Statistical Computing (36-350)

split, apply, combine

Cosma Shalizi and Vincent Vu

October 10, 2011

Agenda

- Splitting and aggregating in data analysis
- Examples of the pattern
- 2011 Masters Golf Tournament
- Some tools in base R: split, *apply, *bind
- Recommended reading: *Teetor*, Chap 6

Patterns in programming and data analysis

- Many programming and data analysis problems involve similar types and sequences of actions
- We will study one particular pattern called "**split, apply, combine**" *

** this name is due to H.Wickham (2011)*

Why patterns matter *

- Always keep distinct
  - ***what*** *you want to do*
  - ***how*** *you do it*
- Focusing on **what** brings clarity to intent
- **how** is an important detail, but can easily obscure the high-level problem

* *more on abstraction next week!*

Why patterns matter

- Study and learn to **recognize** the pattern
- Learn **good**, existing solutions

Splitting and aggregating in data analyses

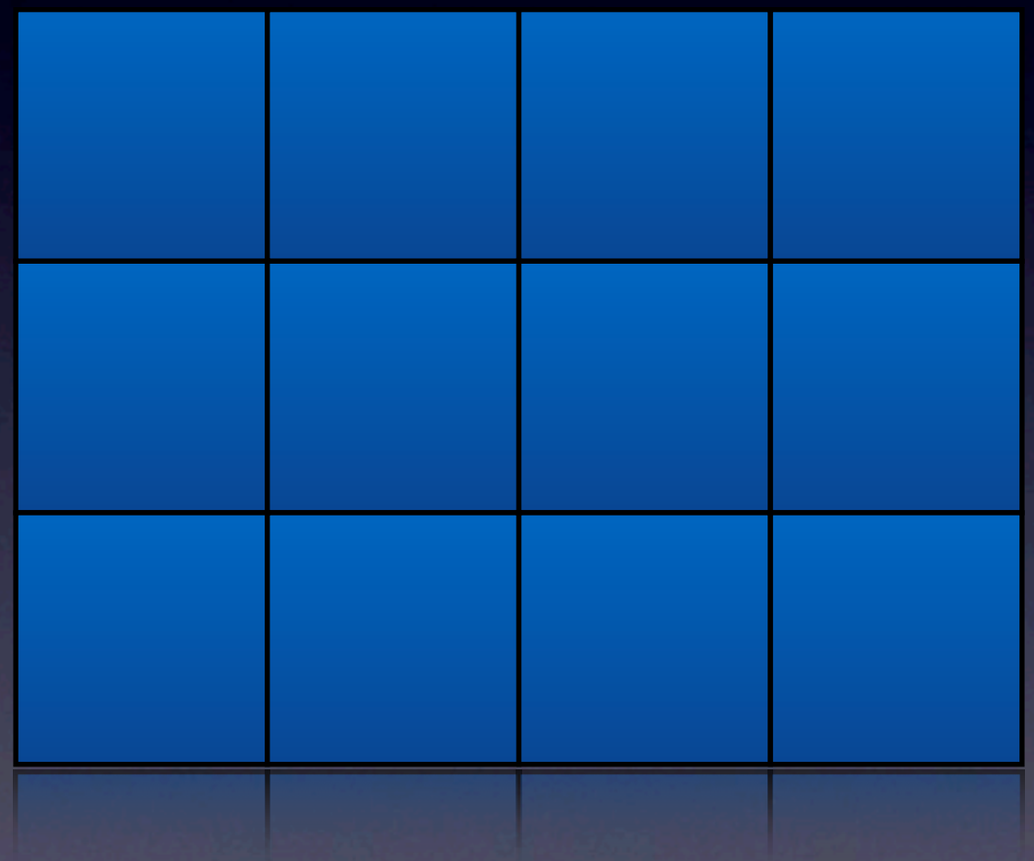
Big, groupable data

- Large datasets usually highly structured
- Data can often be grouped in multiple ways
- Sometimes focus on individual pieces
- Often aggregate information across groups

A primitive example

- Row (column) sums of a matrix
  - Divide the matrix into rows (columns)
  - Compute the sum of each row (column)
  - Combine the results into a vector

