

# *Making the most of R*

Marcel Ramos

October 16, 2015

# Using R for Big Data

- Big Data
- Tidy Data
- Tips for learning R
- Reading data into memory
- Cleaning and Manipulating data with `tidyr` and `dplyr`
- Pipes for fluid and readable programming

# Big Data<sup>1</sup>

Size	Description
Big	Can't fit in memory on one computer: >5 TB
Medium	Fits in memory on a server: 10 GB - 5 TB
Small	Fits in memory on a laptop: <10 GB

*Note:*

R is great at small!

<sup>1</sup>Slide adapted from Hadley Wickham

## Big Data (2)

- Reducible problems (subsetting, sampling, summarizing)
- Big data is often messy data and not much else
- Price to pay for big data

# Principles of Tidy Data

- Often said: 80% of data analysis is cleaning/munging
- Provide a standard way of organizing data<sup>2</sup>

- 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

<sup>2</sup><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

## Useful tips for learning R via Documentation

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

# *Learning R via online courses*

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



## Reading and Loading Datasets into Memory

- Requires installation of devtools package and Rtools (varies by OS)

```
devtools::install_github("username/repository")  
devtools::install_github("hadley/readr")  
devtools::install_github("hadley/haven")
```

## Read Time

```
file.info("data/BRFSS2013_Data.csv")$size/(1024^2)
system.time(read.csv("data/BRFSS2013_Data.csv"))

library(readr)
system.time(read_csv("data/BRFSS2013_Data.csv"))
```

58.8 MB File

Function	Elapsed Time
utils::read.csv	5.115
readr::read_csv	1.836

## Read Time (2)

- You may also consider the `fread` function
- `data.table` syntax is different

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

# Data Munging using *tidyr*

- *tidyr* facilitates 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)

## Examples of use

- Create an example of messy data:

```
library(dplyr); library(tidyr)
data("mtcars")
mtcars <- select(mtcars, c(mpg:hp, wt, vs:carb))
mtcars <- unite(mtcars, cylgear, cyl, gear)
separate(mtcars, cylgear, c("cyl0", "gear0"))[1:3,]
```

	mpg	cyl0	gear0	disp	hp	wt	vs	am	carb
## Mazda RX4	21.0	6	4	160	110	2.620	0	1	4
## Mazda RX4 Wag	21.0	6	4	160	110	2.875	0	1	4
## Datsun 710	22.8	4	4	108	93	2.320	1	1	1

```
mtcars <- select(mtcars, c(1:4, 5, 7:11))
```

## Mutate & Transmute

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

```
##      mpg cyl gear  disp  hp    wt  vs  am  carb  displ_l
## 1  21.0   6   4   160  110  2.620  0   1     4  2.621932
## 2  21.0   6   4   160  110  2.875  0   1     4  2.621932
## 3  22.8   4   4   108   93  2.320  1   1     1  1.769804
## 4  21.4   6   3   258  110  3.215  1   0     1  4.227866
## 5  18.7   8   3   360  175  3.440  0   0     2  5.899347
## 6  18.1   6   3   225  105  3.460  1   0     1  3.687092
```

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

```
##      displ_l
## 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 =
    head(mtcars, 3))
```

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



Be careful of loss of information!

## 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 =
    merge(., airlines) %>% arrange(avg_depdelay) %>% head
```

##	carrier	avg_depdelay	count	name
## 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(iris, key = measure, n, Sepal.Length:Petal.Length,
  separate(measure, c("type", "dimension")))
longdata %>% group_by(Species, type, dimension) %>% summarise()
```

```
## 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)
## 1      Agnostic      27      34      60      81
## 2      Atheist       12      27      37      52
## 3      Buddhist      27      21      30      34
## 4      Catholic     418     617     732     670
## 5  Don't know/refused   15      14      15      11
## 6      Evangelical Prot  575     869    1064     982
## 7           Hindu        1        9        7        9
## 8  Historically Black Prot  228     244     236     238
## 9      Jehovah's Witness   20     27     24     24
```

## 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) %>%
```

```
## 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) %>%
```

```
## 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
```

# Summary

- Big data not always the best option
- Tidy data makes everything easier and saves time
- Learning R can be a bit frustrating but certainly not impossible
- R is great for small types of datasets that fit into memory but can also be used in HPC
- Writing R code should not be a cognitive burden on the user
- R programming should be readable and fun to use!