http://courses.had.co.nz

Transformation

Hadley Wickham

Assistant Professor / Dobelman Family Junior Chair

Department of Statistics / Rice University

July 2012

1. Working directories
2. Loading data
3. Saving data
4. Slicing and dicing
5. Group-wise

Working directory

Why?

All paths in R are relative to the working directory. Life is much easier when you have it correctly set.

Usually want one project per directory. (See also Rstudio's project support)

Makes code easy to move between computers.

Working directory

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

Windows: File | Change dir.

Mac: ⌘-D

Rstudio: Tools | Change working dir...

```
# Find out what directory you’re in
getwd()
```

```
# List files in that directory
dir()
```

Your turn

Make sure your working directory is set to the location where you downloaded the files. Use `dir()` to check you're in the right place.

Loading data

1. Plain text
2. Excel
3. Other stats packages
4. Databases

http://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

Excel

- Save as csv. (Use VBA to automate)
- `RODBC::odbcConnectExcel` http://cran.r-project.org/doc/manuals/R-data.html#RODBC (uses excel)
- `xlsx::read.xlsx` (uses java)
- `gdata::read.xls` (uses perl)

Excel

This is what I always do

- Save as csv. (Use VBA to automate)
- `RODBC::odbcConnectExcel` http://cran.r-project.org/doc/manuals/R-data.html#RODBC (uses excel)
- `xlsx::read.xlsx` (uses java)
- `gdata::read.xls` (uses perl)

Stats packages

- `foreign::read.dta`: stata
- `foreign::read.spss`: spss
- `foreign::read.xport`: SAS export format

DBI

- Standard API for db operations
- Bindings for MySQL, Oracle, PostgreSQL, SQLite, JDBC, ODB
- `dbConnect`, `dbSendQuery`, `dbGetQuery`
- (See also `RODBC` package which implements ODBC albeit with different API to DBI)

Saving data

Your turn

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 `diamonds` to `diamonds-2.csv`.

What happens if you now read in `diamonds-2.csv`? Is it different to your `diamonds` data frame? How?

```
write.csv(diamonds, "diamonds-2.csv")
diamonds2 <- read.csv("diamonds-2.csv")

head(diamonds)
head(diamonds2)

str(diamonds)
str(diamonds2)

# Better, but still loses factor levels
write.csv(diamonds, file = "diamonds-3.csv",
  row.names = F)
diamonds3 <- read.csv("diamonds-3.csv")
```

Saving data

```
# For long-term storage
write.csv(diamonds, file = "diamonds.csv",
  row.names = FALSE)

# For short-term caching
# Preserves factors etc.
saveRDS(diamonds, "diamonds.rds")
diamonds4 <- readRDS("diamonds.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

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

# Reading is even easier:
diamonds5 <- read.csv("diamonds.csv.bz2")

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

Slicing and dicing

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.

Getting started

```
library(plyr)
library(ggplot2)

options(stringsAsFactors = FALSE)

