

Stat 849: Accessing data with R

Douglas Bates

2010-10-04

Outline

Contents

1 Data in Packages

Data sets from R packages

- Many of the sample data sets that we will use are available in R packages on CRAN, the Comprehensive R Archive Network.
- For example, we will use data from the `alr3` package created by Sanford Weisberg to supplement his 2005 book *Applied Linear Regression (3rd edition)*
- We attach a package with the `library()` function. (Another option is the `require()` function.) Many sources on R show that you must use `data()` to access a particular data set after attaching the package. For most recent packages this is no longer necessary.

```
1 library(alr3)
```

- A brief description of the data sets and functions in a package is available as `help(package = <pkgname>)`.

Checking the structure of a data set

- It is always a good idea to check the structure of a particular data set before trying to use it. A frequently-used example data set, available as the `brains` data in the `alr3` package, is the data collected by Allison and Cicchetti on average brain weight versus average body weight for 62 species of animal.

```
1 str(brains)
```

```
'data.frame': 62 obs. of 2 variables:
 $ BrainWt: num  44.5 15.5 8.1 423 119.5 ...
 $ BodyWt : num  3.38 0.48 1.35 464.98 36.33 ...
```

- Also see the `help` page for the data, accessed as `?brains`

Checking a summary

- Another function to apply routinely to data frames is `summary()`. Sometimes the output is too wordy to be helpful (the `str()` output is designed to be very concise) but in general `summary()` is a good idea.

```
1 summary(brains)

      BrainWt      BodyWt
Min.   : 0.14   Min.   : 0.005
1st Qu.: 4.25   1st Qu.: 0.600
Median : 17.25  Median : 3.342
Mean   : 283.14 Mean   : 198.794
3rd Qu.: 166.00 3rd Qu.: 48.201
Max.   :5711.86 Max.   :6654.180
```

When categorical data is not stored as a factor

- One bad practice to watch for, even in data packages created by professional statisticians, is failure to represent categorical variables as factors. Consider the `ais` data from `alr3`

```
1 str(ais)

'data.frame': 202 obs. of  14 variables:
 $ Sex  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Ht   : num  196 190 178 185 185 ...
 $ Wt   : num  78.9 74.4 69.1 74.9 64.6 63.7 75.2 62.3 66.5 62...
 $ LBM  : num  63.3 58.5 55.4 57.2 53.2 ...
 $ RCC  : num  3.96 4.41 4.14 4.11 4.45 4.1 4.31 4.42 4.3 4.51 ..
 $ WCC  : num  7.5 8.3 5 5.3 6.8 4.4 5.3 5.7 8.9 4.4 ...
 $ Hc   : num  37.5 38.2 36.4 37.3 41.5 37.4 39.6 39.9 41.1 41...
 $ Hg   : num  12.3 12.7 11.6 12.6 14 12.5 12.8 13.2 13.5 12.7 ..
 $ Ferr : int  60 68 21 69 29 42 73 44 41 44 ...
 $ BMI  : num  20.6 20.7 21.9 21.9 19 ...
 $ SSF  : num  109.1 102.8 104.6 126.4 80.3 ...
 $ Bfat : num  19.8 21.3 19.9 23.7 17.6 ...
 $ Label: Factor w/ 17 levels "f-b_ball","f-field",...: 1 1 1 1 ..
 $ Sport: Factor w/ 10 levels "b_ball","field",...: 1 1 1 1 1 1 ..
```

Summaries of categorical variables

- A summary of a categorical variable should be (the leading part of) a frequency table. If you get a “five number” summary instead, the data are stored in a numeric format (`int` or `num`).

```
1 summary(ais$Sport)

b_ball  field      gym netball      row      swim  t_400m t_sprnt
 25     19         4       23       37       22     29      15
tennis  w_polo
 11      17

1 summary(ais$Sex)

      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.000   0.000   0.000   0.495   1.000   1.000
```

