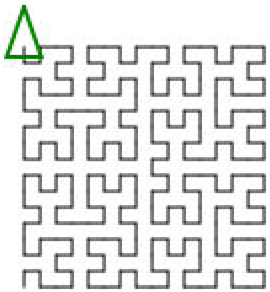
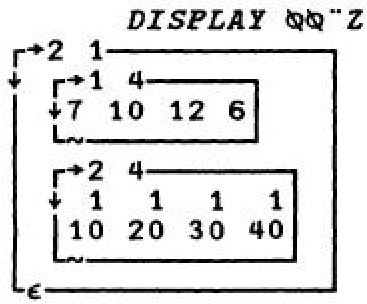
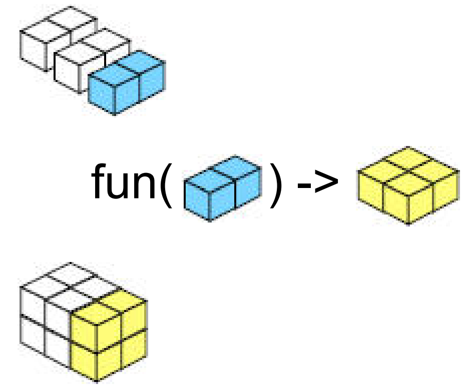





```
proc glm data=class;
  class sex;
  model wt = ht sex;
  output out=results
    p=predict r=resid;
```



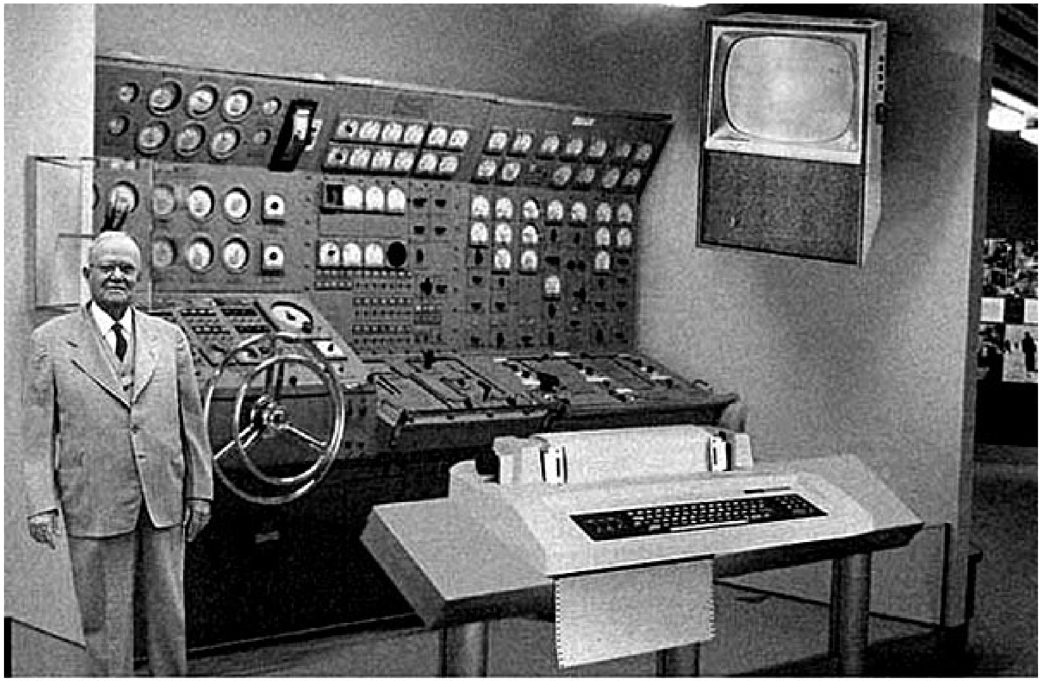
Tools for *Thinking* in Statistical Computation and Graphics: A 40-year journey from APL to SAS to R

Michael Friendly

QM Brownbag Seminar

November, 2010

Prelude: First "home computer" (1957)



Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.

RAND Corp. predicted what a home computer would look like in 2004.

"With a teletype interface and the Fortran language, the computer will be easy to use."

Me: *Dad, can we get one?*

Dad:

What's the steering wheel for?

Why is IKE in that picture?

Who speaks Fortran?

