Statistical Computing 1
Stat 590

Chapter 20
Data manipulation

Erik Erhardt

Department of Mathematics and Statistics
MSC01 1115
1 University of New Mexico
Albuquerque, New Mexico, 87131-0001
Office: MSLC 312
erike@stat.unm.edu

Fall 2015

Outline

1. Read data
2. Factors
3. Save data
4. Subset, summarise, and arrange
5. Join data
6. Split, Apply, Combine via plyr

Adapted from Hadley Wickam's
`http://stat405.had.co.nz/lectures/07-data.pdf` and
`http://stat405.had.co.nz/lectures/11-adv-data-manip.pdf`.

Read data

Read data

- plain text
- csv (comma separated values)
- Excel
- Proprietary formats from other software (stat packages)
- Databases

`https://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
```

Each of these are versions of `read.table()` with certain options prespecified.

Tips

```
# If you know what the missing (NA) 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

Tricky files

Download the tricky files from the website.

- tricky-1.csv
- tricky-2.csv
- tricky-3.csv
- tricky-4.csv

Practice using these tools to load them in.
(Remember to specify the full path or change your working directory!)

How'd you do?

```
t1 <- read.csv("tricky-1.csv")

t2 <- read.csv("tricky-2.csv", header = FALSE)

t3 <- read.delim("tricky-3.csv", sep = "|")

all.equal(t1, t2) # headers do not match
all.equal(t1, t3)
all.equal(t2, t3) # headers do not match

t4 <- count.fields("tricky-4.csv", sep = ",")
t4      # different number of fields over all rows
```

Excel

Save as csv (cleanest way).

or

```
library(gdata)
?read.xls          # (uses perl)
```

Can specify sheet number.

Cleaning data, basic

`slots.csv` is a cleaned version of `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.

Write a short script to clean the `slots.txt` file.

Your turn

slots.txt cleaning

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; they autoformat or hide details!

What was done to clean the file?

Cleaning steps

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

Variable names

```
colnames(slots)
colnames(slots) <- c("w1", "w2", "w3", "prize", "night")
```

Strings and Factors

Strings and Factors

	Possible values	Order
Character	Anything	Alphabetical
Factor	Fixed and finite	Fixed, but arbitrary (default alpha)
Ordered factor	Fixed and finite	Fixed and meaningful

Your turn

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.
- Ordering doesn't affect anything we're interested in, so don't use unless needed.
- Ordering factors will use that ordering in plots and summaries.

```
factor(df, ordered = TRUE)
```

Strings as factors — nope

```r
# By default, strings converted to factors when loading
#   data frames.
# Wrong default - explicitly convert strings to factors.
# Use stringsAsFactors = FALSE to avoid this.

# For one data frame:
read.csv("filename.csv", stringsAsFactors = FALSE)

# For entire session:
options(stringsAsFactors = FALSE)
```

Creating a factor 1

```
# Creating a factor
x <- sample(5, 20, rep = TRUE)
a <- factor(x)
b <- factor(x, levels = 1:10)
d <- factor(x, labels = letters[1:5])
x
##  [1] 1 2 2 3 5 1 4 2 2 5 1 5 1 5 1 5 2 1 2 5
a
##  [1] 1 2 2 3 5 1 4 2 2 5 1 5 1 5 1 5 2 1 2 5
## Levels: 1 2 3 4 5
b
##  [1] 1 2 2 3 5 1 4 2 2 5 1 5 1 5 1 5 2 1 2 5
## Levels: 1 2 3 4 5 6 7 8 9 10
d
##  [1] a b b c e a d b b e a e a e a e b a b e
## Levels: a b c d e
```

Creating a factor 2

Explain this behavior:

```
levels(a); levels(b); levels(d)
## [1] "1" "2" "3" "4" "5"
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
## [1] "a" "b" "c" "d" "e"
table(a); table(b); table(d)
## a
## 1 2 3 4 5
## 6 6 1 1 6
## b
##  1  2  3  4  5  6  7  8  9 10
##  6  6  1  1  6  0  0  0  0  0
## d
## a b c d e
## 6 6 1 1 6
```

Your turn

Applying Factors

1. Convert w1, w2, and w3 to factors with labels from the table.
2. Rearrange levels in terms of value: DD, 7, BBB, BB, B, C, 0.

Value	Label
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)

Applying Factors

```
slots <- read.delim("http://statacumen.com/teach/SC1/slots.txt"
                    , sep = " "
                    , header = FALSE
                    , stringsAsFactors = FALSE)
names(slots) <- c("w1", "w2", "w3", "prize", "night")
levels <- c(0, 6, 1, 2, 3, 7, 5)
labels <- c("0", "C", "B", "BB", "BBB", "7", "DD")
slots$w1 <- factor(slots$w1, levels = levels, labels = labels
                   , ordered = TRUE)
slots$w2 <- factor(slots$w2, levels = levels, labels = labels
                   , ordered = TRUE)
slots$w3 <- factor(slots$w3, levels = levels, labels = labels
                   , ordered = TRUE)
```

Applying Factors

```
str(slots)
```

```
## 'data.frame': 345 obs. of  5 variables:
##  $ w1   : Ord.factor w/ 7 levels "0"<"C"<"B"<"BB"<..: 4 1 1
##  $ w2   : Ord.factor w/ 7 levels "0"<"C"<"B"<"BB"<..: 1 7 1
##  $ w3   : Ord.factor w/ 7 levels "0"<"C"<"B"<"BB"<..: 1 3 1
##  $ prize: int  0 0 0 0 0 0 0 0 5 0 ...
##  $ night: int  1 1 1 1 1 1 1 1 1 1 ...
```

```
levels(slots$w1)
```

```
## [1] "0"   "C"   "B"   "BB"  "BBB" "7"   "DD"
```

```
summary(slots$w1)
```

```
##   0   C   B  BB BBB   7  DD
## 141   6 132  30  14  15   7
```

Factor facts 1-1

```
b
```

```
##  [1] 1 2 2 3 5 1 4 2 2 5 1 5 1 5 1 5 2 1 2 5
## Levels: 1 2 3 4 5 6 7 8 9 10
```

```
# Subsets: by default levels are preserved
b2 <- b[1:5]
b2
```

```
## [1] 1 2 2 3 5
## Levels: 1 2 3 4 5 6 7 8 9 10
```

```
levels(b2)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
```

```
table(b2)
```

```
## b2
##  1  2  3  4  5  6  7  8  9 10
##  1  2  1  0  1  0  0  0  0  0
```