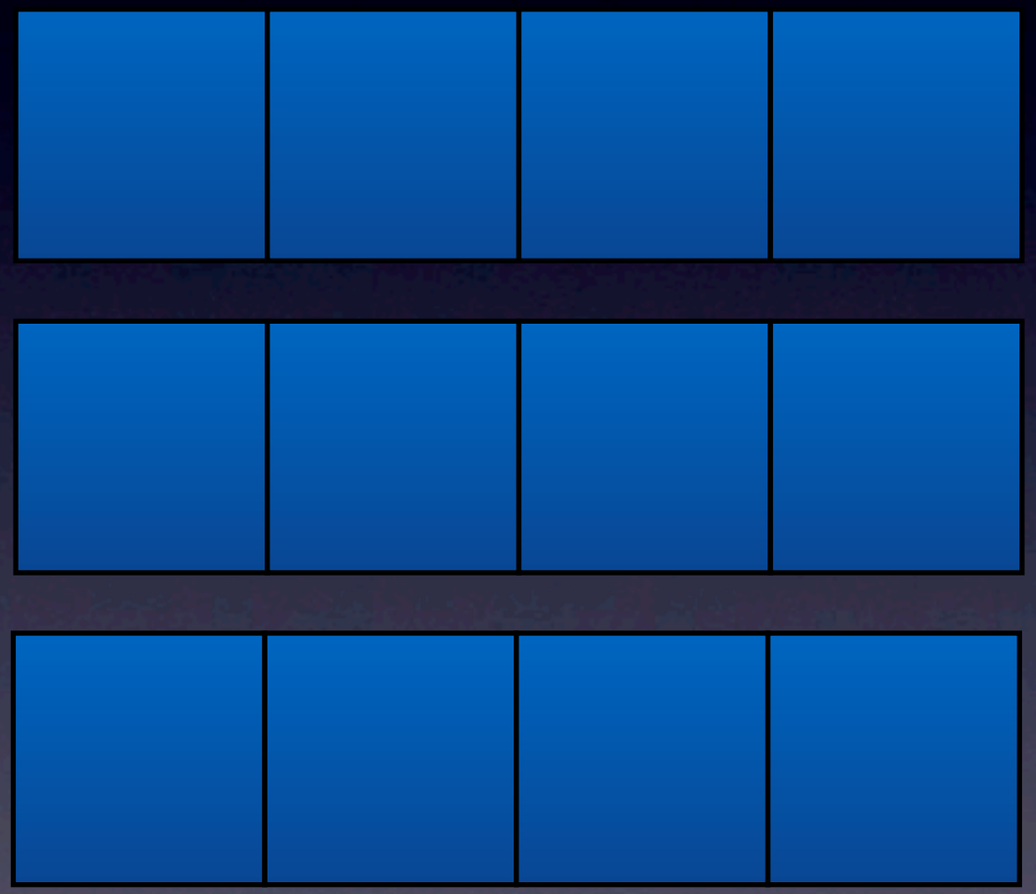
Row sums



`matrix`

(array of dimension 2)

Row sums



Row sums

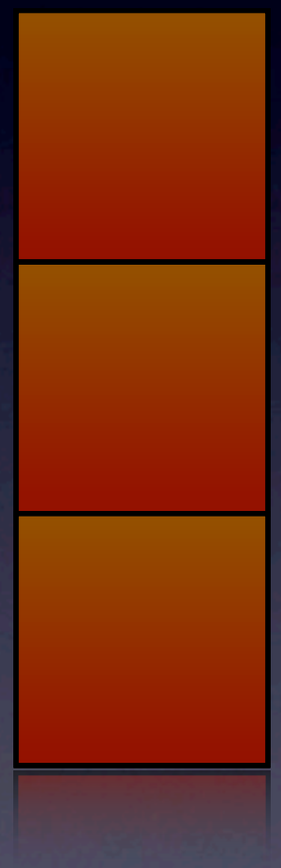
sum()

sum()

sum()

Row sums

Row sums



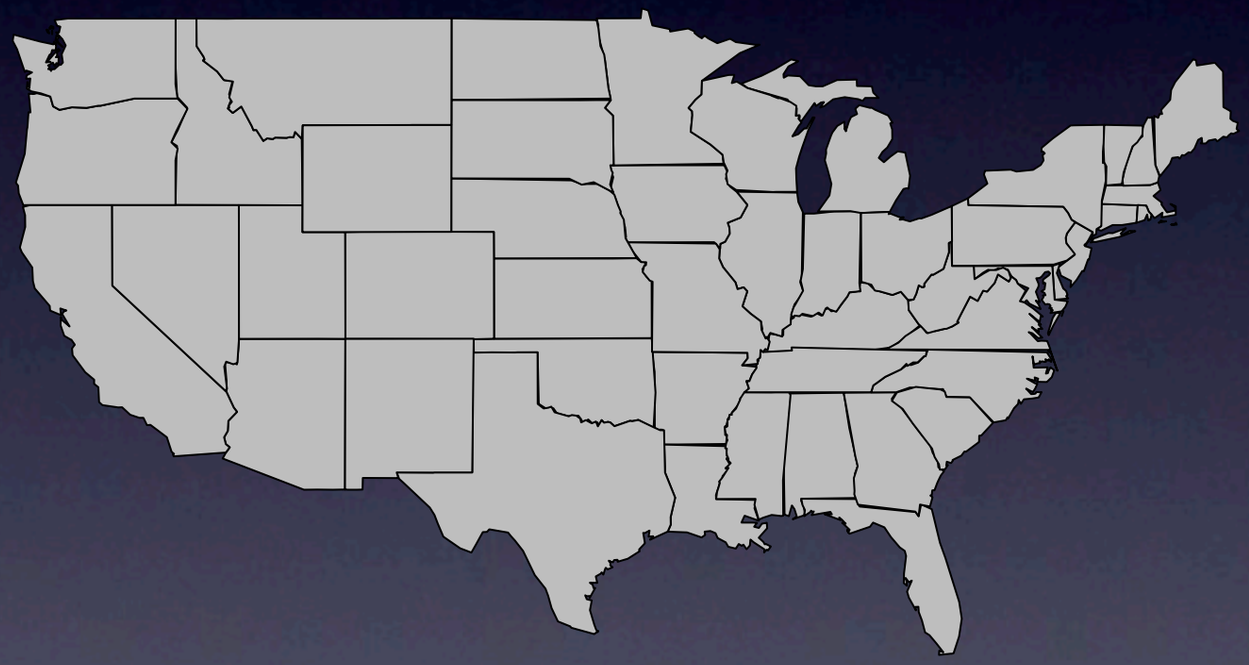
`vector`

(array of dimension 1)