# Can read compressed files directly
bnames <- read.csv("bnames2.csv.bz2")
```

Your turn

Extract your name from the dataset:

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

Plot the trend over time. Guess which geom you should use. Do you need any extra aesthetics?

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

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

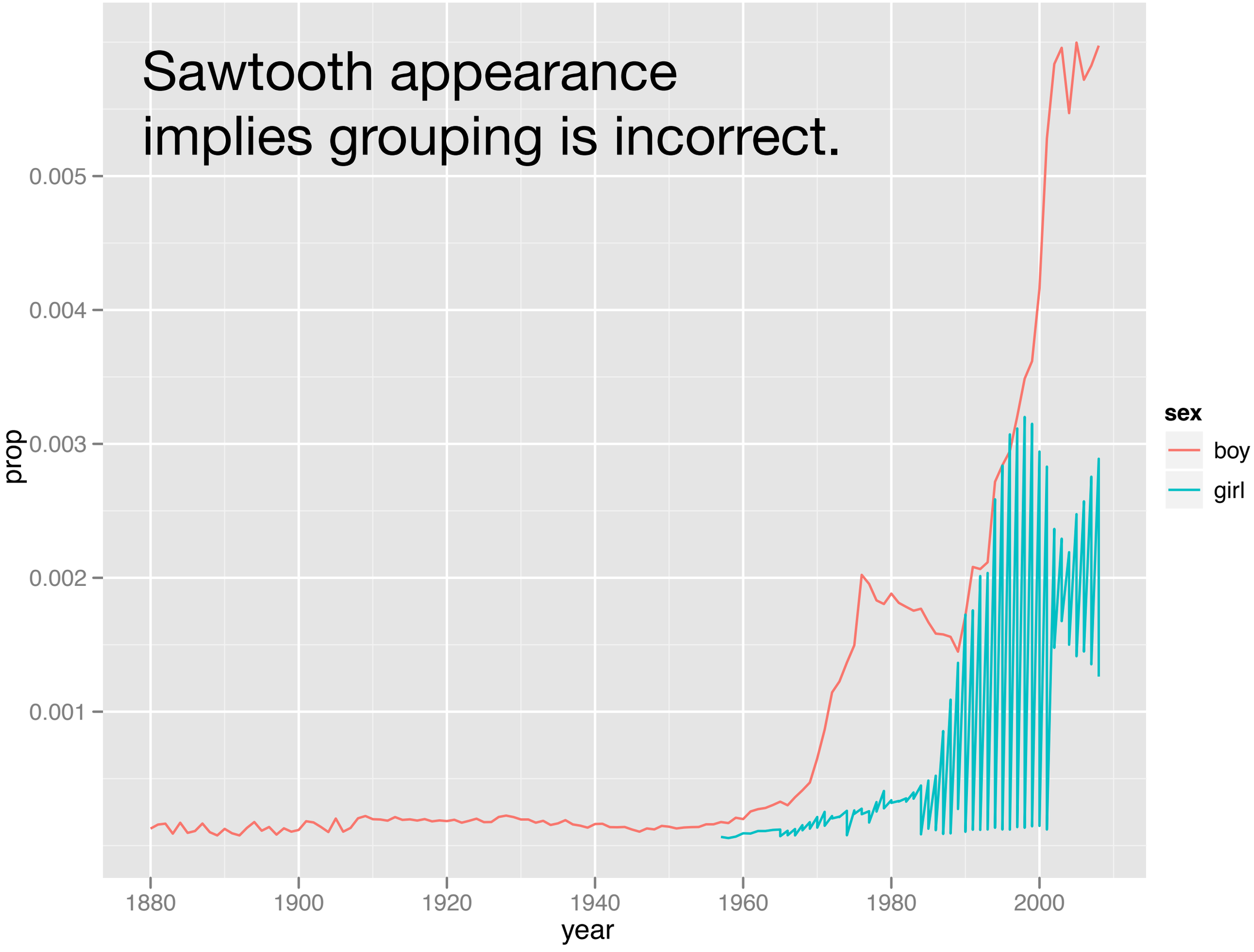
Your turn

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.



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

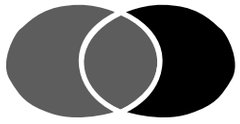
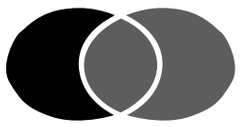
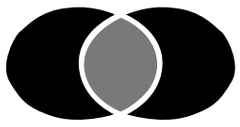
They all have similar syntax. The first argument is a data frame, and all other arguments are interpreted in the context of that data frame. Each returns a data frame.

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

color	value
blue	1
blue	3
blue	4

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

```
# Comparisons
# # < > <= >= != == %in%

# Boolean operators:
```

	a
	b
	a | b
	a & b
	a & !b
	xor(a, b)

Your turn

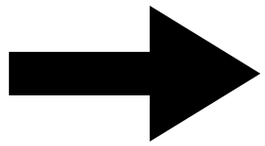
Select the cars from mpg that have:

Are made by Audi

Fewer than 6 cylinders

```
equal_dim <- diamonds$x == diamonds$y
equal <- diamonds[equal_dim, ]

diamonds[diamonds$depth >= 55 & diamonds$depth <= 70, ]

diamonds[diamonds$carat < mean(diamonds$carat), ]

diamonds[diamonds$price / diamonds$carat < 10000, ]

diamonds[diamonds$cut %in% c("Very Good", "Premium",
  "Ideal")
```

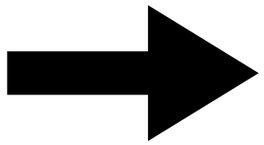
color	value
blue	1
black	2
blue	3
blue	4
black	5



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

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

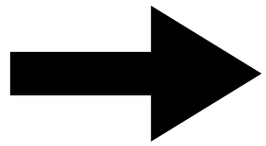
color	value
blue	1
black	2
blue	3
blue	4
black	5



color	value	double	quad
blue	1	2	4
black	2	4	8
blue	3	6	12
blue	4	8	16
black	5	10	20

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

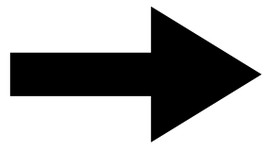
color	value
blue	1
black	2
blue	3
blue	4
black	5



double
2
4
6
8
10

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

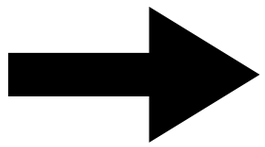
color	value
blue	1
black	2
blue	3
blue	4
black	5



total
15

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

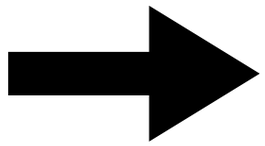
color	value
4	1
1	2
5	3
3	4
2	5



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))
```

