Data manipulation

Hadley Wickham

Assistant Professor / Dobelman Family Junior Chair

Department of Statistics / Rice University

November 2010

1. Baby names data
2. Slicing and dicing
3. Merging data
4. Group-wise operations
5. Challenges

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.

```
> head(bnames, 20)
   year    name soundex     prop sex
1  1880    John    J500 0.081541 boy
2  1880 William    W450 0.080511 boy
3  1880   James    J520 0.050057 boy
4  1880 Charles    C642 0.045167 boy
5  1880  George    G620 0.043292 boy
6  1880   Frank    F652 0.027380 boy
7  1880  Joseph    J210 0.022229 boy
8  1880  Thomas    T520 0.021401 boy
9  1880   Henry    H560 0.020641 boy
10 1880  Robert    R163 0.020404 boy
11 1880  Edward    E363 0.019965 boy
12 1880   Harry    H600 0.018175 boy
13 1880  Walter    W436 0.014822 boy
14 1880  Arthur    A636 0.013504 boy
15 1880    Fred    F630 0.013251 boy
16 1880  Albert    A416 0.012609 boy
17 1880  Samuel    S540 0.008648 boy
18 1880   David    D130 0.007339 boy
19 1880   Louis    L200 0.006993 boy
20 1880     Joe    J000 0.006174 boy
```

```
> tail(bnames, 20)
        year     name soundex     prop  sex
257981 2008     Miya    M000 0.000130 girl
257982 2008     Rory    R600 0.000130 girl
257983 2008  Desirae    D260 0.000130 girl
257984 2008   Kianna    K500 0.000130 girl
257985 2008   Laurel    L640 0.000130 girl
257986 2008   Neveah    N100 0.000130 girl
257987 2008   Amaris    A562 0.000129 girl
257988 2008 Hadassah    H320 0.000129 girl
257989 2008    Dania    D500 0.000129 girl
257990 2008   Hailie    H400 0.000129 girl
257991 2008   Jamiya    J500 0.000129 girl
257992 2008    Kathy    K300 0.000129 girl
257993 2008   Laylah    L400 0.000129 girl
257994 2008     Riya    R000 0.000129 girl
257995 2008     Diya    D000 0.000128 girl
257996 2008 Carleigh    C642 0.000128 girl
257997 2008    Iyana    I500 0.000128 girl
257998 2008   Kenley    K540 0.000127 girl
257999 2008   Sloane    S450 0.000127 girl
258000 2008  Elianna    E450 0.000127 girl
```

Getting started

```
library(plyr)
library(ggplot2)

options(stringsAsFactors = FALSE)
# Can read compressed files directly
# Don't need to unzip first
# Very handy!
bnames <- read.csv("baby-names2.csv.bz2")
births <- read.csv("baby-births.csv")
```

Working directories

In one directory

Data (.csv)

+

Code (.r)

+

Graphics (.png, .pdf)

+

Written report (.tex)

Working directory

Set your working directory to specify where files will be loaded from and saved to.

From the terminal (linux or mac): the working directory is the directory you're in when you start R

On windows: File | Change dir.

On the mac: ⌘-D

Coding strategy

At the end of each interactive session, you want a summary of everything you did. Two options:

1. Save everything you did with `savehistory()` then remove the unimportant bits.

2. Build up the important bits as you go. (this is how I work)

Trends

Your turn

Extract your name from the dataset. Plot the trend over time.

What geom should you use? Do you need any extra aesthetics?