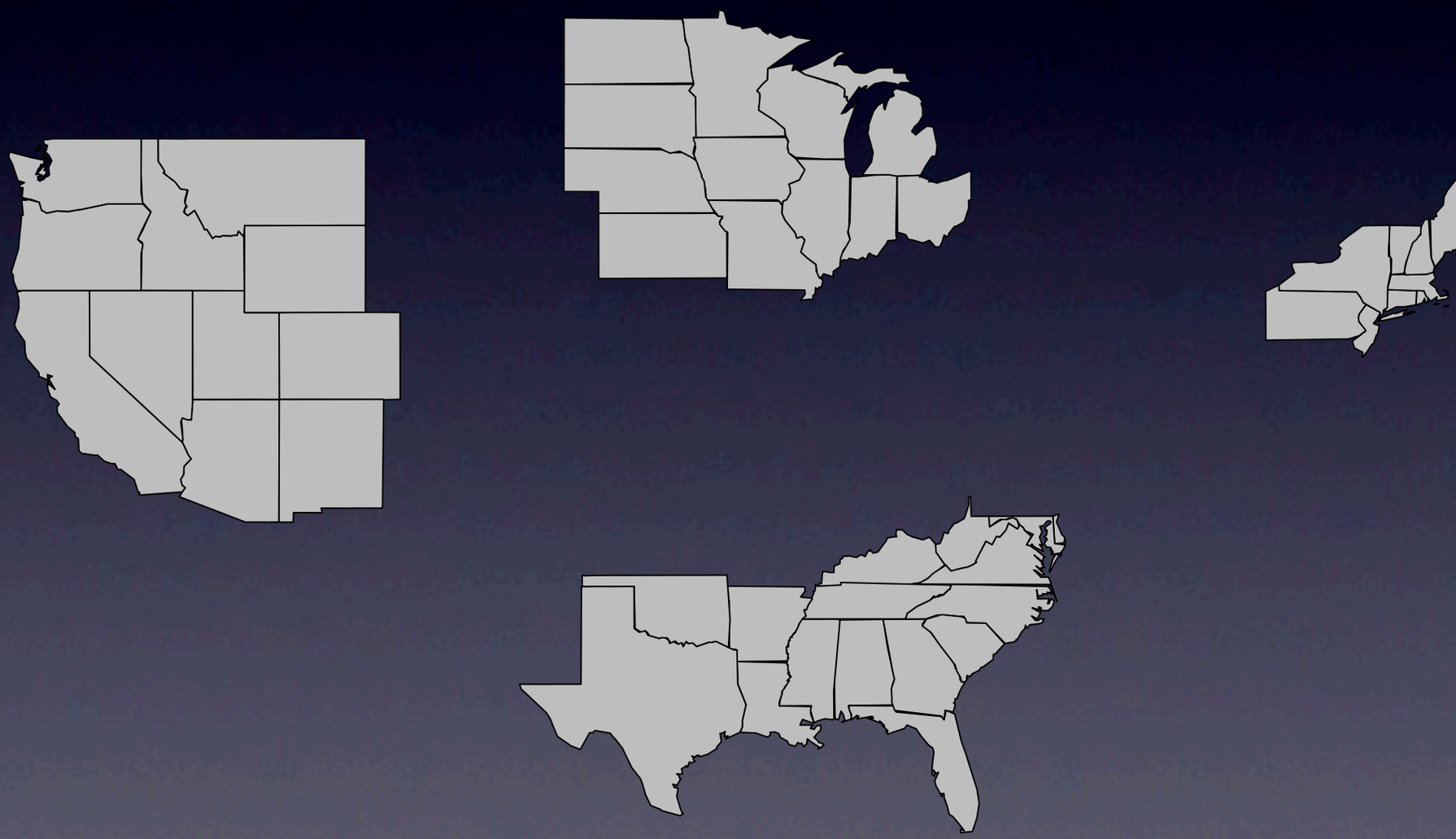
Another example

- Data organized into 48 continental states
- Fit a different model for each of 4 different geographical regions