Prelude: First summer job (1962)

- Test department, Harcourt, Brace & World
- Job: Calculate 45 correlations among 10 tests, for n=500
- Tool: Monroe calculator
- Insight: There has to be a better way
  - n, $\sum x$, $\sum y$ $\sum x^2$, $\sum y^2$, $\sum xy$ can be calculated on a single pass
- Gripes:
  - Don't they have an IT dept?
  - I could write this in Fortran!

Method:

enter X,Y:	10		5
square:	100	2500	25
sum:	$\sum x$, $\sum x^2$	$\sum xy$	$\sum y$, $\sum y^2$

Programming languages: Power & elegance

- **CS view**: All programming languages can be proved to be equivalent (to a Turing machine)
- **Cognitive view**: Languages differ in
  - expressive power: ease of translating what you want to do into the results you want
  - elegance: how well does the code provide a human-readable description of what is done?
  - extensibility: ease of generalizing a method to wider scope
  - learn-ability: your learning curve (rate, asymptote)

Programming languages: Power & elegance

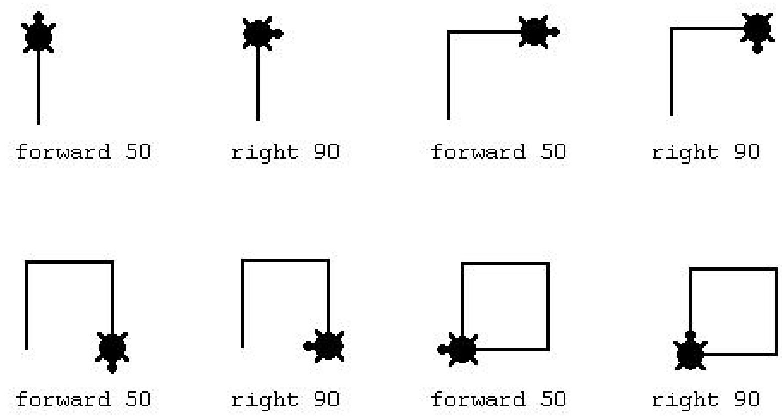
Language	Features:Tools for thinking?
FORTRAN	Subroutines – reusable code Subroutine libraries (e.g., BLAS)
APL, *APL2STAT*	N-way arrays, nested arrays Generalized reduction, outer product Function operators
Logo	Turtle graphics Recursion, list processing
Lisp, LispStat, *ViSta*	Object-oriented computing Functional programming
Perl	Regular expressions Search, match, transform, apply
SAS	??
R	??

Programming languages: Elegance - *Logo*

- Features:
  - Based on Lisp, but tuned to young minds
    - Papert: *Mindstorms: Children, Computers, and Powerful Ideas* (1980)
  - Turtle graphics: draw by directing a turtle, not by (x,y) coordinates
    - Analytic geometry rests on a coordinate system.
    - Turtle geometry is "body syntonic“: Tell turtle what to do.
  - Data types: words, lists, arrays, property lists
  - Lists & list processing: inherited from Lisp, but with gentler syntax. Lists are infinitely expandable & nestable.
  - Recursion rather than iteration is the natural method to process lists
  - Extensions:
    - multiple, animated turtles (sprites);
    - object-oriented programming (message passing) -> SmallTalk

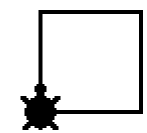
Logo : Turtle graphics

Turtle primitives: forward, back, left, right, penup, pendown, ...



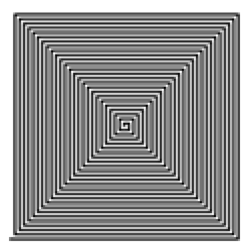
Logo procedures: teach the turtle a new word

```
> to square :side
  repeat 4 [fd side rt 90]
  end

> square 100
```

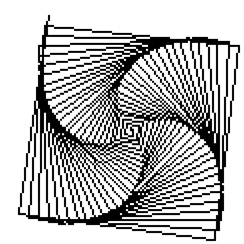


Recursive procedures:

```
to spiral :size :angle
  if :size > 100 [stop]
  forward :size
  right :angle
  spiral (:size + 2) :angle
  end
```

