Handling Data in R

readr, dplyr, tidyr, haven

Marcel Ramos
CUNY School of Public Health
Hunter College

Department of Biostatistics

November 13, 2015

Introduction

- Tips/Resources for learning R
- Installing packages
- Importing in data
    - readr, haven
- Manipulating data
    - dplyr, tidyr
- Mathematical Notation

Tips for learning R (general)

- Learning R may become frustrating at times
- Learning a language
- It's a matter of practice

Useful tips for learning R (stand-alone)

Pseudo code	Example code
install.packages(packagename)	install.packages("dplyr")
library(packagename)	library(dplyr)
?functionname	?select
?package::functionname	?dplyr::select
? 'Reserved keyword or symbol' (or backticks)	? '%>%'
??searchforpossiblyexistingfunctionandortopic	??simulate
help(package = "loadedpackage")	help("dplyr")
browseVignettes("packagename")	browseVignettes("dplyr")

Learning R via online courses

- Coursera
- edX
- RStudio tutorials
- Quick-R - Mostly for basic and base functions
- RStudio Cheatcheets

Installing packages

- Depends on source of package
- GitHub, CRAN, Bioconductor
- Packages
  - `utils`
  - `devtools`
  - `BiocInstaller`

Functions for installing packages

```
utils::install.packages("packagename")
devtools::install_github("githubuser/repository")
```

Bioconductor Packages

- See the Bioconductor site for more info

Pseudo code:

```
source("https://bioconductor.org/biocLite.R")
packages <- c("packagename", "githubuser/repository", "bioconductorp
BiocInstaller::biocLite(packages)
```

- Works for CRAN, GitHub, and Bioconductor packages!

Note about installing `devtools`

- Useful for building packages
- Download and install from GitHub
- Installation dependent on OS (Rtools for Windows)

Functions for handling files

- File management

```
dir()
file.path()
list.files()
untar(); unzip()
```

Using R for data importing and manipulation

```
BiocInstaller::biocLite("hadley/readr")
browseVignettes("readr")
```

- Fast way to read in large files

Available functions in `readr`

- `read_delim()`
- `read_tsv()`
- `read_fwf()`
- `read_table()`
- `read_lines()`
- `read_file()`

Demo

- session1script.Rmd

Results are in...

1.33 Gb file

Function	Elapsed Time (seconds)
utils::read.csv	$\sim$ 137.83
readr::read_csv	$\sim$ 38.99

Using data.table

- Consider using `data.table::fread()` for max performance
- data.table syntax although data.table = FALSE

```
library(data.table)
?fread
```

The `haven` *package*

- Great for reading in foreign data formats
- SAS, SPSS, Stata
- sas7bdat

```
devtools::install_github("hadley/haven")
library(haven)
?read_sas
?read_sav
?read_dta
```

Principles of Tidy Data

- Often said: 80% of data analysis is cleaning/munging
- Provide a standard way of organizing data[1]

1. Each variable forms a column
2. Each observation forms a row
3. Each type of observational unit forms a table

Dataset	Variable	Variable
Observation	Value	Value
Observation	Value	Value

[1] http://vita.had.co.nz/papers/tidy-data.pdf

Principles of Tidy Data (2)

- Why is tidy data important?
- Easier for the analyst and the computer to extract knowledge from a set of values
- Saves a *lot* of time

Data Munging using `tidyr`

- `tidyr` faciliates reshaping of data

1. spread vs. gather *most likely to use
2. extract/separate vs. unite
3. nest vs. unnest

Data Manipulation using `dplyr`

- `dplyr` convention aims to ease cognitive burden
- Function names are easy to remember

1. select (Y)
2. mutate/transmute (add Ys / new Y)
3. filter (get Xs based on condition)
4. slice (get Xs specified)
5. summarise (reduce to single observation)
6. arrange (re-order observations)

