DATA 133 - Introduction to Data Science I

Instructor: Renzhi Cao
Computer Science Department
Pacific Lutheran University

Announcements

- Read books: Page 23-41
- Pair Practice from last Thursday is due, please submit on Sakai.
- Go through pair practice for any questions
- Quiz 2 (30 mins) Available till 10 pm today

Reference book

- R Programming for Data Science. By Roger Peng.
  ISBN-10: 1365056821, April 20, 2016.

Pictures from: https://www.google.com/search?

Learning in today

- R basics - Reading and Writing data from file.

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

There are of course, many R packages that have been developed to read in all kinds of other datasets, and you may need to resort to one of these packages if you are working in a specific area.

Reading Data Files with read.table()
The read.table() function 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. By default read.table() reads an entire file.
- comment.char, a character string indicating the comment character. This defalts to "#". If there are no commented lines in your file, it's worth setting this to be the empty string "".
- skip, the number of lines to skip from the beginning
- stringsAsFactors, should character variables be coded as factors?

Reading data

Let's try:

Download DATA133_example1.csv from website

```
> data <- read.table("DATA133_example1.csv")
```

In this case, R will automatically:

- 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.

The read.csv() function is identical to read.table except that some of the defaults are set differently (like the sep argument).

Reading data

Let's try:

```
> data <- read.table("DATA133_example1.csv", skip=1)   # to skip the header

> data[1:2,]          # the first two rows

>data[,1]             # the first column
```

Reading data

Let's try if you just want to have a look at the data:

```
> data <- read.table("DATA133_example1.csv", nrows=5)   # only read five rows
> data[1:2,]          # the first two rows
>data[,1]             # the first column
```

Useful points for larger datasets:

- How much memory is available on your system?
- What other applications are in use? Can you close any of them?
- Are there other users logged into the same system?
- Operating systems, some limit the amount of memory a single process can access

Practice

1. Download and read the first 100 rows from DATA133_example1.csv, skip the headers, and assign it to variable ‘data’.

2. Check the class type of variable ‘data’, and print the total number of rows and columns for ‘data’. (Hint: nrow(), ncol())

3. Convert variable ‘data’ to matrix type and assign it to ‘dataM’. (Hint: data.matrix())

Break

Writing data

There are analogous functions for writing data to files

- write.table, for writing tabular data to text files (i.e. CSV) or connections
- writeLines, for writing character data line-by-line to a file or connection
- dump, for dumping a textual representation of multiple R objects
- dput, for outputting a textual representation of an R object
- save, for saving an arbitrary number of R objects in binary format (possibly compressed) to a file.
- serialize, for converting an R object into a binary format for outputting to a connection (or file).

Writing data

Let's try:

```
>m <- matrix(seq(1,100,5),4,5)

>m

>write.table(m,sep='  ',file="output.R")

>rm(m)        # delete the m object

>m

>m <- read.table("output.R",sep ='   ')    # what happens if you don't add space for sep?

>m
```

Using the readr Package

The readr package is recently developed by Hadley Wickham to deal with reading in large flat files quickly.
read.table() => read_table()
read.csv() => read_csv()

```
install.packages("readr")
```

```
>library(readr)
>read_csv(mtcars_path)
>write_csv(mtcars, mtcars_path)
```

dput() and dump()

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

```
> ## Create a data frame
> y <- data.frame(a = 1, b = "a")
> ## Print 'dput' output to console
> dput(y)
```

The output of dput() can also be saved directly to a file.

```
> ## Send 'dput' output to a file
> dput(y, file = "y.R")
> ## Read in 'dput' output from a file
> new.y <- dget("y.R")
> new.y
```

dput() and dump()

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

```
> x <- "foo"
> y <- data.frame(a = 1L, b = "a")
```

We can dump() R objects to a file by passing a character vector of their names.

```
> dump(c("x", "y"), file = "data.R")
> rm(x, y)                    # this is going to remove the x and y object
```

The inverse of dump() is source().

```
> source("data.R")
> str(y)
>x
```

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

Connections

Connections to text files can be created with the file() function.

```
> str(file)
```

The open argument allows for the following options:

- "r" open file in read only mode
- "w" open a file for writing (and initializing a new file)
- "a" open a file for appending
- "rb", "wb", "ab" reading, writing, or appending in binary mode (Windows)

Connections

In practice, we often don't need to deal with the connection interface directly as many functions for reading and writing data just deal with it in the background.

```
> ## Create a connection to 'foo.txt'
> con <- file("foo.txt")
>
> ## Open connection to 'foo.txt' in read-only mode
> open(con, "r")
>
> ## Read from the connection
> data <- read.csv(con)
>
> ## Close the connection
> close(con)
```

which is the same as

> data <- read.csv("foo.txt")

Reading lines from a text file

Text files can be read line by line using the readLines() function.

```
> ## Open connection to gz-compressed text file
> con <- gzfile("words.gz")
> x <- readLines(con, 10)
```

The above example used the gzfile() function which is used to create a connection to files compressed using the gzip algorithm.

There is a complementary function writeLines() that takes a character vector and writes each element of the vector one line at a time to a text file.

Reading lines from a URL

The readLines() function for webpages.

```
> ## Open a URL connection for reading
> con <- url("http://www.jhsph.edu", "r")
>
> ## Read the web page
> x <- readLines(con)
>
> ## Print out the first few lines
> head(x)
```

Using URL connections can be useful:
1. producing a reproducible analysis
2. Opening a web browser and downloading a dataset by hand.

Of course, the code you write with connections may not be executable at a later date if things on the server side are changed or reorganized.

Practice of R

Try it!
Pair-programming is due by next class
Read text book