SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Reading Data

Dr.Sc. Oleksii Yehorchenkov

Department of Spatial Planning

Reading Data

There are a few principal functions reading data into R.

- `read.table`, `read.csv`, for reading tabular data
- `readLines`, for reading lines of a text file
- `source`, for reading in R code files (inverse of `dump`)
- `dget`, for reading in R code files (inverse of `dput`)
- `load`, for reading in saved workspaces
- `unserialize`, for reading single R objects in binary form

Writning Data

There are analogous functions for writing data to files

- `write.table`
- `writeLines`
- `dump`
- `dput`
- `save`
- `serialize`

Reading Data Files with `read.table`

The `read.table` function is one of the most commonly used functions for reading data. It has a few important arguments:

- `file`, the name of a file, or a connection
- `header`, logical indicating if the file has a header line
- `sep`, a string indicating how the columns are separated
- `colClasses`, a character vector indicating the class of each column in the dataset
- `nrows`, the number of rows in the dataset
- `comment.char`, a character string indicating the comment character
- `skip`, the number of lines to skip from the beginning

`read.table` (cont'd)

For small to moderately sized datasets, you can usually call read.table without specifying any other arguments

```
data <- read.table("foo.txt")
```

R will automatically:

- skip lines that begin with a #
- figure out how many rows there are (and how much memory needs to be allocated)
- figure what type of variable is in each column of the table. Telling R all these things directly makes R run faster and more efficiently
- `read.csv` is identical to read.table except that the default separator is a comma

Reading in Larger Datasets with `read.table`

With much larger datasets, doing the following things will make your life easier and will prevent R from choking.

- Read the help page for `read.table`, which contains many hints
- Make a rough calculation of the memory required to store your dataset. If the dataset is larger than the amount of RAM on your computer, you can probably stop right here.
- Set `comment.char = ""` if there are no commented lines in your file

Reading in Larger Datasets with `read.table`

- Use the `colClasses` argument. Specifying this option instead of using the default can make `read.table` run **MUCH** faster, often twice as fast. In order to use this option, you have to know the class of each column in your data frame. If all of the columns are "numeric", for example, then you can just set `colClasses = "numeric"`. A quick an dirty way to figure out the classes of each column is the following:

```r
initial <- read.table("datatable.txt", nrows = 100)
classes <- sapply(initial, class)
tabAll <- read.table("datatable.txt", colClasses = classes)
```

- Set `nrows`. This doesn't make R run faster but it helps with memory usage. A mild overestimate is okay.

Know Your System

In general, when using R with larger datasets, it’s useful to know a few things about your system.

- How much memory is available?
- What other applications are in use?
- Are there other users logged into the same system?
- What operating system?
- Is the OS 32 or 64 bit?

Calculating Memory Requirements

I have a data frame with 1,500,000 rows and 120 columns, all of which are numeric data. Roughly, how much memory is required to store this data frame?

```
1,500,000 × 120 × 8 bytes/numeric

= 1440000000 bytes

= 1440000000 / 2^20 bytes/MB

= 1,373.29 MB

= 1.34 GB
```

Reading Data with `readr` package

The readr package is recently developed by Hadley Wickham to deal with reading in large flat files quickly. The package provides replacements for functions like `read.table()` and `read.csv()`. The analogous functions in readr are `read_table()` and `read_csv()`. These functions are often much faster than their base R analogues and provide a few other nice features such as progress meters.