Factor facts 1-2

```
# Remove extra levels
b2[, drop = TRUE]
## [1] 1 2 2 3 5
## Levels: 1 2 3 5
b2
## [1] 1 2 2 3 5
## Levels: 1 2 3 4 5 6 7 8 9 10
factor(b2)
## [1] 1 2 2 3 5
## Levels: 1 2 3 5
```

Factor facts 1-3

```
# But usually better to convert to character
b3 <- as.character(b)
b3
##  [1] "1" "2" "2" "3" "5" "1" "4" "2" "2" "5" "1" "5" "1" "5"
## [17] "2" "1" "2" "5"
table(b3)
## b3
## 1 2 3 4 5
## 6 6 1 1 6
table(b3[1:5])
##
## 1 2 3 5
## 1 2 1 1
```

Factor facts 2-1

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

x <- c(a = "1", b = "2", c = "3")
x
##   a   b   c
## "1" "2" "3"
y <- factor(c("c", "b", "a"), levels = c("c","b","a"))
y
## [1] c b a
## Levels: c b a
as.numeric(y)
## [1] 1 2 3
```

Factor facts 2-2

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

x[y]
##   a   b   c
## "1" "2" "3"
x[as.character(y)]
##   c   b   a
## "3" "2" "1"
x[as.integer(y)]
##   a   b   c
## "1" "2" "3"
```

Factor facts 3-1

```
# Be careful when converting factors to numbers!
x <- sample(5, 20, rep = TRUE)
x
##  [1] 2 2 4 4 4 2 2 2 4 1 5 5 4 3 1 5 1 4 4 2
d <- factor(x, labels = 2^(1:5))
d
##  [1] 4  4  16 16 16 4  4  4  16 2  32 32 16 8  2  32 2  16
## Levels: 2 4 8 16 32
```

Factor facts 3-2

```
# Be careful when converting factors to numbers!
as.numeric(d)
##  [1] 2 2 4 4 4 2 2 2 4 1 5 5 4 3 1 5 1 4 4 2
as.character(d)
##  [1] "4"  "4"  "16" "16" "16" "4"  "4"  "4"  "16" "2"  "32"
## [14] "8"  "2"  "32" "2"  "16" "16" "4"
as.numeric(as.character(d))
##  [1]  4  4 16 16 16  4  4  4 16  2 32 32 16  8  2 32  2 16
```

Save data

Your turn

Save slots

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?

Save slots 0

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

Save slots 1

```
head(slots)
```

```
##   w1 w2 w3 prize night
## 1 BB  0  0     0     1
## 2  0 DD  B     0     1
## 3  0  0  0     0     1
## 4 BB  0  0     0     1
## 5  0  0  0     0     1
## 6  0  0  B     0     1
```

```
head(slots2)
```

```
##   X w1 w2 w3 prize night
## 1 1 BB  0  0     0     1
## 2 2  0 DD  B     0     1
## 3 3  0  0  0     0     1
## 4 4 BB  0  0     0     1
## 5 5  0  0  0     0     1
## 6 6  0  0  B     0     1
```

Save slots 2

```
str(slots)
```

```
## 'data.frame': 345 obs. of  5 variables:
##  $ w1   : Ord.factor w/ 7 levels "0"<"C"<"B"<"BB"<..: 4 1 1
##  $ w2   : Ord.factor w/ 7 levels "0"<"C"<"B"<"BB"<..: 1 7 1
##  $ w3   : Ord.factor w/ 7 levels "0"<"C"<"B"<"BB"<..: 1 3 1
##  $ prize: int  0 0 0 0 0 0 0 0 5 0 ...
##  $ night: int  1 1 1 1 1 1 1 1 1 1 ...
```

```
str(slots2)
```

```
## 'data.frame': 345 obs. of  6 variables:
##  $ X    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ w1   : chr  "BB" "0" "0" "BB" ...
##  $ w2   : chr  "0" "DD" "0" "0" ...
##  $ w3   : chr  "0" "B" "0" "0" ...
##  $ prize: int  0 0 0 0 0 0 0 0 5 0 ...
##  $ night: int  1 1 1 1 1 1 1 1 1 1 ...
```

Save slots 3

```
# Better, but still loses factor level ordering
write.csv(slots, file = "data/slots-3.csv"
          , row.names = FALSE)
slots3 <- read.csv("data/slots-3.csv")
str(slots3)
```

```
## 'data.frame': 345 obs. of  5 variables:
##  $ w1   : chr  "BB" "0" "0" "BB" ...
##  $ w2   : chr  "0" "DD" "0" "0" ...
##  $ w3   : chr  "0" "B" "0" "0" ...
##  $ prize: int  0 0 0 0 0 0 0 0 5 0 ...
##  $ night: int  1 1 1 1 1 1 1 1 1 1 ...
```

Saving data

```r
# 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 vs .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

Saving compressed files

```
# Easy to store compressed files to save space:
write.csv(slots, file = bzfile("data/slots.csv.bz2")
          , row.names = FALSE)

file.size("data/slots.csv")
## [1] 5820
file.size("data/slots.csv.bz2")
## [1] 562
# Reading is even easier:
slots4 <- read.csv("data/slots.csv.bz2")

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

Baby names, subset

Baby names

Top 1000 male and female baby names in the US, from 1880 to 2008.
258,000 records (1000 * 2 * 129)
But only five variables: year, name, soundex, sex, and prop.

```
options(stringsAsFactors = FALSE)
# note, reading a compressed file does not work
#   from http connection, save to disk first
bnames <- read.csv("data/bnames2.csv.bz2")

births <-
  read.csv("http://statacumen.com/teach/SC1/births.csv")
```

```
head(bnames)
```

```
##   year    name     prop sex soundex
## 1 1880    John 0.081541 boy    J500
## 2 1880 William 0.080511 boy    W450
## 3 1880   James 0.050057 boy    J520
## 4 1880 Charles 0.045167 boy    C642
## 5 1880  George 0.043292 boy    G620
## 6 1880   Frank 0.027380 boy    F652
```

```
tail(bnames)
```

```
##        year     name     prop  sex soundex
## 257995 2008     Diya 0.000128 girl    D000
## 257996 2008 Carleigh 0.000128 girl    C642
## 257997 2008    Iyana 0.000128 girl    I500
## 257998 2008   Kenley 0.000127 girl    K540
## 257999 2008   Sloane 0.000127 girl    S450
## 258000 2008  Elianna 0.000127 girl    E450
```