Group-wise transformations

```
# How do we compute the number of people with each
# name over all years? It's pretty easy if you have
# a single name:

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

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

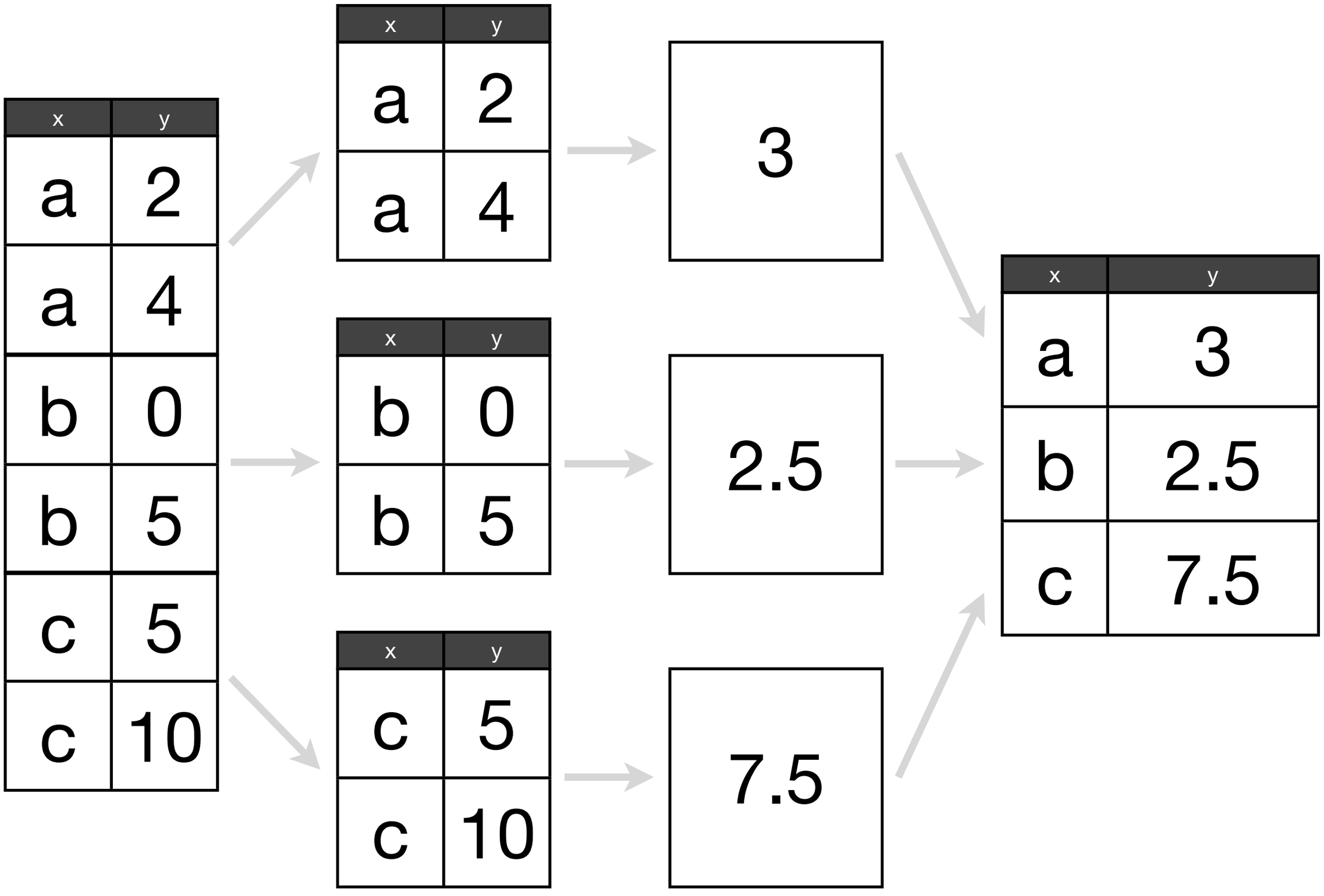
x	y
b	0
b	5

2.5

x	y
c	5
c	10

7.5

Split Apply Combine



```
# 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:

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 `mutate()`

http://plyr.had.co.nz

	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

Tidy data

http://vita.had.co.nz/papers/tidy-data.html

https://vimeo.com/33727555