Splitting by region



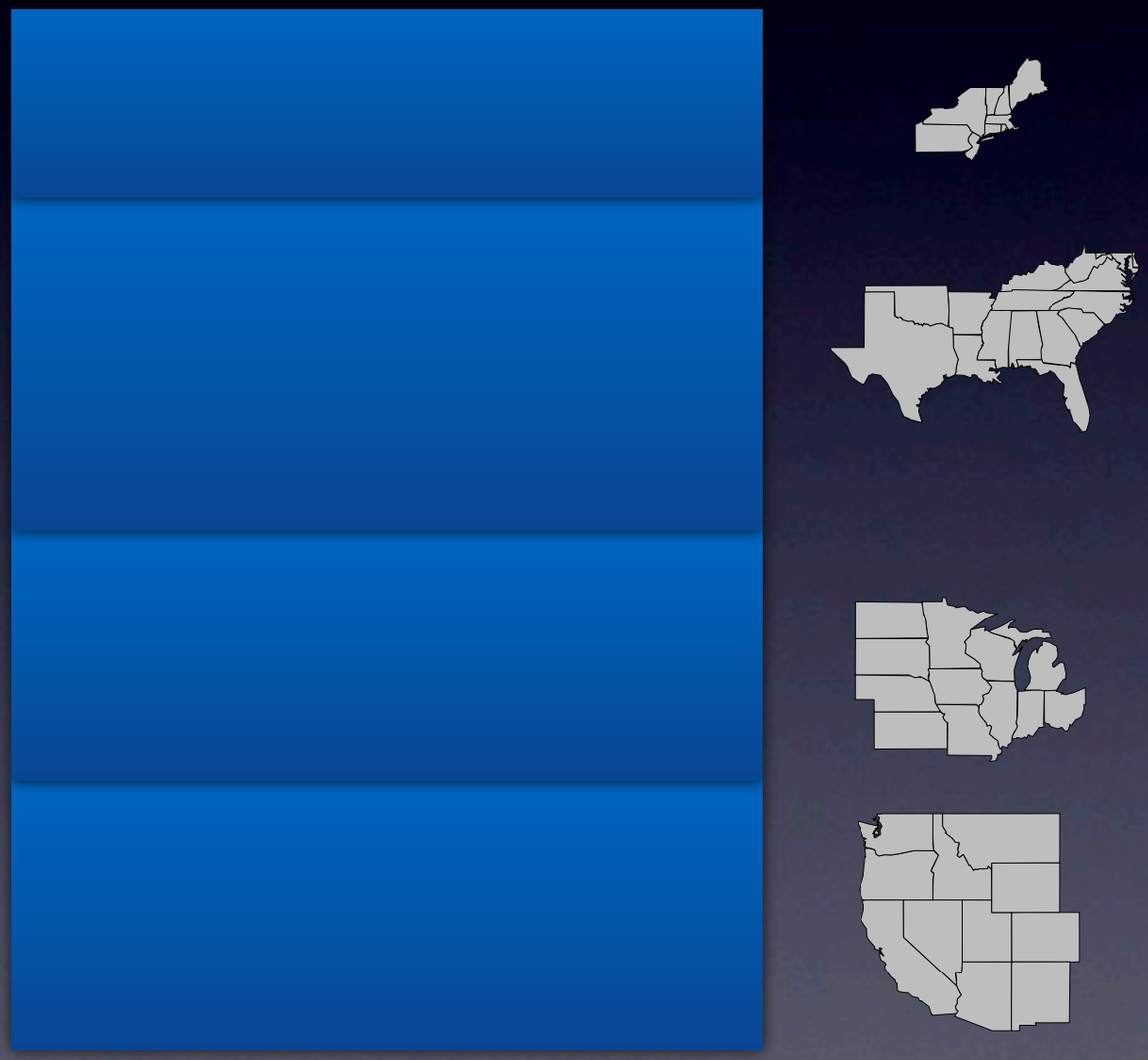
Splitting by region



Splitting by region

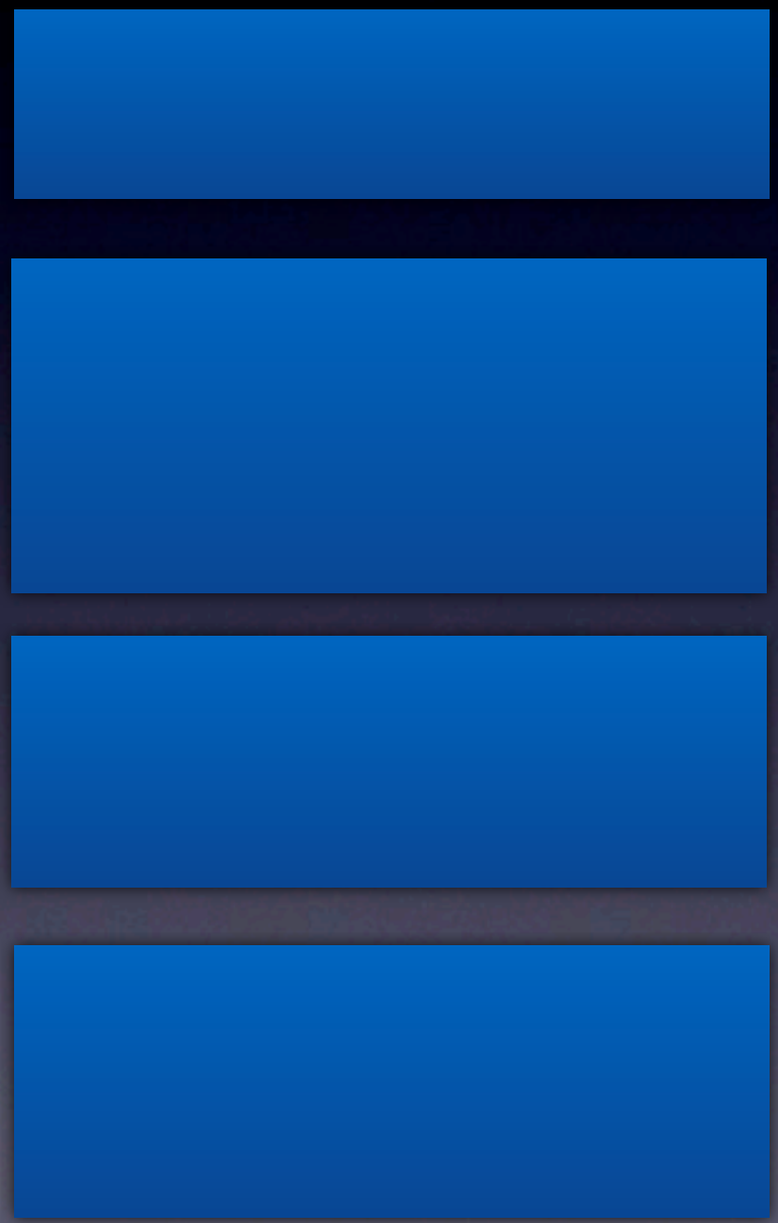
`data.frame`

Splitting by region



`data.frame`

Splitting by region



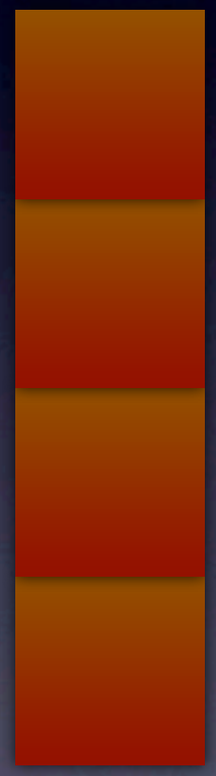
Fitting by region

```
lm(                  )
lm(                  )
lm(                  )
lm(                  )
```

Fitting by region

`lm` objects

Combine into a list



`list` of `lm` objects

The basic pattern

split apply combine

The basic pattern

- Divide the big problem into smaller pieces
- Work on each piece *independently*
- Recombine the pieces

Split, apply, combine

- This is a widely recognized pattern in programming, and many solutions have been developed.

- Examples
  - Python – map(), filter(), reduce()
  - R – split(), *apply(), aggregate(), ...
  - R – plyr package (next time)
  - Google mapReduce

Iteration?