You can find documentation about the package on the webpage https://readr.tidyverse.org/

Reading Data with readr package

Let’s try to read data from `olympic.csv` file.

```r
library (readr)

df <- read_csv("data/olympics.csv")

head(df, 10)
```

```
# A tibble: 10 × 16
   `0`   `1`   `2`   `3`   `4`   `5`   `6`   `7`   `8`   `9`   `10`  `11`  `12` 
   <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 <NA>  № Su… 01 !  02 !  03 !  Total № Wi… 01 !  02 !  03 !  Total № Ga… 01 ! 
 2 Afgh… 13    0     0     2     2     0     0     0     0     0     13    0    
 3 Alge… 12    5     2     8     15    3     0     0     0     0     15    5    
 4 Arge… 23    18    24    28    70    18    0     0     0     0     41    18   
 5 Arme… 5     1     2     9     12    6     0     0     0     0     11    1    
 6 Aust… 2     3     4     5     12    0     0     0     0     0     2     3    
 7 Aust… 25    139   152   177   468   18    5     3     4     12    43    144  
 8 Aust… 26    18    33    35    86    22    59    78    81    218   48    77   
 9 Azer… 5     6     5     15    26    5     0     0     0     0     10    6    
10 Baha… 15    5     2     5     12    0     0     0     0     0     15    5    
# ℹ 3 more variables: `13` <chr>, `14` <chr>, `15` <chr>
```

Reading Data with readr package

As we can see, the header was read incorrectly. Let’s skip the first line of the file.

```r
library (readr)

df <- read_csv("data/olympics.csv", skip = 1)

head(df, 10)
```

```
# A tibble: 10 × 16
   ...1           `№ Summer` `01 !...3` `02 !...4` `03 !...5` Total...6 `№ Winter`
   <chr>               <dbl>      <dbl>      <dbl>      <dbl>     <dbl>      <dbl>
 1 Afghanistan…           13          0          0          2         2          0
 2 Algeria (AL…           12          5          2          8        15          3
 3 Argentina (…           23         18         24         28        70         18
 4 Armenia (AR…            5          1          2          9        12          6
 5 Australasia…            2          3          4          5        12          0
 6 Australia (…           25        139        152        177       468         18
 7 Austria (AU…           26         18         33         35        86         22
 8 Azerbaijan …            5          6          5         15        26          5
 9 Bahamas (BA…           15          5          2          5        12          0
10 Bahrain (BR…            8          0          0          1         1          0
# ℹ 9 more variables: `01 !...8` <dbl>, `02 !...9` <dbl>, `03 !...10` <dbl>,
#   Total...11 <dbl>, `№ Games` <dbl>, `01 !...13` <dbl>, `02 !...14` <dbl>,
#   `03 !...15` <dbl>, `Combined total` <dbl>
```

Reading Data with readr package

In addition it is possible to specify column classes with `col_types` argument

```r
library (readr)

df <- read_csv("data/olympics.csv", skip = 1, col_types = "ciiiiiiiiiiiiiii")

head(df, 10)
```

```
# A tibble: 10 × 16
   ...1           `№ Summer` `01 !...3` `02 !...4` `03 !...5` Total...6 `№ Winter`
   <chr>               <int>      <int>      <int>      <int>     <int>      <int>
 1 Afghanistan…           13          0          0          2         2          0
 2 Algeria (AL…           12          5          2          8        15          3
 3 Argentina (…           23         18         24         28        70         18
 4 Armenia (AR…            5          1          2          9        12          6
 5 Australasia…            2          3          4          5        12          0
 6 Australia (…           25        139        152        177       468         18
 7 Austria (AU…           26         18         33         35        86         22
 8 Azerbaijan …            5          6          5         15        26          5
 9 Bahamas (BA…           15          5          2          5        12          0
10 Bahrain (BR…            8          0          0          1         1          0
# ℹ 9 more variables: `01 !...8` <int>, `02 !...9` <int>, `03 !...10` <int>,
#   Total...11 <int>, `№ Games` <int>, `01 !...13` <int>, `02 !...14` <int>,
#   `03 !...15` <int>, `Combined total` <int>
```

Textual Formats

There are a variety of ways that data can be stored, including structured text files like CSV or tab-delimited, or more complex binary formats. However, there is an intermediate format that is textual, but not as simple as something like CSV. The format is native to R and is somewhat readable because of its textual nature.