Conversion of categorical variables to factors

- For the `ais` data we need to read the documentation, `?ais`, to find that `Sex` has the coding of 0 for male and 1 for female.
- We can use the `within` function to do the conversion and store the modified version. To avoid confusion it is best to give the modified data set a new name.

```
1 str(AIS <- within(ais, Sex <- factor(Sex,
2                                     labels=c("M", "F"))))
```

```
'data.frame': 202 obs. of 14 variables:
 $ Sex : Factor w/ 2 levels "M","F": 2 2 2 2 2 2 2 2 2 2 ...
 $ Ht : num 196 190 178 185 185 ...
 $ Wt : num 78.9 74.4 69.1 74.9 64.6 63.7 75.2 62.3 66.5 62...
 $ LBM : num 63.3 58.5 55.4 57.2 53.2 ...
 $ RCC : num 3.96 4.41 4.14 4.11 4.45 4.1 4.31 4.42 4.3 4.51 ..
 $ WCC : num 7.5 8.3 5 5.3 6.8 4.4 5.3 5.7 8.9 4.4 ...
 $ Hc : num 37.5 38.2 36.4 37.3 41.5 37.4 39.6 39.9 41.1 41...
 $ Hg : num 12.3 12.7 11.6 12.6 14 12.5 12.8 13.2 13.5 12.7 ..
 $ Ferr: int 60 68 21 69 29 42 73 44 41 44 ...
 $ BMI : num 20.6 20.7 21.9 21.9 19 ...
 $ SSF : num 109.1 102.8 104.6 126.4 80.3 ...
 $ Bfat : num 19.8 21.3 19.9 23.7 17.6 ...
 $ Label: Factor w/ 17 levels "f-b_ball","f-field",...: 1 1 1 1 ..
 $ Sport: Factor w/ 10 levels "b_ball","field",...: 1 1 1 1 1 1 ..
```

Summary after conversion

```
1 summary(AIS)
```

Sex	Ht	Wt	LBM
M:102	Min. :148.9	Min. : 37.80	Min. : 34.36
F:100	1st Qu.:174.0	1st Qu.: 66.53	1st Qu.: 54.67
	Median :179.7	Median : 74.40	Median : 63.03
	Mean :180.1	Mean : 75.01	Mean : 64.87
	3rd Qu.:186.2	3rd Qu.: 84.12	3rd Qu.: 74.75
	Max. :209.4	Max. :123.20	Max. :106.00

RCC	WCC	Hc
Min. :3.800	Min. : 3.300	Min. :35.90
1st Qu.:4.372	1st Qu.: 5.900	1st Qu.:40.60
Median :4.755	Median : 6.850	Median :43.50
Mean :4.719	Mean : 7.109	Mean :43.09
3rd Qu.:5.030	3rd Qu.: 8.275	3rd Qu.:45.58
Max. :6.720	Max. :14.300	Max. :59.70

Hg	Ferr	BMI
Min. :11.60	Min. : 8.00	Min. :16.75
1st Qu.:13.50	1st Qu.: 41.25	1st Qu.:21.08
Median :14.70	Median : 65.50	Median :22.72
Mean :14.57	Mean : 76.88	Mean :22.96
3rd Qu.:15.57	3rd Qu.: 97.00	3rd Qu.:24.46
Max. :19.20	Max. :234.00	Max. :34.42

SSF	Bfat	Label	Sport
Min. : 28.00	Min. : 5.630	f-netball:23	row :37
1st Qu.: 43.85	1st Qu.: 8.545	f-row :22	t_400m :29
Median : 58.60	Median :11.650	m-t_400m :18	b_ball :25
Mean : 69.02	Mean :13.507	m-w_polo :17	netball:23
3rd Qu.: 90.35	3rd Qu.:18.080	m-row :15	swim :22
Max. :200.80	Max. :35.520	f-b_ball :13	field :19
		(Other) :94	(Other):47

Checking for consistency of coding of Sex

- Notice that the `Label` column includes information about the sport and whether these are men's or women's teams.
- An easy way to check for consistency in coding is to obtain the unique combinations of `Sex`, `Label` and `Sport`. The number of rows should be the number of levels of `Label`.

```
1 str(sp <- unique(subset(AIS, select=c(Sex,Label,Sport))))
```

```
'data.frame': 17 obs. of 3 variables:
 $ Sex : Factor w/ 2 levels "M","F": 2 2 2 2 2 2 2 2 1 ...
 $ Label: Factor w/ 17 levels "f-b_ball","f-field",...: 1 5 4 6 ..
 $ Sport: Factor w/ 10 levels "b_ball","field",...: 1 5 4 6 2 7 ..
```

