# Statistical Computing (36-350)

## Lecture 11: Split/Apply/Combine with Base R

Cosma Shalizi

Massive thanks to Vince Vu

8 October 2012

# Agenda

- Splitting and aggregated for data analysis
- Examples of the pattern
- Unemployment and strikes across countries
- Tools in base R: `subset`, `split`, `*apply`, `*bind`, `do.call`

READING: *The R Cookbook*, chapter 6; Matloff, chapter 6

Lots of problems in programming and data analysis can be solved by similar types and sequences of actions

**Design patterns** and **Analysis patterns**

We will look at the pattern called “split, apply, combine” (Hadley Wickham)

# The Point of Learning Patterns

Distinguish between **what** you want to do and **how you want to do it**
Focusing on **what** brings clarity to intentions
**How** also matters, but can obscure the high-level problem
Learn the pattern, recognize the pattern, love the pattern
Re-use *good* solutions

# Splitting and Aggregation in Data Analysis

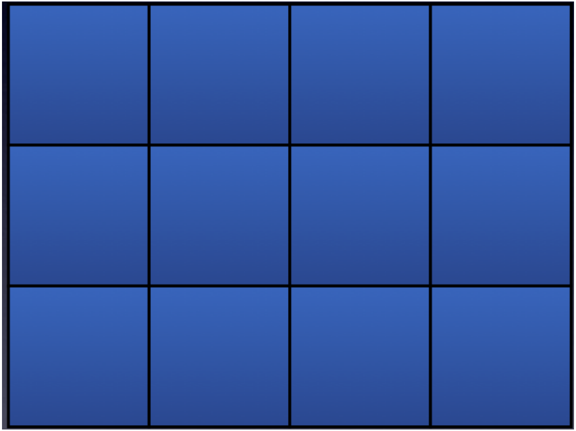
Large data sets are usually highly structured
Structure lets us group data in many different ways
Sometimes we focus on individual pieces of data
Often we aggregate information within groups, and compare across them

# A Trivial Example

Row (column) means of a matrix

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

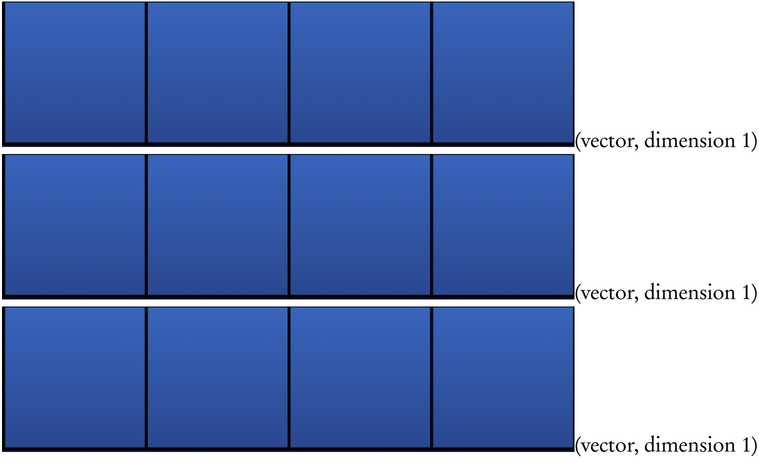
# Row Means



`matrix`

(an array of dimension 2)

# Row Means

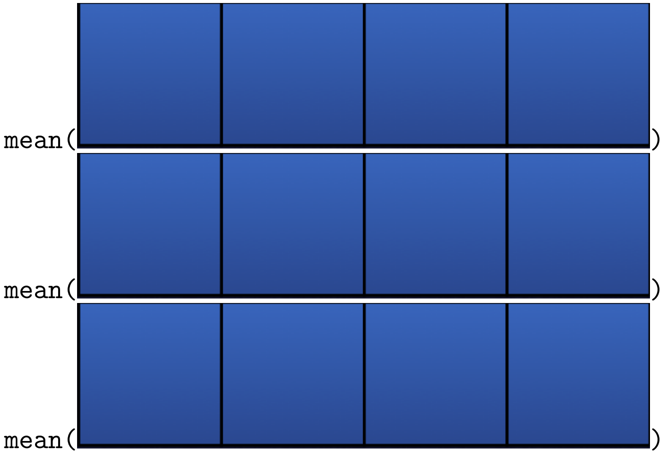


(vector, dimension 1)

