Reading Tabular Data

Principle function

- `read.table`, `read.csv` for reading tabular data(common use, return data frame)
- `readLines` for reading lines of a text file (return character vector)
- `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

Reading Data Files with `read.table`

Arguments

- `file`, the name of a file, or connection(string or path)
- `header` logical indicating if the file has a header line(not the actual data)
- `sep`, a string indicating how columns are separated(comma or semicolon)
- `colClasses`, a character vector indicating the class of each column in the dataset(numerical logical .etc.)
- `nrows` the number of rows in dataset
- `comment.char` a character string indicating the comment character(#)
- `skip`, the number of lines to skip from beginning(non-data region)
- `stringAsFactors` should character variables be coded as factors （Default is true)

`read.table`

Small to moderately sized datasets can use `read.table` without specifying any other arguments.

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

R will automatically

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

Reading Large Tables

We need to do something to prevent R from chocking

- Read the help page for `read.table`
- Make a rough calculation of the memory required to store your dataset. If it is larger than your RAM, STOP IT!
- Set `comment.char = ""` if there are no commented lines in your file
- Use `colClasses` argument, if does not set, it will can from one row to another row in the data set.

```
initial <- read.table("datatable.txt", nrows = 100) ## read 100 rows first
classes <- sapply(initial, class)
tabALL <- read.table("datatable.txt",
                     colClass = classes) ## tell classes
```

Know Your System

- Memory(RAM)
- Running application
- Other users running?
- OS
- 32 or 64 bit ?

Calculating Memory requirements

$$A_{rows} \cdot A_{column} \cdot data \tag{1}$$

Numeric requires 8 bytes per object

Textural Format

Good to store data, easy to edit

work better with version control

Adhere the Unix

not space efficient

`dump` and `dput` is the main function of output, contain more metadata

include the data types in every row in the data frame(don't have to specify)

`source` and `dget` is the function that can process the data which output is not `dump` `dget`

dput-ting R Object

Another way to pass data around is by deparsing the R object with `dput` and reading it back in using `dget`(only single object)

```
> y <- data.frame(a = 1, b = "a")
> deput(y) ## summon R code
structure(list(a = 1,
               b = structure(1L, .Label = "a",
                             class = 'factor')),
          .Name = c("a", "b"), row.names = c(NA, -1L),
          class = "data.frame")
> dput(y, file = "y.R")
> new.y <- dget("y.R") ## reconstrust
> new.y
    a  b
1   1  a
```

Dumpling R Objects

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

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

Interfaces to the Outside World

Data are read using connection interfaces. Connections can be made to files or to other more exotic things.

`read.table()`

- `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

```
> str(file)
function (description = "", open = "", blocking = TURE,
          encoding = getOption("encoding"))
```

`description` is the name of the file

`open` is the 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, connection are powerful tools to let you navigate files or other external objects.

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

is the same as

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

Reading Lines of a Text File

```
> con <- file("hw1_data.csv")
> x <- readLines(con, 5)
> x
 [1] "Ozone,Solar.R,Wind,Temp,Month,Day"
 [2] "41,190,7.4,67,5,1"
 [3] "36,118,8,72,5,2"
 [4] "12,149,12.6,74,5,3"
 [5] "18,313,11.5,62,5,4"
```

`writeLines` takes a character vector and writes each element on line at a time to a text file

`readLines` can also be used to read in lines of webpages

```
## This might take time
> con <- url("https://google.com", "r")
> x <- readLines(con)
> head(x)
[1] "<!doctype html>
```