The `tbl_df` class and show method

- Data frame print is messy
- `tbl_df` provides same functionality (i.e. data.frame methods work)
- Output is neat and descriptive
- See: `?tbl_df`

```
library(dplyr)
tbl_df(mtcars)
```

```
## Source: local data frame [32 x 11]
##
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear   c
##    (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (c
## 1   21.0     6 160.0   110  3.90 2.620 16.46     0     1     4
## 2   21.0     6 160.0   110  3.90 2.875 17.02     0     1     4
## 3   22.8     4 108.0    93  3.85 2.320 18.61     1     1     4
## 4   21.4     6 258.0   110  3.08 3.215 19.44     1     0     3
## 5   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3
## 6   18.1     6 225.0   105  2.76 3.460 20.22     1     0     3
## 7   14.3     8 360.0   245  3.21 3.570 15.84     0     0     3
```

Examples of use

- Create an example of messy data:

```
library(tidyr)
data("mtcars")
mtcars <- tbl_df(mtcars)
mtcars <- select(mtcars, c(mpg:hp, wt, vs:carb))
mtcars <- unite(mtcars, cylgear, cyl, gear)
separate(mtcars, cylgear, c("cyl0", "gear0"))
```

```
## Source: local data frame [32 x 9]
##
##      mpg  cyl0 gear0  disp    hp    wt    vs    am  carb
##    (dbl) (chr) (chr) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl)
## 1   21.0     6     4 160.0   110 2.620     0     1     4
## 2   21.0     6     4 160.0   110 2.875     0     1     4
## 3   22.8     4     4 108.0    93 2.320     1     1     1
## 4   21.4     6     3 258.0   110 3.215     1     0     1
## 5   18.7     8     3 360.0   175 3.440     0     0     2
## 6   18.1     6     3 225.0   105 3.460     1     0     1
```

Mutate & Transumte

```
head(mutate(mtcars, displ_l = disp/61.0237), 2)
```

```
## Source: local data frame [2 x 9]
##
##     mpg cylgear  disp    hp    wt    vs    am  carb  displ_l
##   (dbl)   (chr) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl)    (dbl)
## 1    21     6_4   160   110 2.620     0     1     4 2.621932
## 2    21     6_4   160   110 2.875     0     1     4 2.621932
```

```
head(transmute(mtcars, disp_l = disp/61.0237),2)
```

```
## Source: local data frame [2 x 1]
##
##     disp_l
##      (dbl)
## 1 2.621932
## 2 2.621932
```

Example with base functions

```
data("mtcars")
mtcars <- mtcars[,c("mpg", "cyl", "disp", "hp",
                    "wt", "vs", "am", "gear", "carb")]
mtcars$cylgear <- with(mtcars, paste(cyl, gear, sep = "."))
mtcars[, c("cyl1", "gear1")] <- NA
mtcars[, c("cyl1", "gear1")] <-
  t(sapply(strsplit(mtcars$cylgear, ".", fixed = TRUE), FUN = "[", c
head(mtcars, 3)
```

```
##                mpg cyl disp  hp    wt vs am gear carb cylgear cyl
## Mazda RX4     21.0   6  160 110 2.620  0  1    4    4     6.4
## Mazda RX4 Wag 21.0   6  160 110 2.875  0  1    4    4     6.4
## Datsun 710    22.8   4  108  93 2.320  1  1    4    1     4.4
```

Considerations

Be careful of loss of information!

- Row names were lost when converting to `table_df`
- Solution: add rownames as variable

```
data(mtcars)
carrows <- rownames(mtcars)
mtcars <- tbl_df(mtcars)
mtcars <- mutate(mtcars, models = carrows)
```

Functional programming example

```
hourly_delay <- filter(
  summarise(
    group_by(
      filter(
        flights,
        !is.na(dep_delay)
      ),
      date, hour
    ),
    delay = mean(dep_delay),
    n = n()
  ),
  n > 10
)
```

Pipes for fluid and readable programming

- Piping operator: %>%
- Consider the previous example with pipes:

```
hourly_delay <- flights %>%
  filter(!is.na(dep_delay)) %>%
  group_by(date, hour) %>%
  summarise(delay = mean(dep_delay), n = n()) %>%
  filter(n > 10)
```