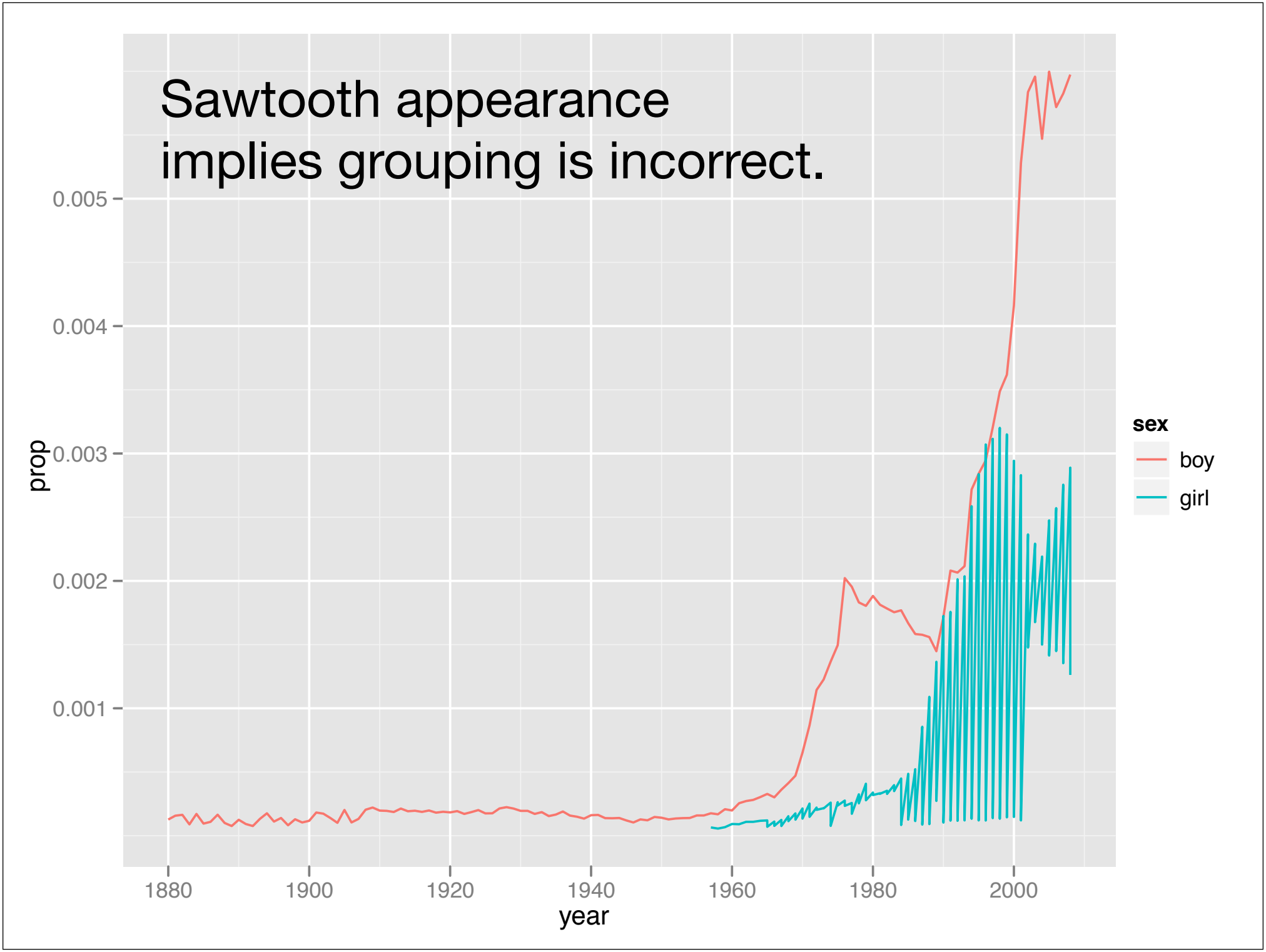
```
hadley <- subset(bnames, name == "Hadley")


qplot(year, prop, data = hadley, colour = sex,
  geom ="line")
# :(
```

Your turn

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.

```
gabi <- subset(bnames, soundex == "G164")
qplot(year, prop, data = gabi)
qplot(year, prop, data = gabi, geom = "line")

qplot(year, prop, data = gabi, geom = "line",
  colour = sex) + facet_wrap(~ name)

qplot(year, prop, data = gabi, geom = "line",
  colour = sex, group = interaction(sex, name))
```


Sawtooth appearance
implies grouping is incorrect.
0.005
0.004
0.003
0.002
0.001
prop
1880
1900
1920
1940
1960
1980
2000
year
sex
boy
girl

Slicing and dicing

Function	Package
`subset`	`base`
`summarise`	`plyr`
`transform`	`base`
`arrange`	`plyr`

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
blue	1
black	2
blue	3
blue	4
black	5

color	value
blue	1
blue	3
blue	4

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

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

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

```
transform(df, double = 2 * value)
```

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

double
2
4
6
8
10

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

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

total
15

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

color	value
4	1
1	2
5	3
3	4
2	5

color	value
1	2
2	5
3	4
4	1
5	3

```
arrange(df, color)
```

color	value
4	1
1	2
5	3
3	4
2	5

color	value
5	3
4	1
3	4
2	5
1	2

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

Your turn

Calculate the total proportion, and largest and smallest proportions of your name.

