Stat405

More about data

Hadley Wickham

1. (Data update + announcement)
2. Motivating problem
3. External data
4. Strings and factors
5. Saving data

Slot machines

State regulated payoffs: how can they be sure casinos are honest?

Where are we going?

In the next few weeks we will focus our attention on some slot machine data. We want to figure out if the slot machine is paying out at the rate the manufacturer claims.

To do this, we'll need to learn more about data formats and how to write functions.

External 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

```
# 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.

(Remember to change your working directory!)

```r
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)

Data cleaning

Cleaning

I cleaned up `slots.csv` for you to practice with. The original data was `slots.txt`. The challenge today is to perform the cleaning yourself.

This should always be the first step in an analysis: ensure that your data is available as a clean csv file. Do this in once in a file called `0-clean.r`.

Your turn

Take two minutes to find as many differences as possible between `slots.txt` and `slots.csv`.

Hint: use File | Open in Rstudio to open a plain text version. Don’t use word or excel.

What did I do to clean up the file?

Cleaning

- Convert from space delimited to csv
- Add variable names
- Convert uninformative numbers to informative labels

Variable names

```
names(slots)
names(slots) <- c(”w1”, ”w2”, ”w3”,
  ”prize”, ”night”)
dput(names(slots))

# This is a very common pattern
```

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
- But, don’t affect anything we’re interested in, so I usually don’t use them
- In the diamonds dataset, cut, color and clarity should be 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("mpg.csv.bz2", 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)
```

Your turn

Convert w1, w2 and w3 to factors with labels from adjacent table

Rearrange levels in terms of value: DD, 7, BBB, BB, B, C, 0

0	Blank (0)
1	Single Bar (B)
2	Double Bar (BB)
3	Triple Bar (BBB)
5	Double Diamond (DD)
6	Cherries (C)
7	Seven (7)

```
slots <- read.delim("slots.txt", sep = " ", header = F,
  stringsAsFactors = F)
names(slots) <- c("w1", "w2", "w3", "prize", "night")

levels <- c(0, 1, 2, 3, 5, 6, 7)
labels <- c("0", "B", "BB", "BBB", "DD", "C", "7")

slots$w1 <- factor(slots$w1, levels = levels, labels = labels)
slots$w2 <- factor(slots$w2, levels = levels, labels = labels)
slots$w3 <- factor(slots$w3, levels = levels, labels = labels)
```

```r
# 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])
```

```
# Factors behave like integers when subsetting,
# not characters!

x <- c(a = ”1”, b = ”2”, c = ”3”)
y <- factor(c(”c”, ”b”, ”a”), levels = c(”c”,”b”,”a”))

x[y]
x[as.character(y)]
x[as.integer(y)]
```

```r
# 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 `slots` to `slots-2.csv`.

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

```
write.csv(slots, "slots-2.csv")
slots2 <- read.csv("slots-2.csv")

head(slots)
head(slots2)

str(slots)
str(slots2)

# Better, but still loses factor levels
write.csv(slots, file = "slots-3.csv", row.names = F)
slots3 <- read.csv("slots-3.csv")
```

Saving data

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

# For short-term caching
# Preserves factors etc.
saveRDS(slots, "slots.rds")
slots2 <- readRDS("slots.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(slots, file = bzfile(”slots.csv.bz2”),
  row.names = FALSE)

# Reading is even easier:
slots4 <- read.csv(”slots.csv.bz2”)

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