File Input/Output in R

Kelly Pierce, UT Austin, Fall 2013 Introduction to Biological Statistics

Input

Character Delimited Files

- Certain characters are used to indicate column breaks
  - Tab
  - Comma
- Hard returns indicate row breaks

Formatting Guidelines

- Use a text file (extensions ‘.txt’ or ‘.csv’)
- Column headers should not contain special characters or spaces
  - ‘Time 1’ becomes ‘Time.1’ in R

Formatting Guidelines

- Wide Format

DATE	DAY	LOCATION	TIME 1	TEMP 1	RH 1	TIME 2	TEMP 2	RH 2	. . .
1-Jun	4	A	7:17	22.5	91.6	8:20	24.9	89.1	. . .
1-Jun	4	B	7:59	24.7	88.6	9:04	25.9	85.7	. . .
1-Jun	4	C	7:42	23.4	91.7	8:35	27.2	83.5	. . .
1-Jun	4	D	7:34	22.4	93.6	8:28	25.4	88.1	. . .
2-Jun	5	A	7:31	20.4	94.8	8:28	25.4	88.1	. . .
2-Jun	5	B	7:24	20	92.7	8:23	23.2	89.9	. . .
2-Jun	5	C	7:04	20.7	90	8:08	21.2	91.5	. . .
2-Jun	5	D	7:48	20.2	92.8	8:41	23.6	88.6	. . .

- Long Format

DATE	DAY	LOCATION	TIME	TEMP	RH
1-Jun	4	A	7:17	22.5	91.6
1-Jun	4	A	8:20	24.9	89.1
1-Jun	4	A	9:15	26.8	86.7
1-Jun	4	A	10:22	30.7	73.2
1-Jun	4	A	11:09	30.4	66.5
1-Jun	4	B	7:59	24.7	88.6
1-Jun	4	B	9:04	25.9	85.7
1-Jun	4	B	10:11	28.7	79.6
1-Jun	4	B	10:55	30.1	71.5
1-Jun	4	B	11:58	33	58.4
. . .	. . .	. . .	. . .	. . .	. . .

The simplest way to load data:

```
read.csv(file.choose())

# opens up a file browser from
  which you can select your data
  file

# remember to assign your data to
  a variable name!
```

Cheat Sheets

Functions for Navigating the File System

```
getwd()       # returns the current working directory
setwd()       # sets the working directory when given
              a file path
file.path()   # converts a text string to a file path
              (useful when concatenating strings)
dir.create()  # create a new directory (folder) in
              the current working directory
```

Functions for Handling Data

```
# Load Data by specifying the file name, or the file
  path if file is not in working directory. Enclose
  the path in quotes.

read.csv()          # read CSV file
read.table()        # read text file
complete.cases()    # return indexes of cases with no
                    missing values
reshape()           # convert between long and wide
                    formats (tricky to use!)
subset()            # extract only data that meet
                    criteria you're interested in
```

read.table() for TXT

```
read.table(file,
   header = FALSE,
   sep = "",
   quote = "\"'",
   dec = ".",
   row.names,
   col.names,
   as.is = !stringsAsFactors,
   na.strings = "NA",
   colClasses = NA,
   nrows = -1,
   skip = 0,
   check.names = TRUE,
   fill = !blank.lines.skip,
   strip.white = FALSE,
   blank.lines.skip = TRUE,
   comment.char = "#",
   allowEscapes = FALSE,
   flush = FALSE,
   stringsAsFactors = default.stringsAsFactors(),
   fileEncoding = "",
   encoding = "unknown")
```

read.csv() for CSV

```
read.csv(file,
   header = TRUE,
   sep = ",",
   quote="\"",
   dec=".",
   fill = TRUE,
   comment.char="",
   ...)
```

read.delim() for TAB

```
read.delim(file,
   header = TRUE,
   sep = "\t",
   quote="\"",
   dec=".",
   fill = TRUE,
   comment.char="",
   ...)
```

Using Datasets with Missing Values

```
# You may not always have a full data set. R can
  handle missing values in several ways. The option
  you choose may impact the results of your
  analysis.

# Arguments to read() functions indicate which
  values are “missing”:
na.strings = “NA”

# Arguments to analysis functions indicate how to
  handle missing values:
na.action = na.fail
na.action = na.omit
na.action = na.exclude
na.action = na.pass
```

Indexing

```
# reference cells by position
data[row number, column number]

# extract one complete row
data[row number,]

# extract a set of rows
data[row number 1: row number 2,]

# extract a set of columns
data[column number 1: column number 2]
data[,column number 1: column number 2]
```

Subsetting

```
# extract column ‘name1’ from dataset ‘data’

data$name1

# extract all rows in data for which the value in
  column‘name1’ is equal to x

subset(data = data, name1 == x)

# extract all rows in data for which the value in
  column ‘name1’ is equal to x and the value in
  column ‘name2’ is equal to y

subset(data=data, name1 == x & name2 == y)
```

Confirming Proper Data Loading

```
head(data) # print the first 6 rows to screen
names(data) # print the column names to screen

# check that you have the expected number of columns
  and rows using the length() function:

# check number of columns (use any column index)
length(data)
length(data[1,])

# check number of rows (use any row index)
length(data[,1])
```

Check Data Types

```
# Ensure you and R both see the data the same way
  using the typeof() function:

# overall and row data types are likely “list”
typeof(data)
typeof(data[1,])

# column data type may vary
typeof(data[,1])
```

Adding Columns and Rows

```
# Directly add a new row comprised of a vector 'values'. If
   'values' is only one item (e.g., 5), that item is repeated
   in every row. This is called 'recycling'.

col.values<-c(new column data)

# New column is automatically named 'new.name'
data['new.column.name']<- values

# Columns are bound, but no name is assigned to the new
   one. Use function names() to assign name manually.
cbind(data, values)

# Add rows using rbind() or by manually editing your
   data file

row.values<-c(new row data)
rbind(data, row.values)
```

Iterating Over Data

```
# Iteration over data does not require the use of
  for-loops (an Advanced R topic). Instead, the
  following will work:

data['new.values']<- data$col.1 + data$col.2

# This adds the values in col.1 and col.2 row-by-
  row, and places the sum in a new column named
  'new.values'. More complicated operations can be
  perfomred in this manner as well.
```

Output

Files saved by default in working directory

```
# Specify an alternate location by writing out a
  full file path:

write.csv(“/file/path/data.csv”)
```

Functions for Writing Files (Output)

```
# Text – functions simultaneously open empty file,
  write data under the given name, and close the
  file

write.table(file.name, data) # delimiter = space

write.csv(file.name, data)   # delimiter = comma
```

Functions for Writing Files (Output)

```
# Graphics - must open the file with one of the
  functions below, then call separate functions
  to write the plot and close the file

pdf(file.name)          # open a PDF
plot(x,y)               # write the file
dev.off()               # close graphics device

png(file.name)          # write a PNG
plot(x,y)               # write the file
dev.off()               # close graphics device
```