(vector, dimension 1)

(vector, dimension 1)

# Row Means

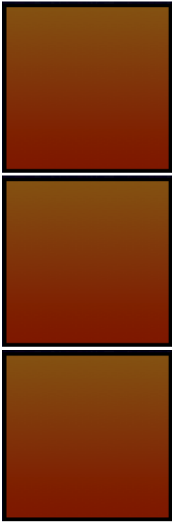


mean( )

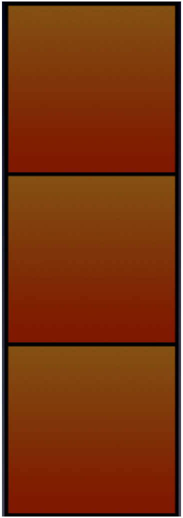
mean( )

mean( )

# Row Means



# Row Means



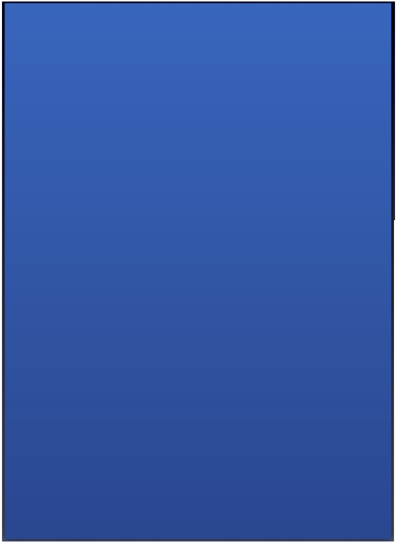
vector (of dimension 1)

# Another Example

Data organized into 48 continental states

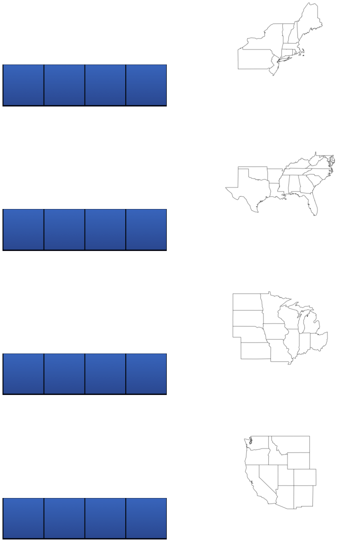
Fit a different model for each of 4 different geographic regions

# Splitting by Region



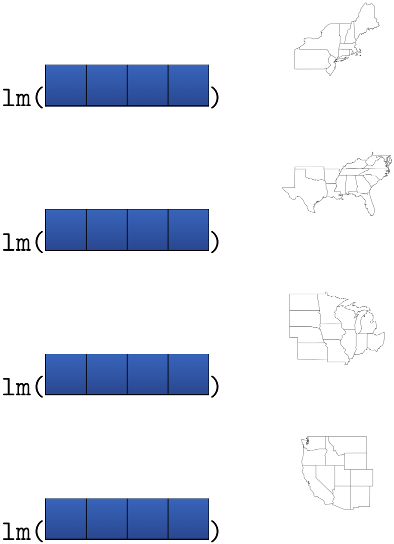
`data.frame`

# Splitting by Region



data.frames

# Splitting by Region



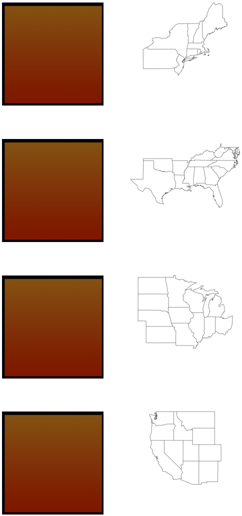
lm( )

lm( )

lm( )

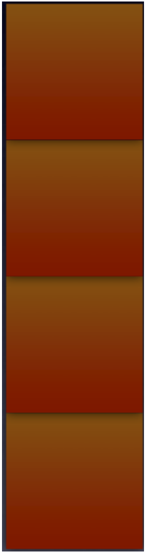
lm( )