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

```
summarise(bnames,
  total = sum(prop),
  largest = max(prop),
  smallest = min(prop))

arrange(hadley, desc(prop))
```

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.

External	Internal
Biblical names Hurricanes Ethnicity Famous people	First/last letter Length Vowels Rank Sounds-like

`join` `ddply`

Merging data

Combining datasets

Name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

+

Name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

= ?

x

Name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

+

y

Name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

=

Name	instrument	band
John	guitar	T
Paul	bass	T
George	guitar	T
Ringo	drums	T
Stuart	bass	NA
Pete	drums	NA

```
join(x, y, type = "left")
```

x

Name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

+

y

Name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

=

Name	instrument	band
John	guitar	T
Paul	bass	T
George	guitar	T
Ringo	drums	T
Brian	NA	F

```
join(x, y, type = "right")
```

x

Name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

+

y

Name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

=

Name	instrument	band
John	guitar	T
Paul	bass	T
George	guitar	T
Ringo	drums	T

```
join(x, y, type = "inner")
```

x

Name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

+

y

Name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

=

Name	instrument	band
John	guitar	T
Paul	bass	T
George	guitar	T
Ringo	drums	T
Stuart	bass	NA
Pete	drums	NA
Brian	NA	F

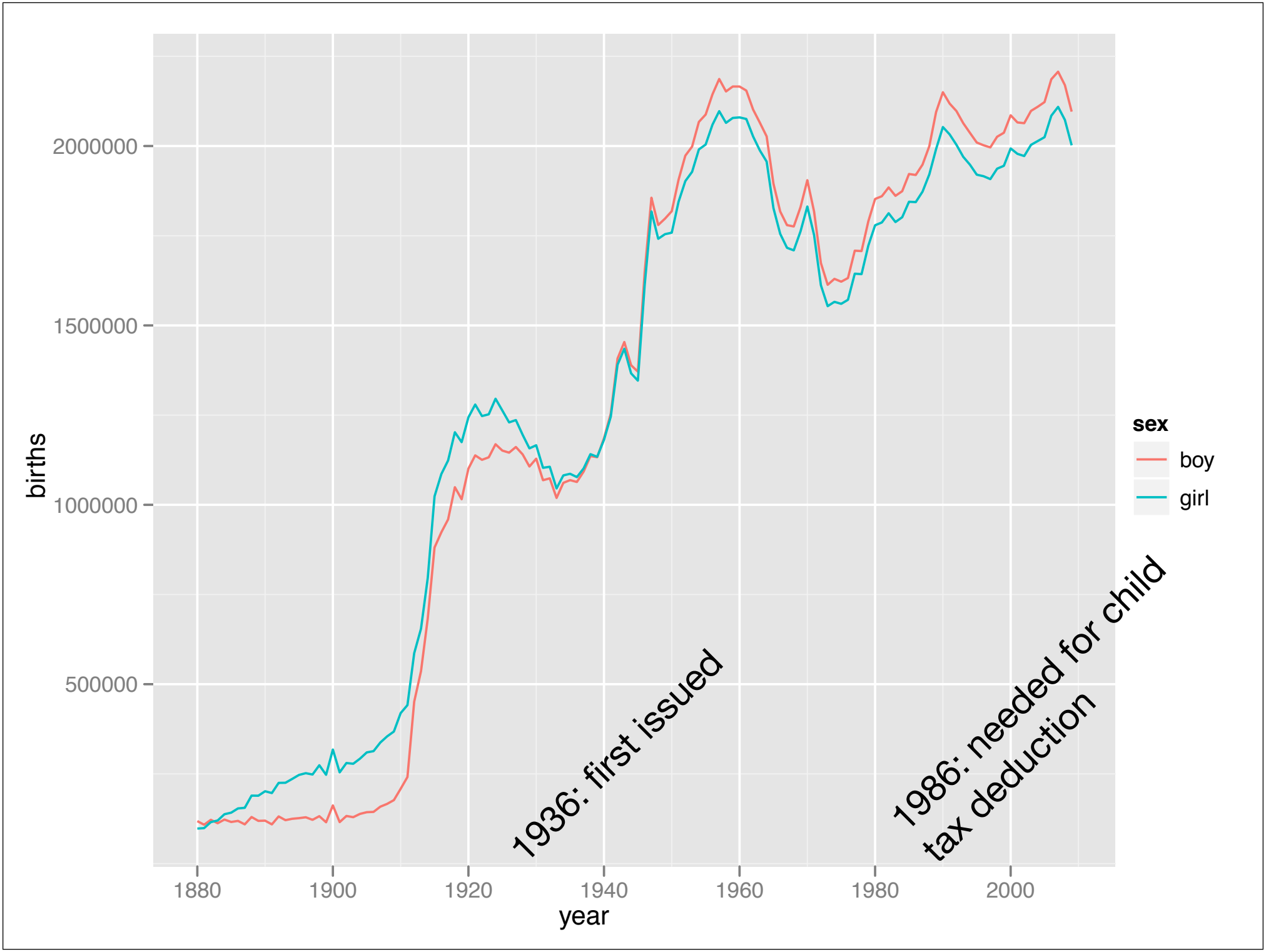
```
join(x, y, type = "full")
```

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.

```
bnames2 <- join(bnames, births,
  by = c("year", "sex"))
tail(bnames2)

bnames2 <- transform(bnames2, n = prop * births)
tail(bnames2)

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


2000000
1500000
1000000
500000
births
1880
1900
1920
1940
1960
1980
2000
year
1936: first issued
1986: needed for child tax deduction
sex
boy
girl

Group-wise operations

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.

How would you do it?

```
hadley <- subset(bnames2, name == "Hadley")
sum(hadley$n)

# Or
summarise(hadley, n = sum(n))

# But how could we do this for every name?
```

```
# Split
pieces <- split(bnames2, list(bnames$name))

# Apply
results <- vector("list", length(pieces))
for(i in seq_along(pieces)) {
  piece <- pieces[[i]]
  results[[i]] <- summarise(piece,
    name = name[1], n = sum(n))
}

# Combine
result <- do.call("rbind", results)
```

```
# Or equivalently

counts <- ddply(bnames2, "name", summarise,
  n = sum(n))
```

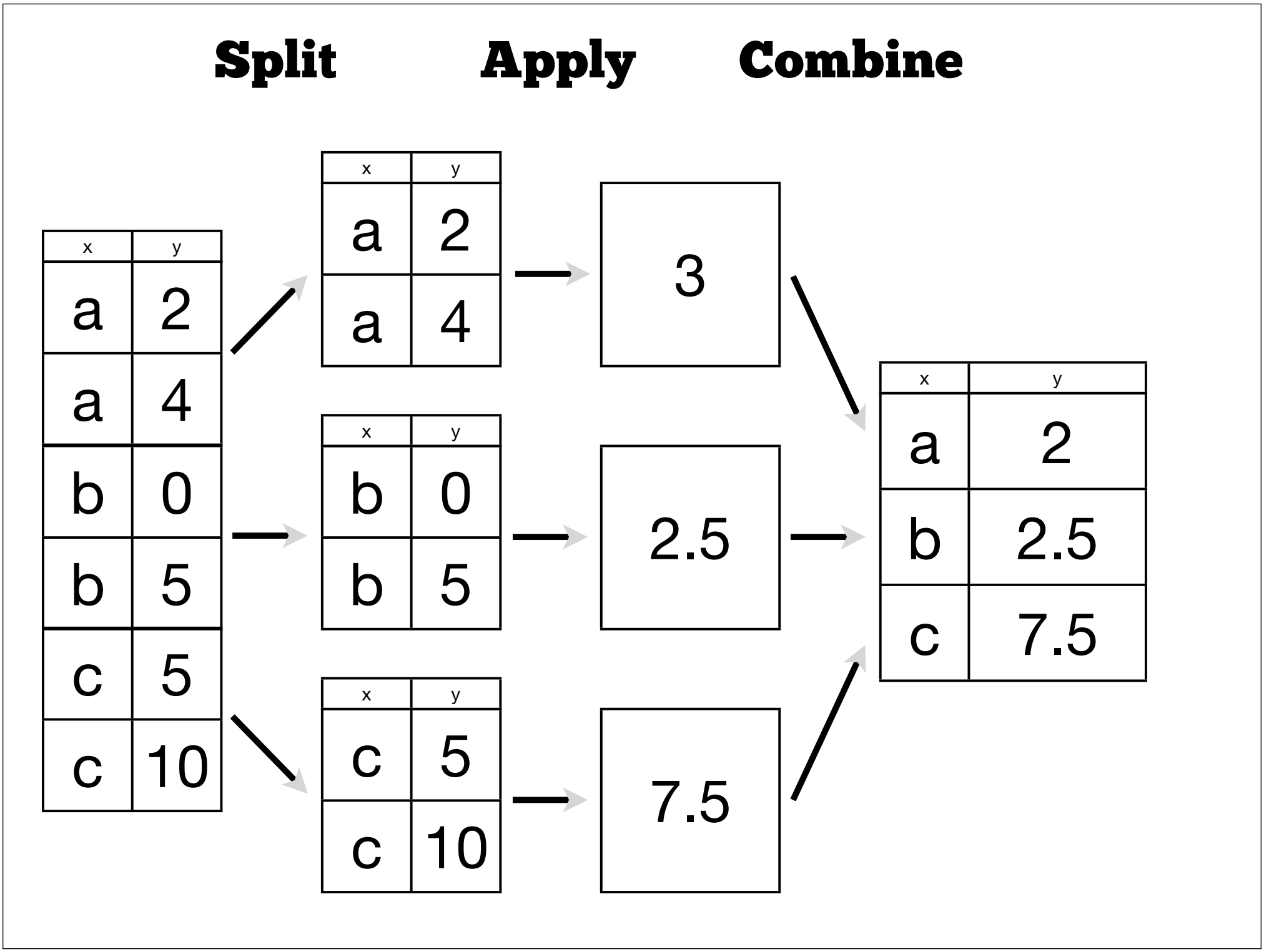
```
# Or equivalent