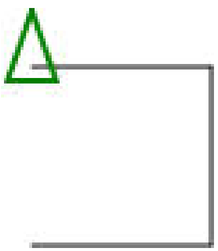
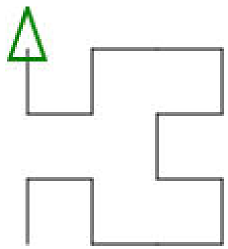


spiral 0 90



spiral 0 91

Logo : Hilbert curves

Logo was more than just pretty pictures

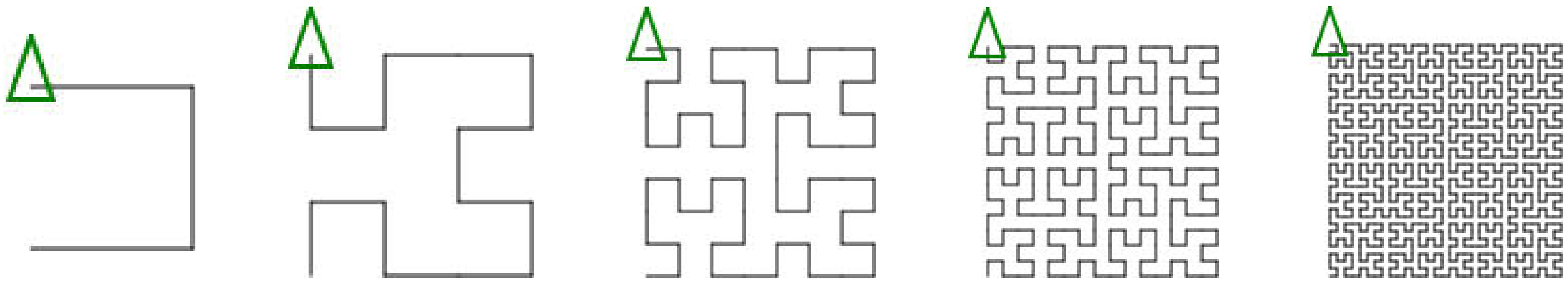
```
to Hilbert0 :turn :size
 right :turn
 forward :size
 left :turn
 forward :size
 left :turn
 forward :size
 right :turn
end
```

Start with some basic shape

What happens if you replace each line with a smaller copy of the basic shape?

What happens if you continue this process?

Logo : Hilbert curves



```
to Hilbert :depth :turn :size
 if :depth = 0 [stop]
 right :turn
  Hilbert (:depth-1) -:turn :size
 forward :size
 left :turn
  Hilbert (:depth-1)  :turn :size
 forward :size
  Hilbert (:depth-1)  :turn :size
 left :turn
 forward :size
  Hilbert (:depth-1) -:turn :size
 right :turn
end
```

Hilbert curve: A continuous, space-filling fractal, of Hausdorff dimension 2

Theorem (Hilbert, 1891): The euclidean length of the n-th depth Hilbert curve, H_n is $2^n - \frac{1}{2^n}$

Proof (by enumeration): Redefine forward to calculate total turtle path length

```
to forward.length :size
  make "total.length :total.length + :size
  forward :size
  end
```

Programming languages: Power - *APL*

- Quotes:
  - *APL2 is arguably the most powerful language yet developed for expressing statistical computation. One's ability to get work done, however, depends as much on the programming environment as on the primitives of the language.*

    Friendly & Fox, *JCGS*, 1994
  - *APL is the most powerful notation for array processing ever invented. Because of its lack of influence on other languages, it will not be discussed further.*

    Cal. Board of CS Curricula: Brown & Wheeler, *APL'2002*, 2002
  - APL as "write-only" language: *Saying "powerful language" is just a friendlier way of saying "obfuscated syntax".*