# Splitting by Region



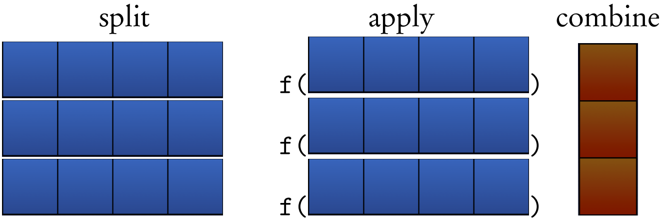
`lm` objects

# Combine into a list



`list` of `lm` objects

# The Basic Pattern



# The Basic Pattern (cont'd.)

| | |
|---|---|
| Split | divide the problem into smaller pieces |
| Apply | Work on each piece independently |
| Combine | Recombine the pieces |

A common pattern for both programming and data analysis, many implementations

Python: map(), filter(), reduce()

Google `mapReduce`

R: `split`, `*apply`, `aggregate`,. . .

R: `plyr` package

# Iteration Considered Unhelpful

Could always do the same thing with `for` loops, but those are

- verbose — lots of "how", obscures "what"
- painful/error-prone book-keeping (indices, placeholders, . . . )
- clumsy — hard to parallelize

# SD by Location

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

Data: 10 × 10 grid of locations, 100 measurements / location
Desired: sample SD at each location

# SD by Location

Iteration:

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

`apply`:

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

# apply()

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

X an array

MARGIN vector of subscripts which the function is applied over

FUNCTION the function to be applied

... additional arguments to function (held constant)

Returns an array if it can, a list if all else fails

# `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()

Variants for different data structures:

- apply() for arrays
- lapply() and sapply() for lists and vectors
- mapply() for multivariate functions

Consult textbooks and R help for details

# But. . .

What about ragged data — different numbers of observations at each location?
More complex situations?

# Politics and Labor Action

Does having a friendlier government make labor action more or less likely?

WE MARCH FOR JOBS FOR ALL NOW!
WE DEMAND VOTING RIGHTS NOW!
END SEGREGATED RULES IN PUBLIC SCHOOLS
WE DEMAND EQUAL RIGHTS NOW!

March on Washington, 1963

Madison protests, 2011

# Political Economy of Strikes Data

Compiled by Prof. Bruce Western at Harvard

Data frame of 8 columns

country, year, days on strike per 1000 workers, unemployment, inflation, left-wing share of gov't, centralization of unions, union density

"centralization" not useful to us so we'll drop it

625 observations from 18 countries, 1951–1985

$18 \times 35 = 630 > 625$, ∴ some years missing from some countries

# A Little Bit of the Data

| country | year | strike.volume | unemployment | inflation | left.parliament | density |
|---|---|---|---|---|---|---|
| Australia | 1983 | 313 | 9.8 | 10.1 | 60 | 48.5 |
| Australia | 1984 | 241 | 8.9 | 4 | 55.4 | 47.6 |
| Australia | 1985 | 226 | 8.2 | 6.7 | 55.4 | 45.9 |
| Austria | 1951 | 43 | 3.5 | 27.5 | 43.6 | NA |
| Austria | 1952 | 39 | 4.7 | 13.6 | 43.6 | NA |
| Austria | 1953 | 20 | 5.8 | -1.6 | 46.7 | NA |

# Plan

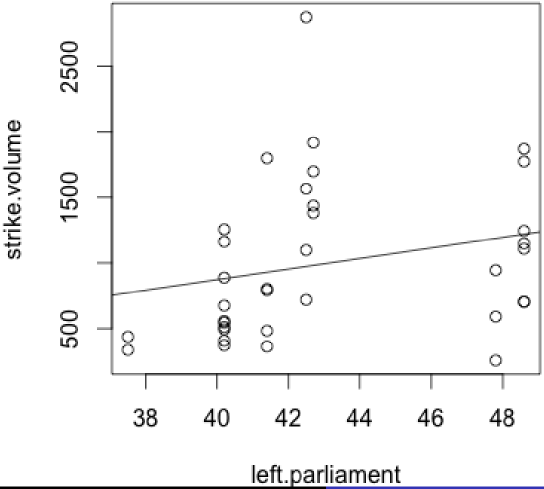
- Look at the relation between strikes and left-wing parties for a country
- Encapsulate the analysis into a function
- **Split** the data by country
- **Apply** the function to each country
- **Combine** the results