- Possible to use for loops to accomplish the task but they are
  - verbose – too much **how** rather than **what**
  - painful – bookkeeping (indices, placeholders, ...)
  - clumsy – preclude implicit parallelization

SD by location

```
x <- array(..., dim = c(10, 10, 100))
```

- Data:
  - 10 x 10 grid of locations
  - 100 measurements at each location
- Problem:
  - Compute sample SD at each location

SD by location

iteration

```
y <- array(dim = dim(x)[1:2])
for(i in 1:dim(x)[1]) {
    for(j in 1:dim(x)[2]) {
        y[i, j] <- sd(x[i, j, ])
    }
}
```

apply

```
y <- apply(x, 1:2, sd)
```

apply()

```
y <- apply(X, MARGIN, FUNCTION, ...)
```

- **X** an array
- **MARGIN** vector of subscripts which the function will be applied over
- **FUNCTION** the function to be applied
- **...** additional arguments to function
- Returns an array (or a list)

apply()

```
y <- apply(x, c(1, 3), f)
```

Compute `f(x[i, , j, ])` for all `i`, `j`

```
y <- apply(x, 2:4, f)
```

Compute `f(x[ , i, j, k, ])` for all `i`, `j`, `k`

*apply()

- Examples
  - `apply()` for arrays
  - `lapply()` for lists
  - `mapply()` for multivariate functions
- Consult textbooks and R help for details

But...

- What about ragged data – different numbers of observations at each location?
- What about more complex situations?

2011 Masters Golf Tournament

PAR 5 - 570 YARDS

8TH TIGER WOODS -8

2ND SHOT

April 7–10, 2011 | Augusta National Golf Club, Augusta, Georgia

2011 MASTERS TOURNAMENT

HOME NEWS SCORING PLAYERS MULTIMEDIA COURSE HISTORY

BUILD YOUR CUSTOM LEADERBOARD BY CLICKING THE + BUTTON OR SEARCHING.

			SCORING TO PAR			ROUNDS				
POS		PLAYER NAME: FIRST | LAST	TOTAL	THRU	TODAY	1	2	3	4	TOTAL
1	↑1	Charl Schwartzel	-14	F	-6	69	71	68	66	274
T2	-	Jason Day	-12	F	-4	72	64	72	68	276
T2	↑4	Adam Scott Titleist	-12	F	-5	72	70	67	67	276
T4	↑5	Tiger Woods	-10	F	-5	71	66	74	67	278
T4	↑5	Geoff Ogilvy Titleist	-10	F	-5	69	69	73	67	278
T4	↑2	Luke Donald Titleist	-10	F	-3	72	68	69	69	278
7	↓5	Angel Cabrera Titleist	-9	F	-1	71	70	67	71	279
T8	-	Bo Van Pelt Titleist	-8	F	-2	73	69	68	70	280
T8	↓6	K.J. Choi Titleist	-8	F	E	67	70	71	72	280

April 7–10, 2011 | Augusta National Golf Club, Augusta, Georgia

2011 MASTERS TOURNAMENT

Charl Schwartzel

South Africa

Select another player

-- Select A Player --

« back to leaderboard

Start Position		Current Position
T2	1	1

Total Strokes: 69 - 71 - 68 - 66 : 274 To Par: -14

Scorecard | Stats | Profile

Updated: Monday, April 11, 12:00 AM

Round 4

EAGLE BIRDIE PAR BOGEY DOUBLE BOGEY +

HOLE	1	2	3	4	5	6	7	8	9	OUT	10	11	12	13	14	15	16	17	18	IN	TOT
YARDS	445	575	350	240	455	180	450	570	460	3725	495	505	155	510	440	530	170	440	465	3710	7435
Par	4	5	4	3	4	3	4	5	4	36	4	4	3	5	4	5	3	4	4	36	72
Score	3	5	2	4	4	3	4	5	4	34	4	4	3	5	4	4	2	3	3	32	66
To Par	-1	-1	-3	-2	-2	-2	-2	-2	-2		-2	-2	-2	-2	-2	-3	-4	-5	-6		-6

Scorecard Data

- Hole-by-hole scores scraped from www.majorschampionships.com
- Organized into a data frame of 21 columns
  - hole 1 – 18 scores, round, player, country
  - 296 rows

Scorecard Data

holes 1–18	round	player	country
...	4	Charl S.	South Africa
...	3	Charl S.	South Africa
...	2	Charl S.	South Africa
...	1	Charl S.	South Africa
...	4	Jason D.	Australia
...	3	Jason D.	Australia
...	...	...	...

Plan

- Look at performance of an individual
- Encapsulate the analysis into a function
- **Split** the data by player
- **Apply** the function to each player
- **Combine** the results

How was Tiger Woods?