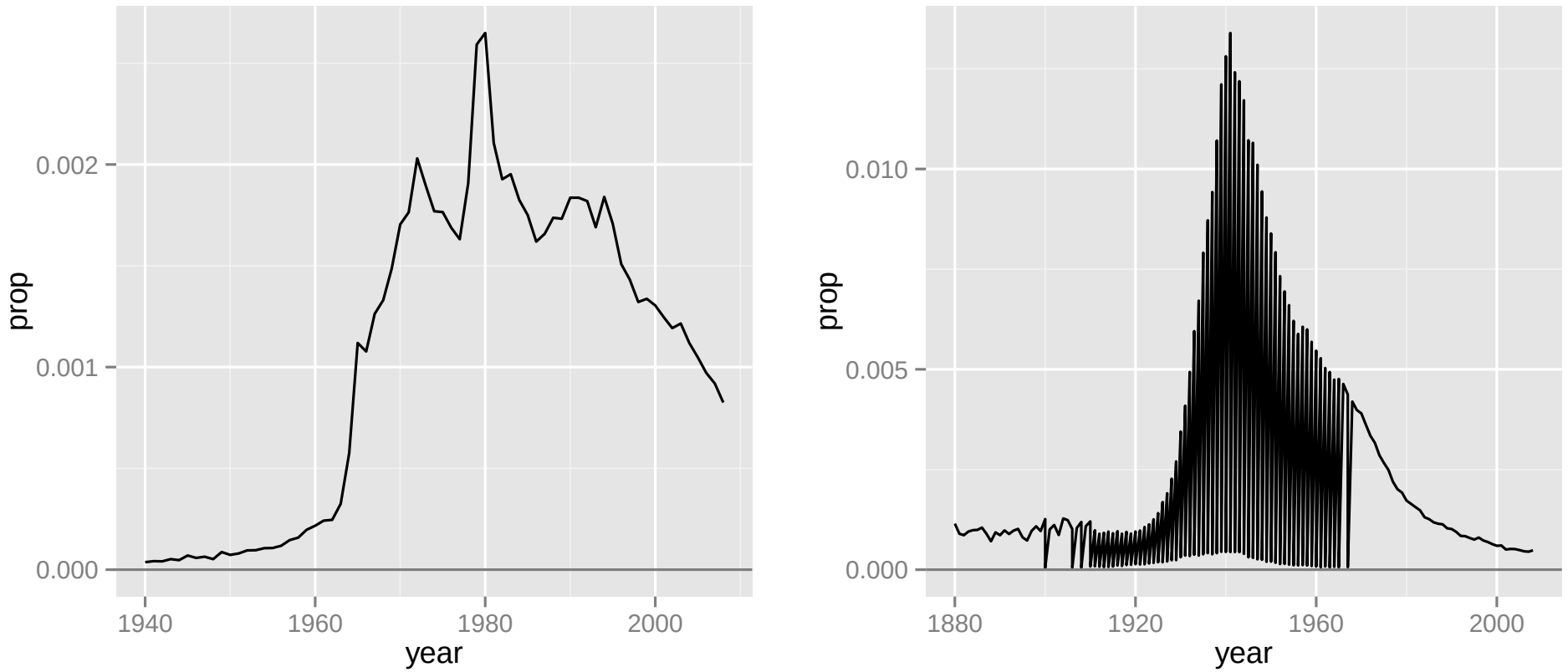
Your turn

Your name, or a similar name

Extract your name from the dataset.
Plot the trend over time.
What geom should you use? Do you need any extra aesthetics?

```r
dat.erik  <- subset(bnames, name == "Erik" )
dat.jerry <- subset(bnames, name == "Jerry")

library(ggplot2)
p1 <- ggplot(dat.erik, aes(x = year, y = prop))
p1 <- p1 + geom_line()
p1 <- p1 + geom_hline(aes(yintercept = 0), colour = "gray50")

p2 <- ggplot(dat.jerry, aes(x = year, y = prop))
p2 <- p2 + geom_line()
p2 <- p2 + geom_hline(aes(yintercept = 0), colour = "gray50")
```

```
library(gridExtra)
grid.arrange(p1, p2, nrow = 1)
```



Your turn

Names that sound like yours

Use the `soundex` variable to extract all names that sound like yours.
Plot the trend over time.
Do you have any difficulties? Think about grouping.

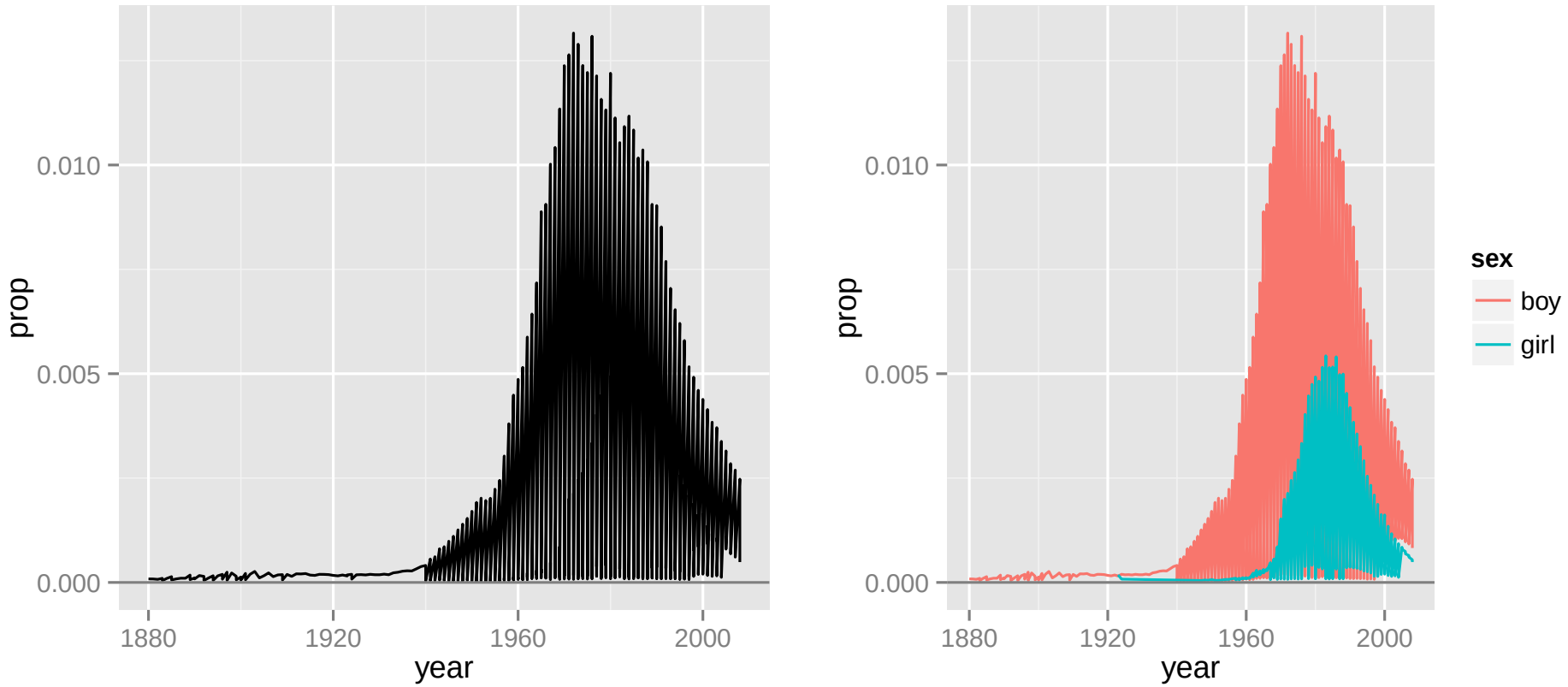
Names plots 1

```
glike <- subset(bnames, soundex == dat.erik[1,"soundex"])

library(ggplot2)
p1 <- ggplot(glike, aes(x = year, y = prop))
p1 <- p1 + geom_line()
p1 <- p1 + geom_hline(aes(yintercept = 0), colour = "gray50")

p2 <- ggplot(glike, aes(x = year, y = prop))
p2 <- p2 + geom_line(aes(colour = sex))
p2 <- p2 + geom_hline(aes(yintercept = 0), colour = "gray50")
```