counts <- ddply(bnames2, "name", summarise,
  n = sum(n))
```

Input data

Way to split up input

2nd argument to `summarise()`

Function to apply to each piece


Split
Apply
Combine
x
y
a
2
a
4
b
0
b
5
c
5
c
10
3
2.5
7.5

Your turn

Repeat the same operation, but use soundex instead of name. What is the most common sound? What name does it correspond to?

```
scounts <- ddply(bnames2, "soundex", summarise,
  n = sum(n))
scounts <- arrange(scounts, desc(n))

# Combine with names
# When there are multiple possible matches,
# join picks the first
scounts <- join(
  scounts, bnames2[, c("soundex", "name")],
  by = "soundex")
head(scounts, 100)

subset(bnames, soundex == "L600")
```

Transformations

Transformations

What about group-wise **transformations**? e.g. what if we want to compute the rank of a name within a sex and year?

This task is easy if we have a single year & sex, but hard otherwise.

How would you do it for a single group?

```
one <- subset(bnames, sex == "boy" & year == 2008)
one$rank <- rank(-one$prop,
  ties.method = "first")

# or
one <- transform(one,
  rank = rank(-prop, ties.method = "min"))
head(one)
```

What if we want to transform every sex and year?

Workflow

1. Extract a single group
2. Figure out how to solve it for just that group
3. Use `ddply` to solve it for all groups

How would you use ddply to calculate all ranks?

```
bnames <- ddply(bnames, c("sex", "year"), transform,
  rank = rank(-prop, ties.method = "min"))
```

`ddply + transform` =
group-wise transformation

`ddply + summarise` =
per-group summaries

`ddply + subset` =
per-group subsets

Tools

You know have all the tools to solve 95% of data manipulation problems in R. It's just a matter of figuring out which tools to use, and how to combine them.

The following challenges will give you some practice.

More plyr functions

Many problems involve splitting up a large data structure, operating on each piece and joining the results back together:

split-apply-combine

How you split up depends on the type of input: **arrays**, **data frames**, **lists**

How you combine depends on the type of output: **arrays**, **data frames**, **lists**, **nothing**

	array	data frame	list	nothing
array	aaply	adply	alply	a_ply
data frame	daply	ddply	dlply	d_ply
list	laply	ldply	llply	l_ply
n replicates	raply	rdply	rlply	r_ply
function arguments	maply	mdply	**mlply**	m_ply

	array	data frame	list	nothing
array	apply	adply	alply	a_ply
data frame	daply	*aggregate*	by	d_ply
list	sapply	ldply	lapply	l_ply
n replicates	replicate	rdply	replicate	r_ply
function arguments	mapply	mdply	mapply	m_ply