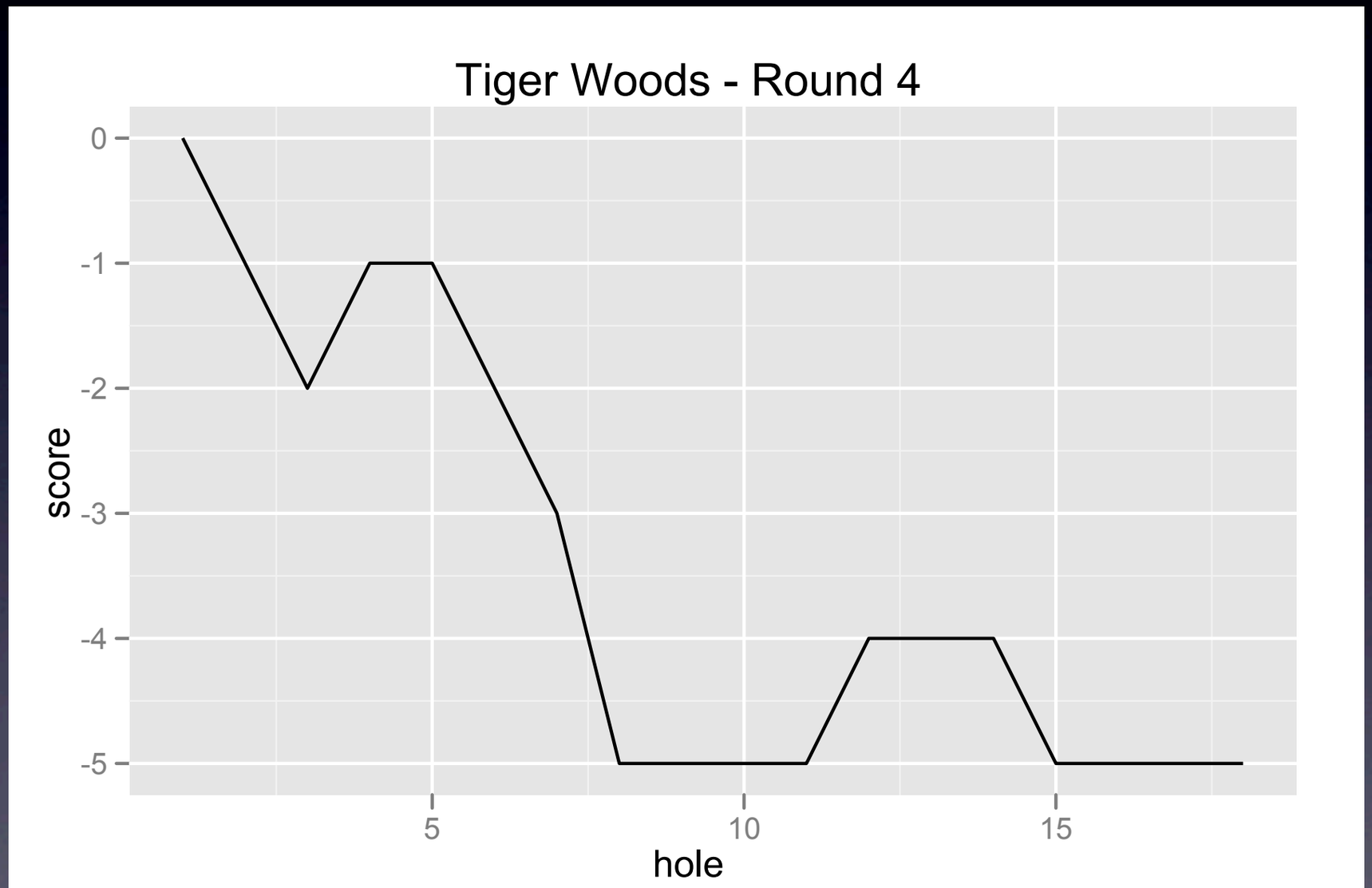
holes 1–18	round	player	country
...	4	Tiger W.	USA

Extract Tiger's scores in the 4th round and calculate his running total to par:

```
df <- subset(masters2011$scorecard,
             player == 'Tiger Woods' & round == 4)
tiger <- as.numeric(df[, 1:18])
tiger <- cumsum(tiger - masters2011$course$par)

plot(tiger, type = '1')
```

How was Tiger Woods?



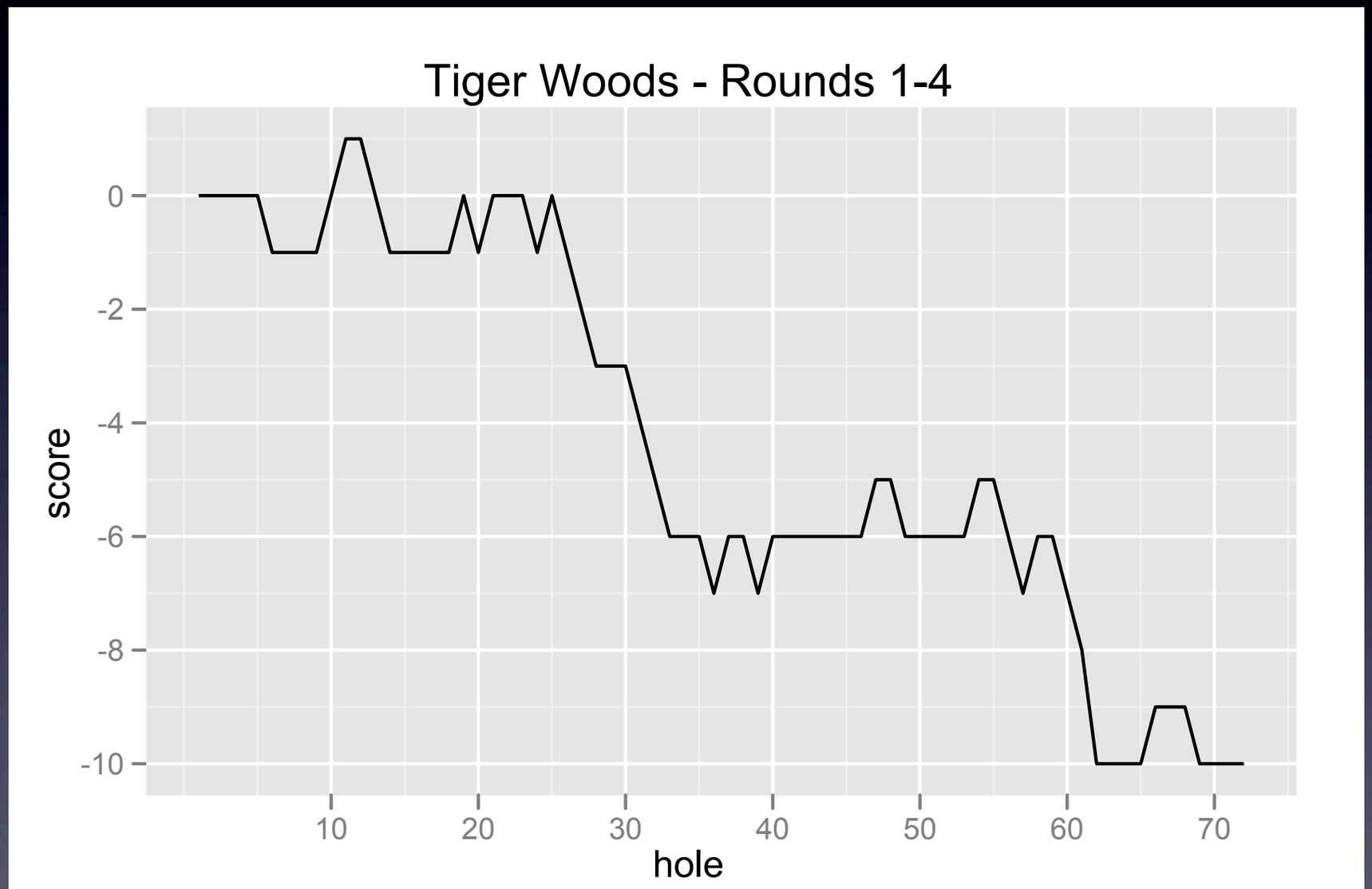
How was Tiger Woods?