Names plots 1

Sawtooth appearance implies grouping is incorrect.

```
library(gridExtra)
grid.arrange(p1, p2, nrow = 1)
```

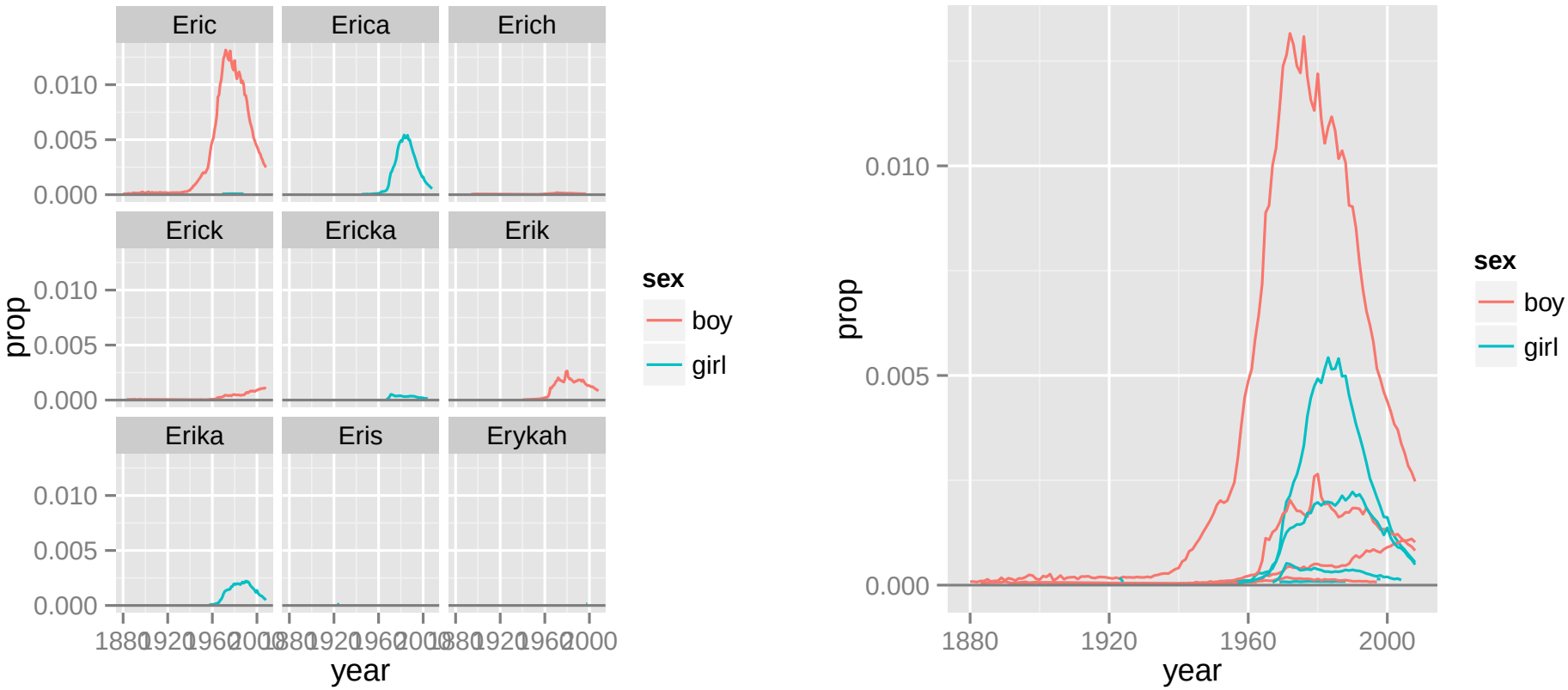


Names plots 2

```
p3 <- ggplot(glike, aes(x = year, y = prop))
p3 <- p3 + geom_line(aes(colour = sex))
p3 <- p3 + geom_hline(aes(yintercept = 0), colour = "gray50")
p3 <- p3 + facet_wrap( ~ name)

p4 <- ggplot(glike, aes(x = year, y = prop
            , group = interaction(sex, name)))
p4 <- p4 + geom_line(aes(colour = sex))
p4 <- p4 + geom_hline(aes(yintercept = 0), colour = "gray50")
```

Names plots 2

```
library(gridExtra)
grid.arrange(p3, p4, nrow = 1)
```



Subset, summarise, and arrange

Four functions

Four functions that filter rows, create summaries, add new variables, and rearrange the rows.

```
subset()

library(plyr)
summarise()
mutate()
arrange()
```

They all have similar syntax.
The first argument is a data frame, and all other arguments are interpreted in the context of that data frame.
Each returns a data frame.