More piping

```
library(nycflights13)
flights %>% group_by(carrier) %>%
  summarise(avg_depdelay = mean(dep_delay, na.rm = TRUE),
            count = n()) %>% left_join(airlines) %>%
    arrange(avg_depdelay) %>% head
```

```
## Source: local data frame [6 x 4]
##
##   carrier avg_depdelay count                       name
##     (chr)        (dbl) (int)                     (fctr)
## 1      US     3.782418 20536        US Airways Inc.
## 2      HA     4.900585   342 Hawaiian Airlines Inc.
## 3      AS     5.804775   714   Alaska Airlines Inc.
## 4      AA     8.586016 32729 American Airlines Inc.
## 5      DL     9.264505 48110   Delta Air Lines Inc.
## 6      MQ    10.552041 26397              Envoy Air
```

Using separate

```
data(iris)
longdata <- gather(tbl_df(iris), key = measure, n,
  Sepal.Length:Petal.Width) %>%  separate(measure, c("type",
    "dimension"))
longdata %>% group_by(Species, type, dimension) %>%
  summarise(avg_dim = mean(n, na.rm = TRUE))
```

```
## Source: local data frame [12 x 4]
## Groups: Species, type [?]
##
##       Species  type dimension avg_dim
##        (fctr) (chr)     (chr)   (dbl)
## 1      setosa Petal    Length   1.462
## 2      setosa Petal     Width   0.246
## 3      setosa Sepal    Length   5.006
## 4      setosa Sepal     Width   3.428
## 5  versicolor Petal    Length   4.260
## 6  versicolor Petal     Width   1.326
```

Piping with `tidyr`

```
library(readr)
(pew <- read_csv("./data/pew.csv"))
```

```
## Source: local data frame [18 x 11]
##
##                  religion <$10k $10-20k $20-30k $30-40k $40-50k
##                     (chr) (int)   (int)   (int)   (int)   (int)
## 1                Agnostic    27      34      60      81      76
## 2                 Atheist    12      27      37      52      35
## 3                Buddhist    27      21      30      34      33
## 4                Catholic   418     617     732     670     638
## 5      Don't know/refused    15      14      15      11      10
## 6        Evangelical Prot   575     869    1064     982     881
## 7                   Hindu     1       9       7       9      11
## 8  Historically Black Prot   228     244     236     238     197
## 9       Jehovah's Witness    20      27      24      24      21
## 10                Jewish    19      19      25      25      30
## 11          Mainline Prot   289     495     619     655     651
```

Using gather

```
pew %>% gather(income, n, -religion) %>% head
```

```
## Source: local data frame [6 x 3]
##
##              religion income     n
##                 (chr) (fctr) (int)
## 1            Agnostic  <$10k    27
## 2             Atheist  <$10k    12
## 3            Buddhist  <$10k    27
## 4            Catholic  <$10k   418
## 5 Don't know/refused  <$10k    15
## 6   Evangelical Prot  <$10k   575
```

income, religion : variables to gather n : variable in cells -religion means all except religion

Using group_by

```
pew %>% gather(income, n, -religion) %>%
  group_by(income) %>% summarise(totals = sum(n))
```

```
## Source: local data frame [10 x 2]
##
##                  income totals
##                  (fctr)  (int)
## 1                  <$10k   1930
## 2                $10-20k   2781
## 3                $20-30k   3357
## 4                $30-40k   3302
## 5                $40-50k   3085
## 6                $50-75k   5185
## 7               $75-100k   3990
## 8              $100-150k   3197
## 9                  >150k   2608
## 10 Don't know/refused   6121
```

Using group_by (2)

```
pew %>% gather(income, n, -religion) %>%
  group_by(religion) %>% summarise(totals = sum(n))
```

```
## Source: local data frame [18 x 2]
##
##                   religion totals
##                      (chr)  (int)
## 1                 Agnostic    826
## 2                  Atheist    515
## 3                 Buddhist    411
## 4                 Catholic   8054
## 5       Don't know/refused    272
## 6         Evangelical Prot   9472
## 7                    Hindu    257
## 8  Historically Black Prot   1995
## 9        Jehovah's Witness    215
## 10                  Jewish    682
## 11           Mainline Prot   7470
```

P.S. Differences between integer and numeric

```
a <- 1:1000
class(a)
```

```
## [1] "integer"
```

```
object.size(a)
```

```
## 4040 bytes
```

```
object.size(as.numeric(a))
```

```
## 8040 bytes
```