Data manipulation

Hadley Wickham

Assistant Professor / Dobelman Family Junior Chair

Department of Statistics / Rice University

September 2011

1. US baby names data
2. Loading data
3. Subsetting
4. Transforming & summarising
5. Group-wise transformations & summaries

Baby names

Top 1000 male and female baby names in the US, from 1880 to 2008.

258,000 records (1000 * 2 * 129)

But only four variables: year, name, sex and prop.

Getting started

```
library(plyr)
library(ggplot2)

options(stringsAsFactors = FALSE)

# Big data tip: read compressed files directly
bnames <- read.csv("baby-names2.csv.bz2")

births <- read.csv("births.csv")
bnames <- join(bnames, births, by = c("year", "sex"))
bnames <- mutate(bnames, n = round(prop * births))
```

```
library(stringr)

vowels <- function(x) {
  str_length(str_replace_all(tolower(x),
    "[^aeiouy]", ""))
}

bnames <- transform(bnames,
  first = tolower(str_sub(name, 1, 1)),
  last = tolower(str_sub(name, -1, -1)),
  vowels = vowels(name),
  length = nchar(name),
  per10000 = 10000 * prop,
  one_per = 1 / prop
)
```

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

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.

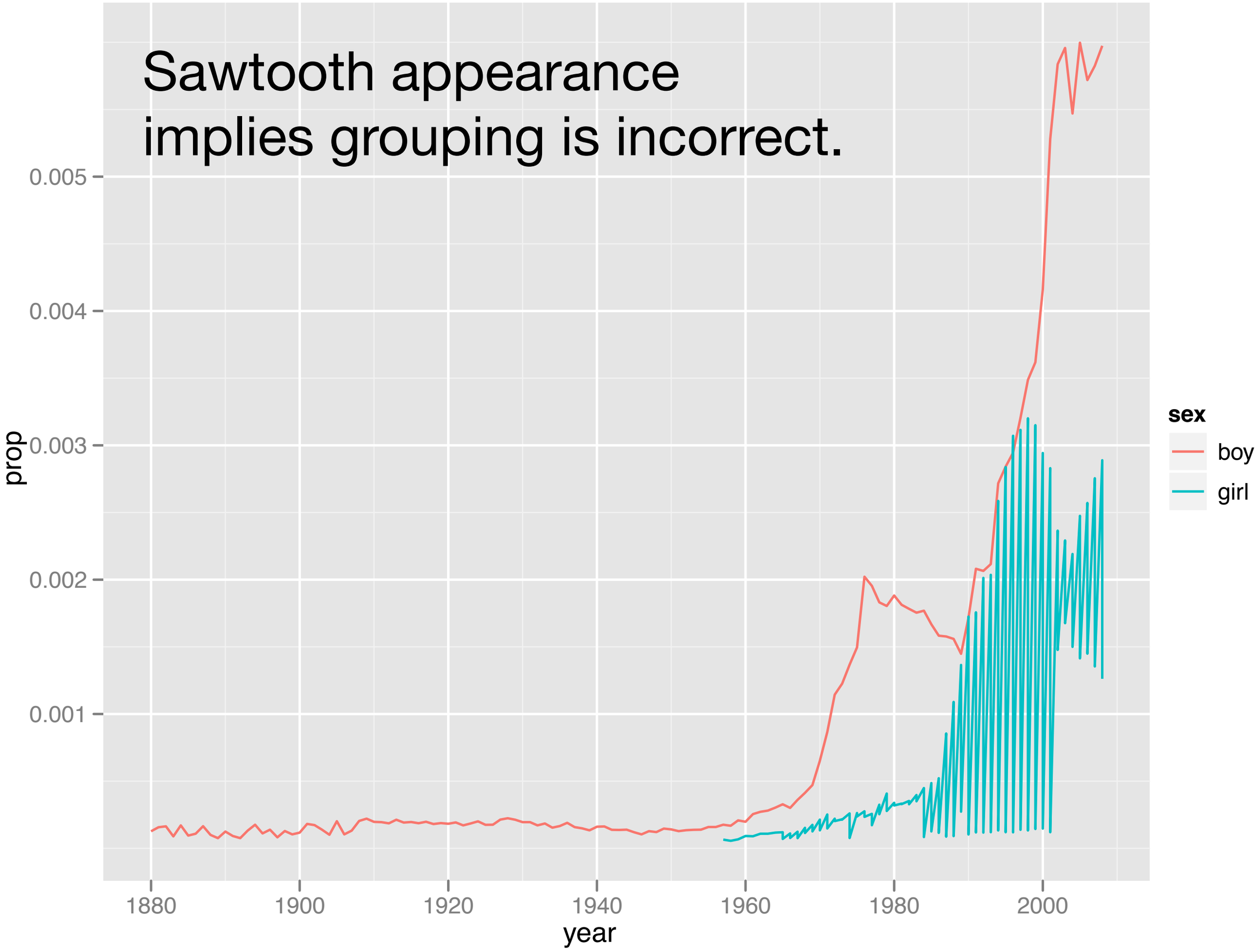
Do you have any difficulties? Think about grouping.

```
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.



Slicing and dicing

Function	Package
`subset`	`base`
`summarise`	`plyr`
`mutate`	`plyr`
`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