Color/value data example

```
df <- data.frame(color = c("blue", "black", "blue"
                           , "blue", "black")
                 , value = 1:5)
str(df)
```

```
## 'data.frame': 5 obs. of  2 variables:
##  $ color: chr  "blue" "black" "blue" "blue" ...
##  $ value: int  1 2 3 4 5
```

```
df
```

```
##   color value
## 1  blue     1
## 2 black     2
## 3  blue     3
## 4  blue     4
## 5 black     5
```

subset()

```
df
```

```
##   color value
## 1  blue     1
## 2 black     2
## 3  blue     3
## 4  blue     4
## 5 black     5
```

```
subset(df, color == "blue")
```

```
##   color value
## 1  blue     1
## 3  blue     3
## 4  blue     4
```

summarise() 1

```
df
```

```
##   color value
## 1  blue     1
## 2 black     2
## 3  blue     3
## 4  blue     4
## 5 black     5
```

```
library(plyr)
summarise(df, double = 2 * value)
```

```
##   double
## 1      2
## 2      4
## 3      6
## 4      8
## 5     10
```

summarise() 2

```
df
```

```
##   color value
## 1  blue     1
## 2 black     2
## 3  blue     3
## 4  blue     4
## 5 black     5
```

```
library(plyr)
summarise(df, total = sum(value))
```

```
##   total
## 1    15
```

mutate() 1

```
df
```

```
##   color value
## 1  blue     1
## 2 black     2
## 3  blue     3
## 4  blue     4
## 5 black     5
```

```
library(plyr)
mutate(df, double = 2 * value)
```

```
##   color value double
## 1  blue     1      2
## 2 black     2      4
## 3  blue     3      6
## 4  blue     4      8
## 5 black     5     10
```

mutate() 2

```
df
```

```
##   color value
## 1  blue     1
## 2 black     2
## 3  blue     3
## 4  blue     4
## 5 black     5
```

```
library(plyr)
mutate(df, total = sum(value))
```

```
##   color value total
## 1  blue     1    15
## 2 black     2    15
## 3  blue     3    15
## 4  blue     4    15
## 5 black     5    15
```

arrange() 1

```
df
```

```
##   color value
## 1  blue     1
## 2 black     2
## 3  blue     3
## 4  blue     4
## 5 black     5
```

```
library(plyr)
arrange(df, color)
```

```
##   color value
## 1 black     2
## 2 black     5
## 3  blue     1
## 4  blue     3
## 5  blue     4
```

arrange() 2

```
df
```

```
##   color value
## 1  blue     1
## 2 black     2
## 3  blue     3
## 4  blue     4
## 5 black     5
```

```
library(plyr)
arrange(df, desc(color))
```

```
##   color value
## 1  blue     1
## 2  blue     3
## 3  blue     4
## 4 black     2
## 5 black     5
```

Your turn

Apply to your name

In which year was your name most popular? Least popular?
Reorder the data frame containing your name from highest to lowest popularity.
Add a new column that gives the number of babies per million with your name.

Your name 1

In which year was your name most popular? Least popular?

```
summarise(dat.erik
        , least = year[prop == min(prop)]
        , most = year[prop == max(prop)])
```

```
##   least most
## 1  1940 1980
```

```
# OR
summarise(dat.erik
        , least = year[which.min(prop)]
        , most = year[which.max(prop)])
```

```
##   least most
## 1  1940 1980
```

Your name 2

Reorder the data frame containing your name from highest to lowest popularity.

```
head(arrange(dat.erik, desc(prop)), 4)
```

```
##   year name     prop sex soundex
## 1 1980 Erik 0.002649 boy    E620
## 2 1979 Erik 0.002592 boy    E620
## 3 1981 Erik 0.002106 boy    E620
## 4 1972 Erik 0.002030 boy    E620
```

```
tail(arrange(dat.erik, desc(prop)), 4)
```

```
##    year name    prop sex soundex
## 66 1944 Erik 4.7e-05 boy    E620
## 67 1941 Erik 4.2e-05 boy    E620
## 68 1942 Erik 4.1e-05 boy    E620
## 69 1940 Erik 3.7e-05 boy    E620
```

Your name 3

Add a new column that gives the number of babies per million with your name.

```
head(mutate(dat.erik, perMil = round(1e6 * prop)))
##       year name    prop sex soundex perMil
## 60969 1940 Erik 3.7e-05 boy    E620     37
## 61872 1941 Erik 4.2e-05 boy    E620     42
## 62860 1942 Erik 4.1e-05 boy    E620     41
## 63742 1943 Erik 5.2e-05 boy    E620     52
## 64776 1944 Erik 4.7e-05 boy    E620     47
## 65619 1945 Erik 7.0e-05 boy    E620     70
```

Your turn

Brainstorm

Thinking about the data, what are some of the trends that you might want to explore?
What additional variables would you need to create?
What other data sources might you want to use?
Pair up and brainstorm for 2 minutes.

Operations External vs Internal to dataset

External	Internal
Biblical names	First/last letter
Hurricanes	Length
Ethnicity	Vowels
Famous people	Rank
	Sounds-like
`join()`	`ddply()`

Merging/Joining data

Combining datasets

```
what_played <- data.frame(
              name = c("John", "Paul", "George"
                     , "Ringo", "Stuart", "Pete")
            , instrument = c("guitar", "bass", "guitar"
                     , "drums", "bass", "drums"))

members <- data.frame(
             name = c("John", "Paul", "George"
                    , "Ringo", "Brian")
           , band = c("TRUE", "TRUE", "TRUE"
                    , "TRUE", "FALSE"))
```

