

Learning *R*

Carl James Schwarz

StatMathComp Consulting by Schwarz
cschwarz.stat.sfu.ca @ gmail.com

TESTING

1. Reading data

Reading data with R

R is fairly flexible.

- *.csv files - easiest
- Excel spreadsheets directly
- tables with white space delimiters
- Reading tables from URLs
- Internal data
- Querying most database systems (not part of this course)
- Scraping web pages (not part of this course)
- Fixing variable names to be valid *R* names.

Dealing with Dates and Times is always a pain.

R often converts character data to factors (a pain)

Simple format in text format

- observations in rows; variables in columns
- separate values by a comma; enclose values in quotes if contain a comma
- variable names in first row
- Excel and most database packages can generate

```
1 cereal <- read.csv('../..'/SampleData/cereal.csv',  
2                   header=TRUE, as.is=TRUE, strip.white=TRUE)
```

- Data is “disconnected” from database
- *as.is=TRUE* stops automatic conversion - especially true for date/times.
- *strip.white=TRUE* removes extra white space at front/end of values
- Lots of options (see help page)

```
> head(cereal)
```

	name	mfr	type	calories	protein	fat	...
1	100%_Bran	N	C	60	4	1	...
2	100%_Natural_Bran	Q	C	110	3	5	...

```
'data.frame': 77 obs. of 15 variables:
```

```
$ name      : chr  "100%_Bran" "100%_Natural_Bran" "All-Bran" ...
$mfr       : chr  "N" "Q" "K" "K" ...
$type      : chr  "C" "C" "C" "C" ...
$ calories : int  60 110 80 50 110 110 110 140 90 90 ...
$ protein  : int  4 3 4 4 2 2 2 3 2 3 ...
```

Notice that NO factors created.

Many packages to read Excel workbooks

Two most popular are:

- *xlsx* - requires java to be installed and working (!)
- *readxl* - much easier to use (recommended)

Many other packages around with varying degree of flexibility and speed.

```
1 library(readxl)
2 cereal2 <- readxl::read_excel('file.path("ALLOfDATA.xls"),
3                               sheet='cereal',
4                               skip=7,
5                               .name_repair="universal") # fixes illegal
6 head(cereal2)
7 str(cereal2)
```

- `.name_repair="universal"` - fixed illegal names
- Can specify rows/columns/cell ranges to read.
- Be careful with dates and times.


```
> head(cereal2)
```

```
              Name Manufacturer Mfr Hot.C
1          100% Bran      Nabisco   N
2 100% Nat. Bran Oats & Honey Quaker Oats  Q
```

```
> str(cereal2)
```

```
'data.frame': 76 obs. of 18 variables:
 $ Name          : chr  "100% Bran" "100% Nat. Bran Oats &
 $ Manufacturer  : chr  "Nabisco" "Quaker Oats" "Quaker Oat
 $ Mfr           : chr  "N" "Q" "Q" "K" ...
 $ Hot.Cold      : chr  "C" "C" "C" "C" ...
 $ Calories      : num  80 230 210 80 50 210 120 120 250 20
```

Notice that NO factors created.

```
1 library(readxl)
2 cereal3 <- readxl::read_excel(file.path('ALLOfDATA.xls'),
3                               sheet='cereal',
4                               skip=7,
5                               .name_repair="universal")
6 head(cereal3)
7 str(cereal3)
```

- `col_types` is automatically set to *"guess"* which works most of the time.
- Be careful of dates and time.

Reading data - Excel workbooks

```
> head(cereal3)
```

```
# A tibble: 6 x 18
```

```
      Name Manufacturer Mfr 'Hot/Cold'
  <chr>          <chr> <chr> <chr>
1 100% Bran      Nabisco      N      C
2 100% Nat. Bran Oats & Honey Quaker Oats  Q      C
```

```
> str(cereal3)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame': 76 obs. of 18 variables:
 $ Name           : chr  "100% Bran" "100% Nat. Bran Oats & Honey" "100% Nat. Bran Oats & Honey" "100% Nat. Bran Oats & Honey" ...
 $ Manufacturer   : chr  "Nabisco" "Quaker Oats" "Quaker Oats" "Quaker Oats" ...
 $ Mfr           : chr  "N" "Q" "Q" "K" ...
 $ Hot/Cold      : chr  "C" "C" "C" "C" ...
 $ Calories      : num  80 230 210 80 50 210 120 120 250 200 ...
```

Notice that NO factors created.

