Learning *R*

Carl James Schwarz

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

Reading data in *R*

Table of Contents I

Reading data with *R*

Reading data

R is fairly flexible.

- **.csv* files - easiest
- Excel spreadsheets directly
- tables with white space deliminters
- 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)

Reading data - csv files

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

```
cereal <- read.csv('../../SampleData/cereal.csv',
                   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)

Reading data - csv files

```
> head(cereal)
                      name mfr type calories protein fat so
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.

Reading data - Excel workbooks

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.

Reading data - Excel workbooks

```
library(readxl)
cereal2 <- readxl::read_excel('file.path("ALLofDATA.xls"),
                          sheet='cereal',
                          skip=7,
                          .name_repair="universal") # fixes illeg
head(cereal2)
str(cereal2)
```

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

Reading data - Excel workbooks

```
> 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.

Reading data - Excel workbooks

```r
library(readxl)
cereal3 <- readxl::read_excel(file.path('ALLofDATA.xls'),
                             sheet='cereal',
                             skip=7,
                             .name_repair="universal")
head(cereal3)
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 ‘Ho
                                <chr>        <chr> <chr>
1                           100% Bran      Nabisco     N
2          100% Nat. Bran Oats & Honey  Quaker Oats     Q

> str(cereal3)
Classes ’tbl_df’, ’tbl’ and ’data.frame’: 76 obs. of  18 var
 $ 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.
Notice class of object is a *tibble* as well as a data frame.

Reading data - tibbles vs. data.frames

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() meth
> # single columns.
> # see help(package=tibble) and vignettes for more details
> df1  <- data.frame(v1=c("a", "b"), v2=c(1,2), stringsAsFac
> 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)
```

Reading data - White space delmited

White space delimited data.

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

Reading data - White space delmited

```
cereal4 <- read.table("http://lib.stat.cmu.edu/datasets/199
                      header=FALSE, as.is=TRUE, strip.white=
names(cereal4) <- c('Name','mfr','type','Calories','protein
                   'sugars','shelf','potass','vitamins','we
head(cereal4)
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 delmited

```
> head(cereal4)
                      Name mfr type Calories protein Fat so
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.

Reading data - Internal data

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

```
type.code.csv <- textConnection("
type, code
C , Cold Cereal
H , Hot Cereal  ")

type.code <- read.csv(type.code.csv, header=TRUE,
      strip.white=FALSE, as.is=TRUE)
head(type.code)
str(type.code)
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.

Reading data - Internal data

Remove extra white space in variable values!!

```
type.code.csv <- textConnection("
type, code
C , 'Cold Cereal'
H , 'Hot Cereal'  ")

type.code <- read.csv(type.code.csv, header=TRUE,
     strip.white=TRUE, as.is=TRUE)
head(type.code)
str(type.code)
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.

Reading data - Adjusting variable names

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

```
sample.csv <- textConnection("
Bird #, Wieght, Length mm, Mass (g)
1, 100, 101, 102
2, 200, 201, 202")

sample <- read.csv(sample.csv, header=TRUE,
                   strip.white=TRUE, as.is=TRUE)
head(sample)
str(sample)
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

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

Reading data - Adjusting variable names

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

```
sample2 <- sample
names(sample2)
names(sample2) <- c("Bird","Weight","Length","Mass")
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.

```r
sample2 <- sample
names(sample2)
names(sample2) <- c("Bird","Weight","Length","Mass")
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:

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

Reading data - Adjusting variable names

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

```
sample2 <- sample
names(sample2)

select <- grepl("Wieght", names(sample2))
select
sum(select)
names(sample2)[select]

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

Reading data - Adjusting variable names I

The *plyr::rename()* function is useful.

```
> sample3 <- sample
> sample3
  Bird # Wieght Length mm Mass (g)
1      1    100       101      102
2      2    200       201      202

> # you can renamesall or selected columns
> sample3 <- sample
> sample3 <- plyr::rename(sample3,
+                         c("Bird #"="Bird",
+                           "Wieght"="Weight",
+                           "Length mm"="Length",
+                           "Mass (g)"="Mass"))
```

Reading data - Adjusting variable names II

```
> head(sample3)
  Bird Weight Length Mass
1    1    100    101  102
2    2    200    201  202
```

Reading data - Exercise

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

```
library(readxl)
butts <- read_excel(file.path('bird-butts-data.xlsx'), sheet
                    col_names=TRUE, skip=1)
butts[1:5,]
dim(butts)
str(butts)

# Or, save the sheet from the Excel file and read the csv f
butts <- read.csv("../sampledata/bird-butts-data-correlatio
butts[1:5,]
dim(butts)
str(butts)

# Fix the names
select <-  grepl('wieght', names(butts))
select
sum(select)
names(butts)[select]

names(butts)[ select] <- "Butts.weight"
```

Reading data - Exercise

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
```

Reading data - Summary

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

- *read.csv()* is easist 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.