Combining data sets

What should we get when we combine these two datasets?

```
what_played
##      name instrument
## 1    John     guitar
## 2    Paul       bass
## 3  George     guitar
## 4   Ringo      drums
## 5  Stuart       bass
## 6    Pete      drums
```

```
members
##      name  band
## 1    John  TRUE
## 2    Paul  TRUE
## 3  George  TRUE
## 4   Ringo  TRUE
## 5   Brian FALSE
```

join 1

```
what_played
##     name instrument
## 1   John     guitar
## 2   Paul       bass
## 3 George     guitar
## 4  Ringo      drums
## 5 Stuart       bass
## 6   Pete      drums
```

```
members
##     name  band
## 1   John  TRUE
## 2   Paul  TRUE
## 3 George  TRUE
## 4  Ringo  TRUE
## 5  Brian FALSE
```

```
join(what_played
   , members
   , type = "left")
## Joining by:  name
##     name instrument band
## 1   John     guitar TRUE
## 2   Paul       bass TRUE
## 3 George     guitar TRUE
## 4  Ringo      drums TRUE
## 5 Stuart       bass <NA>
## 6   Pete      drums <NA>
```

join 2

```
what_played
##     name instrument
## 1   John     guitar
## 2   Paul       bass
## 3 George     guitar
## 4  Ringo      drums
## 5 Stuart       bass
## 6   Pete      drums
```

```
members
##     name  band
## 1   John  TRUE
## 2   Paul  TRUE
## 3 George  TRUE
## 4  Ringo  TRUE
## 5  Brian FALSE
```

```
join(what_played
    , members
    , type = "right")
## Joining by:  name
##     name instrument  band
## 1   John     guitar  TRUE
## 2   Paul       bass  TRUE
## 3 George     guitar  TRUE
## 4  Ringo      drums  TRUE
## 5  Brian       <NA> FALSE
```

join 3

```
what_played
##     name instrument
## 1   John     guitar
## 2   Paul       bass
## 3 George     guitar
## 4  Ringo      drums
## 5 Stuart       bass
## 6   Pete      drums
```

```
members
##     name  band
## 1   John  TRUE
## 2   Paul  TRUE
## 3 George  TRUE
## 4  Ringo  TRUE
## 5  Brian FALSE
```

```
join(what_played
   , members
   , type = "inner")
## Joining by:  name
##     name instrument band
## 1   John     guitar TRUE
## 2   Paul       bass TRUE
## 3 George     guitar TRUE
## 4  Ringo      drums TRUE
```

join 4

```
what_played
##     name instrument
## 1   John     guitar
## 2   Paul       bass
## 3 George     guitar
## 4  Ringo      drums
## 5 Stuart       bass
## 6   Pete      drums
```

```
members
##     name  band
## 1   John  TRUE
## 2   Paul  TRUE
## 3 George  TRUE
## 4  Ringo  TRUE
## 5  Brian FALSE
```

```
join(what_played
   , members
   , type = "full")
## Joining by:  name
##     name instrument  band
## 1   John     guitar  TRUE
## 2   Paul       bass  TRUE
## 3 George     guitar  TRUE
## 4  Ringo      drums  TRUE
## 5 Stuart       bass  <NA>
## 6   Pete      drums  <NA>
## 7  Brian       <NA> FALSE
```

join(x, y, type =)

`type =`	Action
`"left"`	Include all of x, and matching rows of y
`"right"`	Include all of y, and matching rows of x
`"inner"`	Include only rows in both x and y
`"full"`	Include all rows

Your turn

Convert from proportions to absolute numbers by combining `bnames` with `births`, and then performing the appropriate calculation.

Baby names, join

```
colnames(bnames)
## [1] "year"    "name"    "prop"    "sex"     "soundex"
colnames(births)
## [1] "year"   "sex"    "births"
bnames2 <- join(bnames, births, by = c("year", "sex"))
tail(bnames2)
##        year     name     prop  sex soundex  births
## 257995 2008     Diya 0.000128 girl    D000 2072756
## 257996 2008 Carleigh 0.000128 girl    C642 2072756
## 257997 2008    Iyana 0.000128 girl    I500 2072756
## 257998 2008   Kenley 0.000127 girl    K540 2072756
## 257999 2008   Sloane 0.000127 girl    S450 2072756
## 258000 2008  Elianna 0.000127 girl    E450 2072756
```

Baby names, mutate

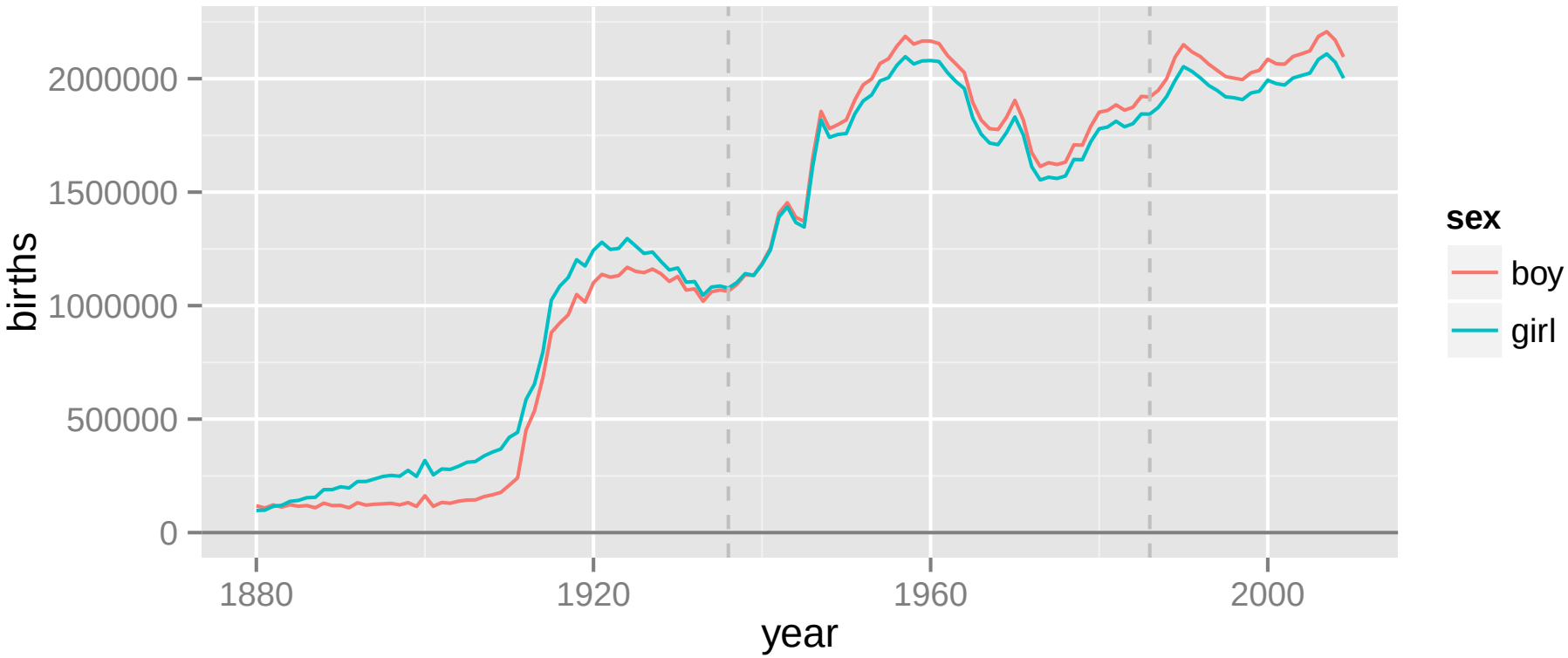
```
bnames2 <- mutate(bnames2, n = prop * births)
tail(bnames2, 3)
```

```
##        year    name     prop  sex soundex  births      n
## 257998 2008  Kenley 0.000127 girl    K540 2072756 263.24
## 257999 2008  Sloane 0.000127 girl    S450 2072756 263.24
## 258000 2008 Elianna 0.000127 girl    E450 2072756 263.24
```

```
bnames2 <- mutate(bnames2, n = round(prop * births))
tail(bnames2, 3)
```

```
##        year    name     prop  sex soundex  births   n
## 257998 2008  Kenley 0.000127 girl    K540 2072756 263
## 257999 2008  Sloane 0.000127 girl    S450 2072756 263
## 258000 2008 Elianna 0.000127 girl    E450 2072756 263
```

```r
# Births database does not contain all births!
library(ggplot2)
p1 <- ggplot(births, aes(x = year, y = births))
p1 <- p1 + geom_line(aes(colour = sex))
p1 <- p1 + geom_hline(aes(yintercept = 0), colour = "gray50")
p1 <- p1 + geom_vline(aes(xintercept = c(1936, 1986))
                      , colour = "gray75", linetype = "dashed")
print(p1)
```