Extract Tiger's scores for all 4 rounds and calculate his running total to par:

```
tiger <- subset(masters2011$scorecard,
                player == 'Tiger Woods')
tiger <- as.matrix(tiger[order(tiger$round), 1:18])
# Convert row-wise from a matrix to vector rowwise
tiger <- as.vector(t(tiger))
tiger <- cumsum(tiger - rep(masters2011$course$par, 4))

plot(tiger, type = 'l')
```

How was Tiger Woods?



How was X?

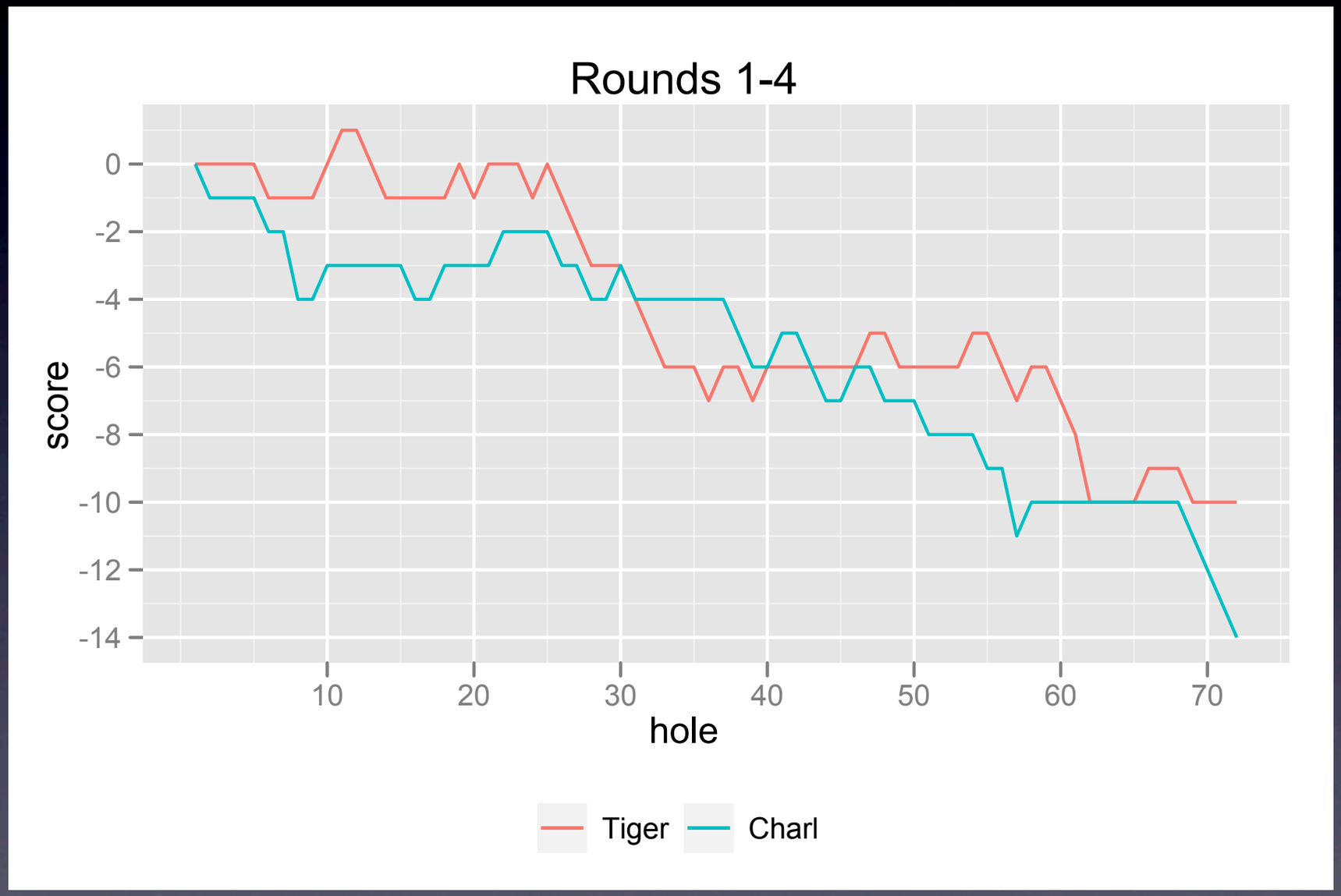
```
runningTotal <- function(df, par = masters2011$course$par)
{
    # Reorder the rows of the data frame by round
    # (so that the scores are in chronological order)
    # and extract the scores as a matrix
    x <- as.matrix(df[order(df$round), 1:18])
    n <- nrow(x)

    # Convert from an 18 x n matrix to a vector of
    # length 18*n, row-wise
    x <- as.vector(t(x))

    # Calculate the running over/under score
    x <- cumsum(x - rep(par, n))

    return(x)
}
```

How was Charl Schwartzel?




```
charl <- runningTotal(subset(masters2011$scorecard,
                             player == 'Charl Schwartzel'))
```

Splitting the data frame

holes 1–18	round	player	country
...	4	Charl S.	South Africa
...	3	Charl S.	South Africa
...	2	Charl S.	South Africa
...	1	Charl S.	South Africa
...	4	Jason D.	Australia
...	3	Jason D.	Australia
...	...	...	...