# Italy, For Example

| country | year | strike.volume | unemployment | inflation | left.parliament | density |
|---|---|---|---|---|---|---|
| Italy | 1973 | 1698 | 6.2 | 10.8 | 42.7 | 43.3 |
| Italy | 1974 | 1381 | 5.3 | 19.1 | 42.7 | 46.2 |
| Italy | 1975 | 1918 | 5.8 | 17 | 42.7 | 48 |

```
df <- subset(strikes, country=="Italy")
italy <- lm(strike.volume ~ left.parliament, data=df)
plot(strike.volume ~ left.parliament, data=df)
abline(italy)
```

# Italy, For Example



# What About Country X?

```
strikes_vs_left <- function(df,coefficients.only=FALSE) {
  fit <- lm(strike.volume ~ left.parliament, data=df)
  if (coefficients.only) {
   return(coefficients(fit))
  } else {
    return(fit)
  }
}
```

How about Belgium?

```
belgium <- strikes_vs_left(subset(strikes,country=="Belgium"))
```

EXERCISE: Make a plot like the one for Italy

# Split the data frame

```
x <- split(strikes, strikes$country)
```

`$country` is a factor vector: countries are levels of the factor
split the data frame according to the levels of `$country`
x is a list of data frames

# Apply strikes_vs_left()

```
y <- lapply(x, strikes_vs_left, coefficients.only=TRUE)
```

Apply `strikes_vs_left()` to each element of `x`
Result is a list of coefficient vectors

Turning off `coefficients.only` would give a list of `lm` model objects

# Combine the vectors into an array

```
coefs <- do.call(rbind, y)
```

Equivalent to

```
rbind(y[[1]], y[[2]], ... y[[18]])
```

but don't have to know how long y is

Vectors bound together have to be of the same length

# All Together

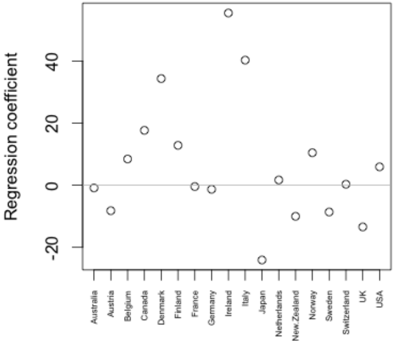
## split, apply, combine, using only base R

```
x <- split(strikes, strikes$country)
y <- lapply(x, strikes_vs_left, coefficients.only=TRUE)
coefs <- do.call(rbind, y)
```

## Iteration

```
coefs <- matrix(nrow=nlevels(strikes$country),ncol=2)
for (i in 1:nlevels(strikes$country)) {
  x <- subset(strikes, country==levels(strikes$country)[i])
  coefs[i,] <- strikes_vs_left(x,coefficients.only=TRUE)
}
rownames(coefs) <- levels(strikes$country)
```

EXERCISE: replace `subset()` with more iteration



```
plot(coefs[,2],xaxt="n",xlab="",ylab="Regression coefficient")
axis(side=1,at=seq(along=rownames(coefs)),labels=rownames(coefs),
  las=2,cex.axis=0.5)
abline(h=0,col="grey")
```

Lots of (apparent) heterogeneity across countries

Actual differences across countries might be conflated with different economic circumstances: try adding covariates to the regression

Arranging countries alphabetically is uninformative — maybe by geography or cultural groupings?

EXERCISE: Re-arrange so all English-speaking countries are on the far right

Really should have error bars if we're going to compare

EXERCISE: Modify code to return standard errors for coefficients, use `segments` to add ±2se error bars to each point estimate

# Summary

The split, apply, combine pattern is very common

Recognize it!

Iteration is usually not a good solution

`*apply` is usually a better solution

Next time: abstracting the pattern with the `plyr` package