1936: birth certificates were first issued

1986: needed for child tax deduction

How would we combine these?

```
members$instrument <- c("vocals", "vocals", "backup"
                        , "backup", "manager")
```

```
what_played
##     name instrument
## 1   John     guitar
## 2   Paul       bass
## 3 George     guitar
## 4  Ringo      drums
## 5 Stuart       bass
## 6   Pete      drums
```

```
members
##     name  band instrument
## 1   John  TRUE     vocals
## 2   Paul  TRUE     vocals
## 3 George  TRUE     backup
## 4  Ringo  TRUE     backup
## 5  Brian FALSE    manager
```

?

Combine, try 1

```
what_played
##      name instrument
## 1    John      guitar
## 2    Paul        bass
## 3  George      guitar
## 4   Ringo       drums
## 5  Stuart        bass
## 6    Pete       drums
```

```
members
##      name  band instrument
## 1    John  TRUE     vocals
## 2    Paul  TRUE     vocals
## 3  George  TRUE     backup
## 4   Ringo  TRUE     backup
## 5   Brian FALSE    manager
```

```
join(what_played
    , members
    , type = "full")
## Joining by:  name, instrument
##      name instrument  band
## 1    John      guitar  <NA>
## 2    Paul        bass  <NA>
## 3  George      guitar  <NA>
## 4   Ringo       drums  <NA>
## 5  Stuart        bass  <NA>
## 6    Pete       drums  <NA>
## 7    John      vocals  TRUE
## 8    Paul      vocals  TRUE
## 9  George      backup  TRUE
## 10  Ringo      backup  TRUE
## 11  Brian    manager FALSE
#  ... nope.
```

Combine, try 2

```
what_played
##      name instrument
## 1    John     guitar
## 2    Paul       bass
## 3 George     guitar
## 4  Ringo      drums
## 5 Stuart       bass
## 6   Pete      drums
```

```
members
##      name  band instrument
## 1    John  TRUE     vocals
## 2    Paul  TRUE     vocals
## 3 George  TRUE     backup
## 4  Ringo  TRUE     backup
## 5  Brian FALSE    manager
```

```
join(what_played
    , members
    , by = "name"
    , type = "full")
##      name instrument  band
## 1    John     guitar  TRUE
## 2    Paul       bass  TRUE
## 3 George     guitar  TRUE
## 4  Ringo      drums  TRUE
## 5 Stuart       bass  <NA>
## 6   Pete      drums  <NA>
## 7  Brian    manager FALSE
#  ... nope.
```

Combine, try 3

```
what_played
##      name instrument
## 1    John     guitar
## 2    Paul       bass
## 3  George     guitar
## 4   Ringo      drums
## 5  Stuart       bass
## 6    Pete      drums
```

```
members
##      name  band instrument
## 1    John  TRUE     vocals
## 2    Paul  TRUE     vocals
## 3  George  TRUE     backup
## 4   Ringo  TRUE     backup
## 5   Brian FALSE    manager
```

```
colnames(members)[3]
## [1] "instrument"
names(members)[3] <- "instrument2"
colnames(members)[3]
## [1] "instrument2"
join(what_played
   , members
   , type = "full")
## Joining by:  name
##      name instrument  band instrument2
## 1    John     guitar  TRUE      vocals
## 2    Paul       bass  TRUE      vocals
## 3  George     guitar  TRUE      backup
## 4   Ringo      drums  TRUE      backup
## 5  Stuart       bass  <NA>        <NA>
## 6    Pete      drums  <NA>        <NA>
## 7   Brian       <NA> FALSE     manager
#  ... yes!
```

Groupwise operations: Split, Apply, Combine

Number of people

How do we compute the number of people with each name over all years?
It's pretty easy if you have a single name.
(For example, how many people with your name were born over the entire 128 years?)
How would you do it?

One name

```
dat.erik <- subset(bnames2, name == "Erik")
sum(dat.erik$n)
```

```
## [1] 140877
```

```
# Or
summarise(dat.erik, n = sum(n))
```

```
##        n
## 1 140877
```

But how could we do this for every name?