Notice class of object is a *tibble* as well as a data frame.

tibble vs. *data.frame*

- *tibbles* created by H. Wickham as a replacement for data frames
- Most interchangeable except for *print()* and subsetting

See *help(package=tibble)* and vignettes for more details

Reading data - tibbles vs. data.frames I

tibble vs. data.frame

```
> # These are mostly interchangeable except for print() metho
> # single columns.
> # see help(package=tibble) and vignettes for more details
> df1 <- data.frame(v1=c("a", "b"), v2=c(1,2), stringsAsFacto
> tib1 <- tibble::tibble      (v1=c("a", "b"), v2=c(1,2)) # no
>
> # compare the output from
> df1
  v1 v2
1  a  1
2  b  2

> tib1
# A tibble: 2 x 2
```

Reading data - tibbles vs. data.frames II

```
  v1      v2
<chr> <dbl>
1 a         1
2 b         2
```

```
>
```

```
> # compare the output from
```

```
> df1$xx
```

```
NULL
```

```
> tib1$xx
```

```
NULL
```

```
Warning message:
```

```
Unknown or uninitialised column: 'xx'.
```

```
>
```

Reading data - tibbles vs. data.frames III

```
> # compare the output from
> df1 [,"v1"]
[1] "a" "b"
> tib1[,"v1"]
# A tibble: 2 x 1
  v1
  <chr>
1 a
2 b
> # first is a vector; second is a tibble with 1 columns
>
> # some legacy code gets upset with the latter behaviour
> # you can force a tibble to be a data frame using
> df2 <- as.data.frame(tib1)
```

White space delimited data.

- similar to csv files
- careful with values that contain white space

Reading data - White space delimited

```
1 cereal4 <- read.table("http://lib.stat.cmu.edu/datasets/1993
2                       header=FALSE, as.is=TRUE, strip.white=
3 names(cereal4) <- c('Name', 'mfr', 'type', 'Calories', 'protein
4                       'sugars', 'shelf', 'potass', 'vitamins', 'wei
5 head(cereal4)
6 str(cereal4)
```

- Notice that I specified a URL
- Notice how column names are specified if data does not contain them in first row

Reading data - White space delimited

```
> head(cereal4)
```

	Name	mfr	type	Calories	protein	Fat	s
1	100%_Bran	N	C	70	4	1	
2	100%_Natural_Bran	Q	C	120	3	5	

```
> str(cereal4)
```

```
'data.frame': 77 obs. of 15 variables:
```

```
$ Name      : chr  "100%_Bran" "100%_Natural_Bran" "All-Bran"  
$ mfr       : chr  "N" "Q" "K" "K" ...  
$ type      : chr  "C" "C" "C" "C" ...  
$ Calories: int  70 120 70 50 110 110 110 130 90 90 ...
```

Data used underscores to prevent breaking values at white space.

Often require small amounts of data that should be stored with the script.

- *textConnection()* function useful.
- similar to reading *.csv file.

Reading data - Internal data

```
1 type.code.csv <- textConnection("
2 type, code
3 C , Cold Cereal
4 H , Hot Cereal ")
5
6 type.code <- read.csv(type.code.csv, header=TRUE,
7   strip.white=FALSE, as.is=TRUE)
8 head(type.code)
9 str(type.code)
10 type.code$type == "C"
```

- Can only read it “once” without redefining it.
- Connection name is arbitrary, but I adopt a simple convention.
- Notice that connection name NOT in quotes in *read.csv()*
- Notice how column names are specified if data does not contain them in first row

Reading data - Internal data

```
> head(type.code)
  type      code
1  C    Cold Cereal
2  H    Hot Cereal

> str(type.code)
'data.frame': 2 obs. of  2 variables:
 $ type: chr  "C " "H "
 $ code: chr  " Cold Cereal" " Hot Cereal  "

> type.code$type == "C"
[1] FALSE FALSE
```

CAUTION: Notice extra white space around variable values.

Remove extra white space in variable values!!

```
1 type.code.csv <- textConnection("
2 type, code
3 C , 'Cold Cereal'
4 H , 'Hot Cereal' ")
5
6 type.code <- read.csv(type.code.csv, header=TRUE,
7   strip.white=TRUE, as.is=TRUE)
8 head(type.code)
9 str(type.code)
10 type.code$type == "C"
```

Reading data - Internal data

```
> head(type.code)
  type      code
1    C Cold Cereal
2    H  Hot Cereal

> str(type.code)
'data.frame': 2 obs. of  2 variables:
 $ type: chr  "C" "H"
 $ code: chr  "Cold Cereal" "Hot Cereal"

> type.code$type == "C"
[1]  TRUE FALSE
```

Notice extra white space around variable values has been removed.

It is sometime necessary to adjust variable names after reading

- Variable name has an misspelling
- Variable name is not a valid *R* variable name
 - Must start with a letter
 - Contain letters, numbers, periods (.), underscores (_), but not blanks or other characters

CAUTION: Some functions do automatic “correction” of variable names and others do not.

See *make.names()* for more details.

Reading data - Adjusting variable names

```
1 sample.csv <- textConnection("
2 Bird #, Wiegth, Length mm, Mass (g)
3 1, 100, 101, 102
4 2, 200, 201, 202")
5
6 sample <- read.csv(sample.csv, header=TRUE,
7                   strip.white=TRUE, as.is=TRUE)
8 head(sample)
9 str(sample)
10 sample$Bird..
```

Reading data - Adjusting variable names

```
> head(sample)
  Bird.. Wieght Length.mm Mass..g.
1      1   100        101    102
2      2   200        201    202

> str(sample)
'data.frame': 2 obs. of  4 variables:
 $ Bird..   : int  1 2
 $ Wieght   : int 100 200
 $ Length.mm: int  101 201
 $ Mass..g. : int  102 202

> sample$Bird..
[1] 1 2
```

Notice how variable names are converted to valid *R* names.