Splitting the data frame

holes 1–18	round	player	country
...	4	Charl S.	South Africa
...	3	Charl S.	South Africa
...	2	Charl S.	South Africa
...	1	Charl S.	South Africa
...	4	Jason D.	Australia
...	3	Jason D.	Australia
...	...	...	...

Split the data frame

```
df <- masters2011$scorecard
x <- split(df, df$player)
```

- **`$player`** is a factor vector: players are levels of the factor
- split the data frame according to the levels of **`$player`**
- **`x`** is a list of data frames

Apply `runningTotal()`

```
x <- lapply(x, runningTotal)
```

- Apply **`runningTotal()`** to each element of **`x`**
- Result is a list of vectors

Combine the vectors into an array

```
scores <- do.call(rbind, x)
```

- Equivalent to

  `rbind(x[[1]], x[[2]], ...)`

- *Note*: vectors in **`x`** have to be of same lengths

All together

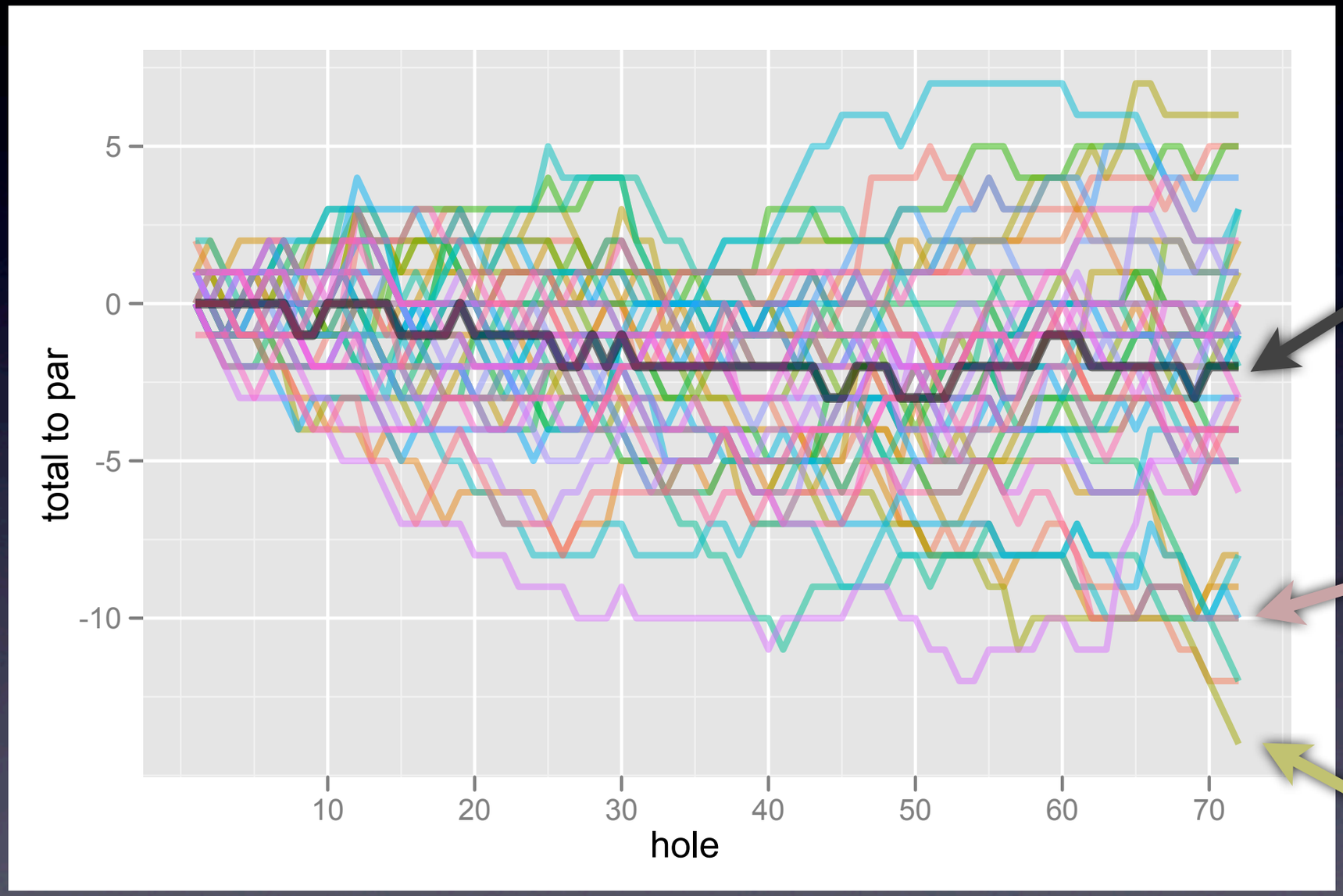
split, apply, combine using base R

```
x <- split(df, df$player)
x <- lapply(x, runningTotal)
scores <- do.call(rbind, x)
```

iteration

```
scores <- matrix(nrow = nlevels(df$player),
                 ncol = 18 * 4)
for(i in 1:nlevels(df$player)) {
    x <- subset(df, player == levels(df$player)[i])
    scores[i, ] <- runningTotal(x)
}
rownames(scores) <- levels(df$player)
```

Masters 2011



median golfer

```
matplot(t(scores), xlab='hole',ylab='total to par', type='l')
lines(1:ncol(scores), apply(scores, 2, median), lwd = 2)
```

Summary

- The split, apply, combine pattern appears in many problems
- Recognize it
- iteration (with for loops) is usually not a good solution to the problem
- Next: abstracting the pattern and using the plyr package