-content-text"]/div/table[2]') %>% html_t
able()

# create tibble
as.tibble(table)
```

```
# A tibble: 13 x 3
   Release       Date
     <chr>      <chr>
 1    0.16
 2    0.49 1997-04-23
 3    0.60 1997-12-05
 4  0.65.1 1999-10-07
 5     1.0 2000-02-29
 6     1.4 2001-12-19
 7     2.0 2004-10-04
 8     2.1 2005-04-18
 9    2.11 2010-04-22
10    2.13 2011-04-14
11    2.14 2011-10-31
12    2.15 2012-03-30
13     3.0 2013-04-03
# ... with 1 more variables: Description <chr>
```

Additional reading

- For more details on all steps of data analysis check out Hadley Wickham's R for Data Science (http://r4ds.had.co.nz/).
- For more advanced content on objects check out Hadley Wickham's Advanced R (http://adv-r.had.co.nz/).
- For more on pirates and data analysis check out the respective chapters in YaRrr! The Pirate's Guide to R YaRrr! Chapter Link (https://bookdown.org/ndphillips/YaRrr/htests.html)