Printing the unique combinations

```
1 print(sp, row.names=FALSE)
```

```
Sex    Label    Sport
F  f-b_ball  b_ball
F    f-row    row
F f-netball netball
F    f-swim    swim
F  f-field    field
F  f-t_400m  t_400m
F f-t_sprnt t_sprnt
F  f-tennis  tennis
F    f-gym    gym
M    m-swim    swim
M    m-row    row
M  m-b_ball  b_ball
M  m-t_400m  t_400m
M    m-field    field
M m-t_sprnt t_sprnt
M  m-w_polo  w_polo
M  m-tennis  tennis
```

2 Data from a local file

Reading from a file

- Data stored with a rectangular structure (columns are variables, rows are cases) in a text file are read with `read.table` or one of its special purpose variants, `read.delim`, for tab-separated values, or `read.csv`, for comma-separated values.
- Generally you can read files of a few thousand records or less using the default settings. You may need to change the logical argument `header` according to the structure of your file.
- On Windows it can be difficult to get the form of a file name exactly right. Use `file.choose` to bring up a “chooser” panel.
- Before trying to read a very large file you should read the manual R Data Import/Export manual to learn settings that can make `read.table` faster and less memory intensive.
- The `count.fields` function is useful in tracking down problems in reading files. See also chapter 4 of Gentleman (2008).

An example

- A data file from the Spring ‘08 Masters Exam is stored as `/p/stat/Data/MS.exam/s08/dietary_intake.txt`. A glance at the first few lines shows a whitespace-delimited data file with column headers. For convenience I copied the file to my `/tmp` directory.

```
1 str(diet <- read.table("/tmp/dietary_intake.txt",
2                        header=TRUE))
```

```
'data.frame': 60 obs. of 9 variables:
 $ ID      : int  14 14 14 19 26 31 41 68 77 77 ...
 $ sex     : int  2 2 2 2 2 1 1 2 1 1 ...
 $ age     : num  5.5 6.2 7.1 6.6 7.1 7.1 7.2 7.4 5.7 6.2 ...
 $ TEE     : int  1332 1616 1252 1421 1767 1881 1758 1537 1401 ..
 $ EER     : int  1398 1451 1539 1419 1672 1692 2068 1453 1508 ..
 $ EI_FFQ  : int  2847 2528 2164 3028 2925 1713 2929 1763 1937 ..
 $ EI_FR   : num  1478 1593 1382 1328 2034 ...
 $ VitA_FFA: num  1688 1372 1086 1583 1125 ...
 $ VitA_FR : num  528 9044 749 11371 4427 ...
```

3 Data from the web

Reading from remote files

- R supports http and ftp transfers via “connections” (naturally you must be connected to the net to use this feature).
- For functions like `read.table()` and friends you can use a URL instead of a file name. Sometimes if the URL is very long I split it into pieces and use `paste` with the optional argument `sep` set to the empty string to reconstruct the name.

- To read the `grocery_retailer.txt` file from the Stat 849 data directory I use

```
1 Data849 <- "http://www.stat.wisc.edu/~st849-1/data/"
2 str(groc <-
3     read.table(paste(Data849, "grocery_retailer.txt",
4                     sep=""), header=TRUE))
```

```
'data.frame':      52 obs. of  4 variables:
 $ Y : int  4264 4496 4317 4292 4945 4325 4110 4111 4161 4560 ...
 $ X1: int  305657 328476 317164 366745 265518 301995 269334 26..
 $ X2: num  7.17 6.2 4.61 7.02 8.61 6.88 7.23 6.27 6.49 6.37 ...
 $ X3: int  0 0 0 0 1 0 0 0 0 0 ...
```

4 Summary

Summary

- Data can be accessed as stored data sets in R package or read from local files or from files accessible on the web.
- Always check the structure of the data after reading it. Watch for categorical variables stored as numeric values. Also check for missing value codes that may have gone undetected. (The argument `na.strings` is used to specify missing value codes other than `NA`.)
- Check the data set summary to see if it is as expected.
- If a variable should have been numeric but ends up as a factor with weird levels, check for undetected missing value codes.
- Another occasional problem is the use of unmatched quote character in strings. (Frequent culprits in genomic data are 3' and 5'.)