```
mutate(df, double = 2 * value,
    quadruple = 2 * double)
```

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

Using the data frame containing your name:

Reorder from highest to lowest popularity.

Calculate the total number of people with your name, and the average number of people given your name each year

Add a new column that stores the rank of each year according to n

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

summarise(hadley,
  total = sum(n),
  avg = mean(n),
  avg2 = sum(n) / 129)

mutate(hadley, rank = rank(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`

Group-wise transformations

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(bnames, name == "Hadley")
sum(hadley$n)

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

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

```r
# Split
pieces <- split(bnames, 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(bnames, "name", summarise,
  n = sum(n))
```

```
# Or equivalent

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

Input data

Way to split up input

2nd argument to `summarise()`

Function to apply to each piece

```
# Even faster is the special purpose count function:
counts <- count(bnames, "name", "n")

# Often where special purpose functions exist they
# will be faster.  Emphasis in plyr is on clearly
# expressing what you want, not on speed.
# (Hopefully next version will combine the best of
# both worlds)
```

x	y
a	2
a	4
b	0
b	5
c	5
c	10

Split

x	y
a	2
a	4
b	0
b	5
c	5
c	10

x	y
a	2
a	4

x	y
b	0
b	5

x	y
c	5
c	10

Split Apply

x	y
a	2
a	4
b	0
b	5
c	5
c	10

x	y
a	2
a	4

3

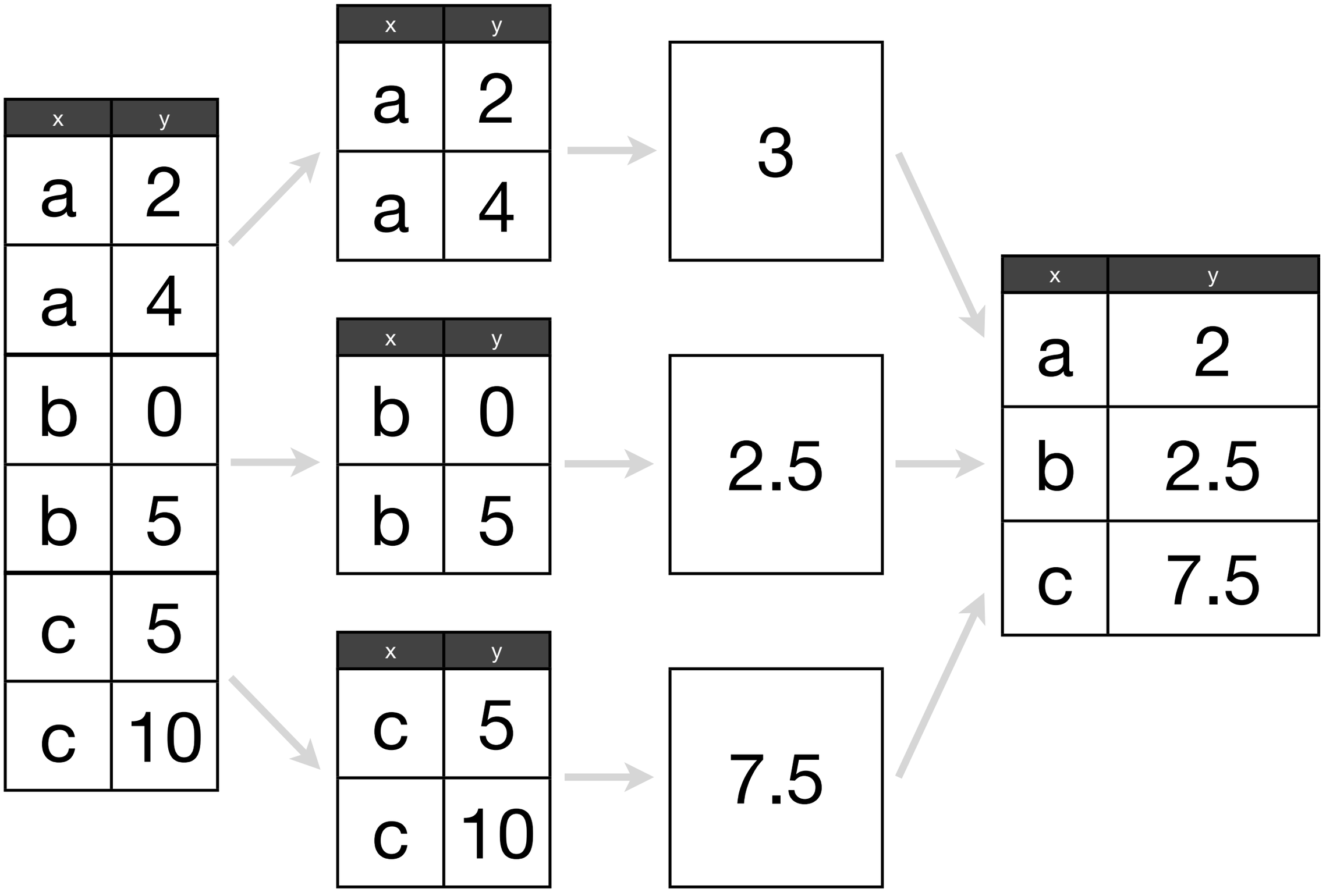
x	y
b	0
b	5

2.5

x	y
c	5
c	10

7.5

Split Apply Combine



Rank

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.

Rank

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.

Take two minutes to think about how you might attack such a problem

```
one <- subset(bnames, sex == "boy" & year == 2008)
one <- mutate(one,