Reading data - Adjusting variable names

```
1 sample.csv <- textConnection("
2 Bird #, Wieght, Length mm, Mass (g)
3 1, 100, 101, 102
4 2, 200, 201, 202")
5 sample <- read.csv(sample.csv, header=TRUE,
6                   strip.white=TRUE, as.is=TRUE,
7                   check.names=FALSE)
8 head(sample)
9 str(sample)
10 sample$Bird..
11 sample$"Bird #"
```

Reading data - Internal data

```
> head(sample)
  Bird # Wieght Length mm Mass (g)
1      1   100     101     102

> str(sample)
'data.frame': 2 obs. of 4 variables:
 $ Bird #    : int  1 2
 $ Wieght    : int 100 200
 $ Length mm: int 101 201

> sample$Bird..
NULL

> sample$"Bird #"
[1] 1 2
```

It is awkward (and sometime very difficult) to deal with irregular variable names.

The `names()` function allows you access to variable names.

```
1 sample2 <- sample
2 names(sample2)
3 names(sample2) <- c("Bird","Weight","Length","Mass")
4 head(sample2)
```

```
> head(sample2)
  Bird Weight Length Mass
1    1    100    101  102
2    2    200    201  202
```

The `names()` function allows you access to variable names.

```
1 sample2 <- sample
2 names(sample2)
3 names(sample2) <- c("Bird", "Weight", "Length", "Mass")
4 head(sample2)
```

```
> head(sample2)
  Bird Weight Length Mass
1    1    100    101  102
2    2    200    201  202
```

Reading data - Adjusting variable names

The `names()` function allows you access to variable names.
Selective changing of names:

```
1 sample2 <- sample
2 names(sample2)
3 names(sample2)[2] <- c("Weight")
4 head(sample2)
```

```
> names(sample2)
[1] "Bird #"      "Wieght"      "Length mm"  "Mass (g)"
```

```
> names(sample2)[2] <- c("Weight")
```

```
> head(sample2)
  Bird # Weight Length mm Mass (g)
1     1   100     101    102
2     2   200     201    202
```

The `names()` function allows you access to variable names.
Selective changing of names that is more robust

```
1 sample2 <- sample
2 names(sample2)
3
4 select <- grepl("Wieght", names(sample2))
5 select
6 sum(select)
7 names(sample2)[select]
8
9 names(sample2)[select] <- c("Weight")
10 head(sample2)
```


Reading data - Adjusting variable names

The `names()` function allows you access to variable names.

Selective changing of names that is more robust

```
> names(sample2)
[1] "Bird #"      "Wieght"      "Length mm"  "Mass (g)"
>
> select <- grepl("Wieght", names(sample2))
> select
[1] FALSE  TRUE FALSE FALSE
> sum(select)
[1] 1
> names(sample2)[select]
[1] "Wieght"
>
> names(sample2)[select] <- c("Weight")
> head(sample2)
  Bird # Weight Length mm Mass (g)
1     1    100     101    102
2     2    200     201    202
```

Consider the Birds 'n Butts dataset.

- Save the *Correlational* worksheet as *csv* and read it.
- Read the *Correlational* worksheet directly.
- Change the error in the variable name.

Reading data - Exercise

```
1 library(readxl)
2 butts <- read_excel(file.path('bird-butts-data.xlsx'), sheet = 'Data',
3                     col_names=TRUE, skip=1)
4 butts[1:5,]
5 dim(butts)
6 str(butts)
7
8 # Or, save the sheet from the Excel file and read the csv file
9 butts <- read.csv("../sampledata/bird-butts-data-correlation.csv")
10 butts[1:5,]
11 dim(butts)
12 str(butts)
13
14 # Fix the names
15 select <- grepl('wiegth', names(butts))
16 select
17 sum(select)
18 names(butts)[select]
19
20 names(butts)[ select] <- "Butts.weight"
```

Changing the variable name:

```
> select <- grepl('wieght', names(butts))
> select
[1] FALSE FALSE FALSE  TRUE FALSE
> sum(select)
[1] 1
> names(butts)[select]
[1] "Butts.wieght"
>
> names(butts)[ select] <- "Butts.weight"
> butts[1:5,]
  Nest Species Nest.content Butts.weight Number.of.mites
1     1   HOSP         empty          6.13                4
2     2   HOSP         empty          3.73               30
```

Fairly rich set of functions to read data. Most common is to read rectangular structure into a data frame.

- `read.csv()` is easiest followed by reading Excel sheet directly.
- Able to access data bases as well - see *R* manuals.
- Use `textConnection()` for small tables so that data kept with script.
- CAUTION: Extra white space around variable values.
- CAUTION: Do NOT let *R* convert strings to factors.
- CAUTION: Dates and times
- CAUTION: Non-standard variable names.