Manually: Split, Apply, Combine

```
# Split
pieces <- split(bnames2, list(bnames$name))
  # pieces is a list of lists
# Apply
results <- vector("list", length(pieces))
  # results is an empy list of lists
for(i in seq_along(pieces)) {
 piece <- pieces[[i]]
 results[[i]] <- summarise(piece, name = name[1], n = sum(n))
}
  # results now has two elements in each list, name and n
# Combine
result <- do.call("rbind", results)
str(result)
```

```
## 'data.frame': 6782 obs. of  2 variables:
##  $ name: chr  "Aaden" "Aaliyah" "Aarav" "Aaron" ...
##  $ n   : num  959 39665 219 509464 25 ...
```

Equivalently, with ddply (from plyr)

```
# Or equivalently
library(plyr)
counts <- ddply(bnames2, "name", summarise, n = sum(n))
str(counts)
```

```
## 'data.frame': 6782 obs. of  2 variables:
##  $ name: chr  "Aaden" "Aaliyah" "Aarav" "Aaron" ...
##  $ n   : num  959 39665 219 509464 25 ...
```

- input data: bnames2
- way to split up input: "name"
- function to apply to each piece: "summarise"
- additional arguments to function: n = sum(n)
- (custom functions can be written in place of summarise)

ddply, visual example 1

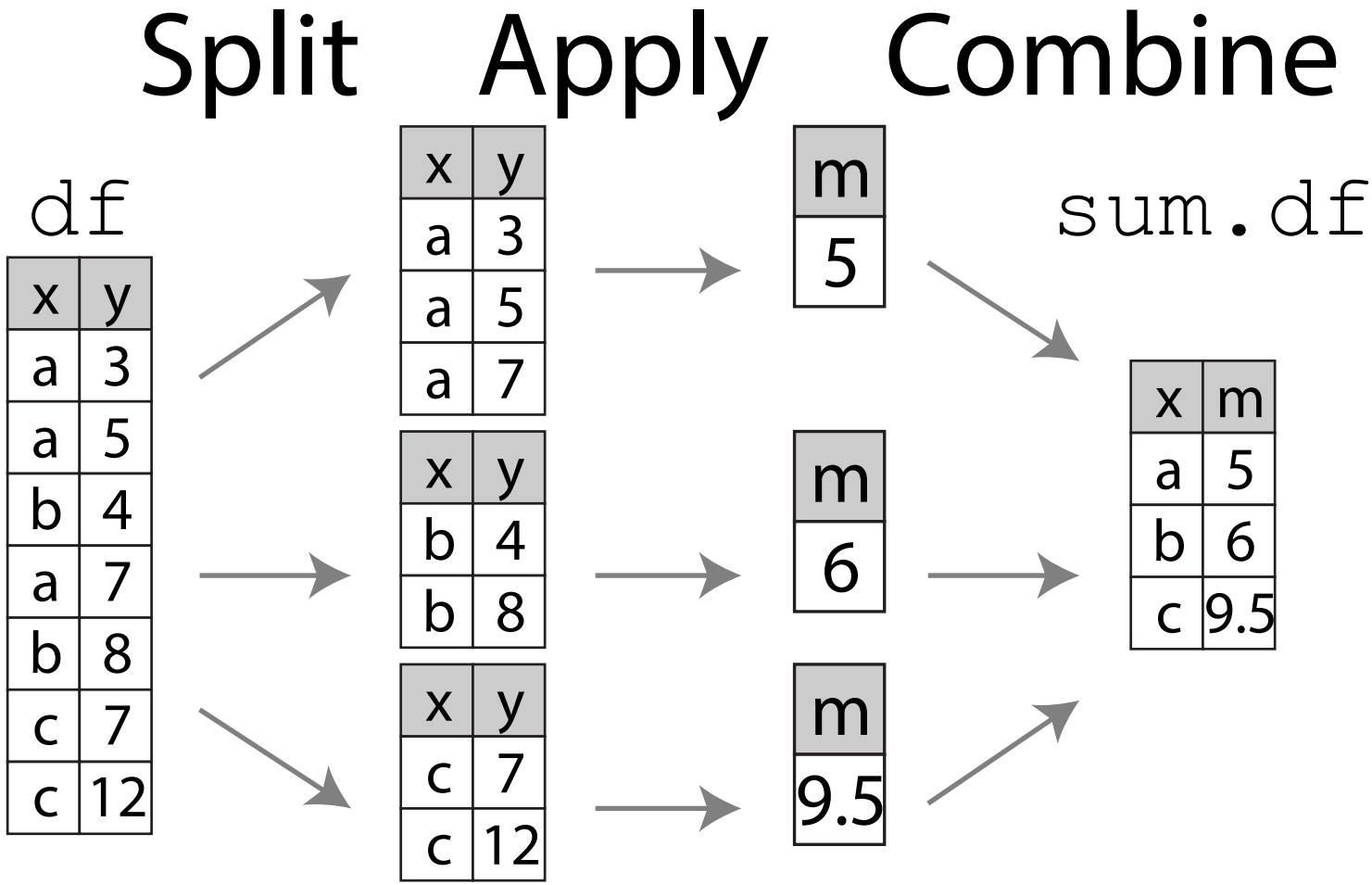
```
df <- data.frame(x = c("a", "a", "b", "a", "b", "c", "c")
                 , y = c(3, 5, 4, 7, 8, 7, 12))
df
```

```
##   x  y
## 1 a  3
## 2 a  5
## 3 b  4
## 4 a  7
## 5 b  8
## 6 c  7
## 7 c 12
```

```
library(plyr)
sum.df <- ddply(df, "x", summarise, m = mean(y))
sum.df
```

```
##   x   m
## 1 a 5.0
## 2 b 6.0
## 3 c 9.5
```

ddply, visual example 2




```
sum.df <- ddply(df, "x", summarise, m = mean(y))
```

Your turn

Soundex

Repeat the same operation, but use `soundex` instead of `name`.
What is the most common sound?
What name does it correspond to?
(Hint: use join)

Most popular name sound

```
# count by soundex
scounts <- ddply(bnames2, "soundex", summarise, n = sum(n))
# sort descending
scounts <- arrange(scounts, desc(n))

# Combine with names. When there are multiple
#   possible matches, picks first match.
scounts <- join(scounts, bnames2[, c("soundex", "name")]
                , by = "soundex", match = "first")
```

Most popular name sound

```
# most popular sound
head(scounts)
```

```
##   soundex       n    name
## 1    J500 9991737    John
## 2    M240 5823791 Michael
## 3    M600 5553703    Mary
## 4    J520 5524958   James
## 5    R163 5047182  Robert
## 6    W450 4116109 William
```

```
# names with that sound
head(subset(bnames, soundex == "J500"))
```

```
##     year    name      prop sex soundex
## 1   1880    John 0.081541 boy    J500
## 49  1880     Jim 0.002914 boy    J500
## 272 1880    Juan 0.000329 boy    J500
## 353 1880  Jimmie 0.000203 boy    J500
## 354 1880 Johnnie 0.000203 boy    J500
```