  rank = rank(desc(prop), ties.method = "min"))
head(one)
```

To rank in descending order

Usual method of dealing with ties

What if we want to transform every sex and year?

Input data

Way to split up input

Function to apply to each piece

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

2nd argument to `transform()`

Summaries

In a similar way, we can use `ddply()` for group-wise summaries.

There are many base R functions for special cases. Where available, these are often much faster; but you have to know they exist, and have to remember how to use them.

```
# Explore average length

sy <- ddply(bnames, c("sex", "year"), summarise,
  avg_length = weighted.mean(length, prop))

qplot(year, avg_length, data = sy, colour = sex,
  geom = "line")
```

```
# Explore number of names of each length

syl <- ddply(bnames, c("sex", "length", "year"),
  summarise, prop = sum(prop))
qplot(year, prop, data = syl, colour = sex,
  geom = "line") + facet_wrap(~ length)

twoletters <- subset(bnames, length == 2)
unique(twoletters$name)
qplot(year, prop, data = twoletters, colour = sex,
  geom = "line") + facet_wrap(~ name)
```

Your turn

Use these tools to explore how the following have changed over time:

The number of vowels in a name.

The distribution of first (or last) letters.

The total proportion of babies with names in the top 1000, or top 100 or top 10.

```
vys <- ddply(bnames, c("vowels", "year", "sex"),
  summarise, prop = sum(prop))
qplot(year, prop, data = vys, colour = sex,
  geom = "line") + facet_wrap(~ vowels)

syl <- ddply(bnames, c("sex", "last", "year"),
  summarise, prop = sum(prop))
qplot(year, prop, data = syl, colour = sex,
  geom = "line") + facet_wrap(~ last)

sy <- ddply(bnames, c("year","sex"), summarise,
  prop = sum(prop))
qplot(year, prop, data = sy, colour = sex,
  geom = "line")
```

More about plyr

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

Fiddly details

Labelling

Progress bars

Consistent argument names

Missing values / Nulls

http://plyr.had.co.nz