Data input & output

Hadley Wickham

Assistant Professor / Dobelman Family Junior Chair

Department of Statistics / Rice University

June 2012

1. Working directories
2. Loading data
3. Strings and factors
4. Saving data

Working directory

Why?

All paths in R are relative to the working directory. Life is much easier when you have it correctly set.

Usually want one project per directory. (See also Rstudio's project support)

Makes code easy to move between computers.

Working directory

Terminal (linux or mac): the working directory is the directory you're in when you start R

Windows: File | Change dir.

Mac: ⌘-D

Rstudio: Tools | Change working dir...

```
# Find out what directory you’re in
getwd()
```

```
# List files in that directory
dir()
```

Your turn

Make sure your working directory is set to the location where you downloaded the files. Use `dir()` to check you're in the right place.

Loading data

1. Plain text
2. Excel
3. Other stats packages
4. Databases

http://cran.r-project.org/doc/manuals/R-data.html

Plain text

`read.delim()`: tab separated

`read.delim(sep = "|")`: | separated

`read.csv()`: comma separated

`read.fwf()`: fixed width

Tips

```
# You look at the raw contents of the file with
file.show("file")

# If you know what the missing code is, use it
read.csv("file", na.string = ".")
read.csv("file", na.string = "-99")

# Use count.fields to check the number of
# columns in each row. The following
# call uses the same default as read.csv
count.fields("file", sep = ",",
  quote = "", comment.char = "")
```

Your turn

Download the tricky files from the website. Practice using these tools to load them in.

```
read.csv("tricky-1.csv")
read.csv("tricky-2.csv", header = FALSE)
read.delim("tricky-3.csv", sep = "|")
count.fields("tricky-4.csv", sep = ",")
```

Excel

- Save as csv. (Use VBA to automate)
- `RODBC::odbcConnectExcel` http://cran.r-project.org/doc/manuals/R-data.html#RODBC (uses excel)
- `xlsx::read.xlsx` (uses java)
- `gdata::read.xls` (uses perl)

Excel

This is what I always do

- Save as csv. (Use VBA to automate)
- `RODBC::odbcConnectExcel` http://cran.r-project.org/doc/manuals/R-data.html#RODBC (uses excel)
- `xlsx::read.xlsx` (uses java)
- `gdata::read.xls` (uses perl)

Strings & factors

	Possible values	Order
Character	Anything	Alphabetical
Factor	Fixed and finite	Fixed, but arbitrary (default is alphabetical)
Ordered factor		Fixed and meaningful

Quiz

Take one minute to decide which data type is most appropriate for each of the following variables collected in a medical experiment:

Subject id, name, treatment, sex, number of siblings, address, race, eye colour, birth city, birth state.

Factors

- R's way of storing categorical data
- Have ordered `levels()` which:
  - Control order on plots and in `table()`
  - Are preserved across subsets
  - Affect contrasts in linear models

Ordered factors

- Imply that there is an intrinsic ordering the levels
- Don’t affect anything we’re interested in, so not very important
- In the diamonds dataset, `cut`, `color` and `clarity` are ordered factors

```
# By default, strings converted to factors when
# loading data frames. I think this is the wrong
# default - you should always explicitly convert
# strings to factors. Use stringsAsFactors = F to
# avoid this.
```

```
# For one data frame:
read.csv("myfile.csv", stringsAsFactors = F)

# For entire session:
options(stringsAsFactors = F)
```

```r
# Creating a factor
x <- sample(5, 20, rep = T)
a <- factor(x)
b <- factor(x, levels = 1:10)
c <- factor(x, labels = letters[1:5])

levels(a); levels(b); levels(c)
table(a); table(b); table(c)
```

```
# Subsets: by default levels are preserved
b2 <- b[1:5]
levels(b2)
table(b2)

# Remove extra levels
b2[, drop = TRUE]
factor(b2)

# But usually better to convert to character
b3 <- as.character(b)
table(b3)
table(b3[1:5])
```

```
# Be careful when converting factors to numbers!

x <- sample(5, 20, rep = T)
d <- factor(x, labels = 2^(1:5))
as.numeric(d)
as.character(d)
as.numeric(as.character(d))
```

Saving data

Your turn

Guess the name of the function you might use to write an R object back to a csv file on disk. Use it to save `diamonds` to `diamonds-2.csv`.

What happens if you now read in `diamonds-2.csv`? Is it different to your `diamonds` data frame? How?

```
write.csv(diamonds, "diamonds-2.csv")
diamonds2 <- read.csv("diamonds-2.csv")

head(diamonds)
head(diamonds2)

str(diamonds)
str(diamonds2)

# Better, but still loses factor levels
write.csv(diamonds, file = "diamonds-3.csv",
  row.names = F)
diamonds3 <- read.csv("diamonds-3.csv")
```

Saving data

```
# For long-term storage
write.csv(diamonds, file = "diamonds.csv",
  row.names = FALSE)

# For short-term caching
# Preserves factors etc.
saveRDS(diamonds, "diamonds.rds")
diamonds4 <- readRDS("diamonds.rds")
```

.csv	.rds
`read.csv()`	`readRDS()`
`write.csv(row.names = FALSE)`	`saveRDS()`
Only data frames	Any R object
Can be read by any program	Only by R
Long term storage	Short term caching of expensive computations

```
# Easy to store compressed files to save space:
write.csv(diamonds, file = bzfile("diamonds.csv.bz2"),
  row.names = FALSE)

# Reading is even easier:
diamonds5 <- read.csv("diamonds.csv.bz2")

# Files stored with saveRDS() are automatically
# compressed.
```