- dumping and dputing are useful because the resulting textual format is editable, and in the case of corruption, potentially recoverable.
- Unlike writing out a table or csv file, dump and dput preserve the metadata (sacrificing some readability), so that another user doesn't have to specify it all over again.
- Textual formats can work much better with version control programs like **git** which can only track changes meaningfully in text files
- Textual formats can be longer-lived; if there is corruption somewhere in the file, it can be easier to fix the problem
- Textual formats adhere to the "Unix philosophy"
- Downside: The format is not very space-efficient

dputting R Objects

Another way to pass data around is by deparsing the R object with dput and reading it back in using dget.

```r
y <- data.frame(a = 1, b = "a")
dput(y)
```

```
structure(list(a = 1, b = "a"), class = "data.frame", row.names = c(NA,
-1L))
```

```r
dput(y, file = "data/y.R")
new.y <- dget("data/y.R")
new.y
```

```
  a b
1 1 a
```

Dumping R Objects

Multiple objects can be deparsed using the dump function and read back in using source.

```r
x <- "foo"
y <- data.frame(a = 1, b = "a")
dump(c("x", "y"), file = "data/data.R")
rm(x, y)
source("data/data.R")
y
```

```
  a b
1 1 a
```

```r
x
```

```
[1] "foo"
```

Interfaces to the Outside World

Data are read in using connection interfaces. Connections can be made to files (most common) or to other more exotic things.

- file, opens a connection to a file
- gzfile, opens a connection to a file compressed with gzip
- bzfile, opens a connection to a file compressed with bzip2
- url, opens a connection to a webpage

File Connections

```r
str(file)
```

```
function (description = "", open = "", blocking = TRUE, encoding = 
getOption("encoding"), 
    raw = FALSE, method = getOption("url.method", "default"))
```

- description is the name of the file
- open is a code indicating
  - “r” read only
  - “w” writing (and initializing a new file)
  - “a” appending
  - “rb”, “wb”, “ab” reading, writing, or appending in binary mode (Windows)

Connections

In general, connections are powerful tools that let you navigate files or other external objects. In practice, we often don't need to deal with the connection interface directly.

```r
con <- file("foo.txt", "r")
data <- read.csv(con)
close(con)
```

is the same as

```r
data <- read.csv("foo.txt")
```

Reading Lines of a Text File

```r
con <- file("data/words.txt")
x <- readLines(con, 30)
x
```

```
 [1] "aa"          "aah"         "aahed"       "aahing"      "aahs"       
 [6] "aal"         "aalii"       "aaliis"      "aals"        "aardvark"   
[11] "aardvarks"   "aardwolf"    "aardwolves"  "aas"         "aasvogel"   
[16] "aasvogels"   "aba"         "abaca"       "abacas"      "abaci"      
[21] "aback"       "abacus"      "abacuses"    "abaft"       "abaka"      
[26] "abakas"      "abalone"     "abalones"    "abamp"       "abampere"   
```

```r
close(con)
```

writeLines takes a character vector and writes each element one line at a time to a text file.

Reading Lines of a Text File

readLines can be useful for reading in lines of webpages

```r
con <- url("http://stuba.sk", "r")
x <- readLines(con)
close(con)

head(x, 10)
```

```
 [1] ""
 [2] ""
 [3] "<!DOCTYPE html>"
 [4] "<html lang=\"sk\" >"
 [5] "<head>"
 [6] "            <meta http-equiv=\"Content-Type\" content=\"text/html;
charset=utf-8\" >"
 [7] "    <meta name=\"copyright\" content=\"Slovenská technická univerzita v
Bratislave (STU)\" >"
 [8] "    <meta name=\"description\" content=\"Slovenská technická univerzita
v Bratislave (STU) - oficiálna stránka univerzity\" >"
 [9] "    <meta name=\"keywords\" content=\"univerzita, student, skola,
vzdelavanie, absolvent, veda, vyskum, projekt, medzinarodny, spolupraca\" >"
[10] "    <meta name=\"viewport\" content=\"width=device-width, initial-
scale=1.0, maximum-scale=5.0\" >"
```