    Jim Lehmer

```
      (2=+⌿0=I∘.|I)/I←⍳50     ⍝ prime numbers from 1..50
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

APL: Notation as a tool for thought

- *By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems*

  A.N. Whitehead

  - e.g., matrix algebra (Arthur Cayley, ~1850): $\mathbf{A\,x = b} \rightarrow \mathbf{x = A^{-1}\,b}$
- Characteristics of computational notation (K. E. Iverson):
  - **Universality**: Any problem in executable form → unambiguous result
  - **Ease** of expressing constructs arising in problems
  - **Suggestivity**: expressions in one set of problems suggest others for application in other problems
  - **Subordination** of detail, e.g., vectors → n-way arrays, named functions
  - **Economy**: Utility of a language as a tool for thought increases with the range of topics it can treat, but decreases with the size of vocabulary and complexity of grammatical rules the user must keep in mind
  - Amenability to **formal proofs**: Extent to which notation facilitates proofs (by induction, exhaustion, etc.)

APL2, APL2STAT Features

Many features, but three most important for statistical computation and graphics:

- n-way arrays, nested arrays
- implicit iteration with data and each (¨)
- operators: functions of functions

APL2, APL2STAT Features

- n-way arrays, nested arrays

  e.g. partitioned matrices

$$\mathbf{Z} = \left(\mathbf{y} \;\vdots\; \mathbf{X}\right) \qquad\qquad \mathbf{Z'Z} = \left(\mathbf{y} \;\vdots\; \mathbf{X}\right)'\left(\mathbf{y} \;\vdots\; \mathbf{X}\right) = \left[\begin{array}{c:c} \mathbf{y'y} & \mathbf{y'X} \\ \hdashline \mathbf{X'y} & \mathbf{X'X} \end{array}\right]$$

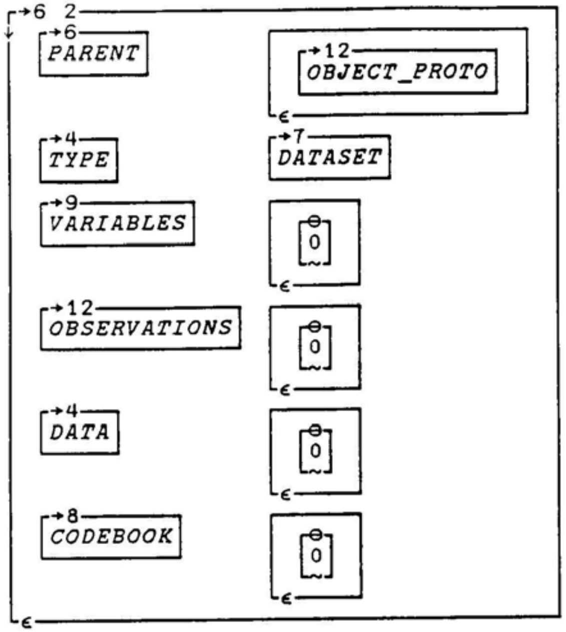
```
      DISPLAY Z
┌→1 2─────────────┐
↓ ┌→4┐  ┌→4 2┐    │
│ ↓ 7│  ↓1 10│    │
│ │10│  │1 20│    │
│ │12│  │1 30│    │
│ │ 6│  │1 40│    │
│ └~─┘  └~───┘    │
└∊────────────────┘
```

```
      DISPLAY ⍉⍉¨Z
┌→2 1──────────────────┐
↓ ┌→1 4──────────┐     │
│ ↓7  10  12  6  │     │
│ └~─────────────┘     │
│ ┌→2 4────────────────┐│
│ ↓  1   1   1   1     ││
│ │10  20  30  40      ││
│ └~───────────────────┘│
└∊─────────────────────┘
```

```
      DISPLAY (⍉⍉¨Z) TIMES Z
┌→2 2──────────────────────┐
↓ ┌→1 ┐  ┌→1 2────┐        │
│ ↓329│  ↓35  870 │        │
│ └~──┘  └~───────┘        │
│ ┌→2 ┐  ┌→2 2──────┐      │
│ ↓ 35│  ↓  4   100 │      │
│ │870│  │100  3000 │      │
│ └~──┘  └~─────────┘      │
└∊─────────────────────────┘
```

APL2STAT: Datasets as nested arrays

```
DISPLAY DATASET_PROTO
```



APL2STAT objects:

- N x 2 nested arrays of (property, value)
- Value can be any *APL2STAT* object
- Inheritance of methods through TYPE, PARENT properties

```
      SAMPLE
PARENT         DATASET_PROTO
TYPE           DATASET
VARIABLES       EDUCATION INCOME GENDER
OBSERVATIONS    BEN VALERIE ANITA MARIA JOHN JIM ABIGAIL JUAN JUDY
DATA            12  25  M
                16   .  F
                 9  18  F
                12  21  F
                20  55  M
                12  30  M
                14  45  F
                 .  51  M
                12  18  F
CODEBOOK        Sample data set. The variables are:
                 [1] Education (years)
                 [2] Income ($000)
                 [3] Gender ('M'/'F')
```

APL2, APL2STAT Features

- implicit iteration with data and each
  - APL has no built-in control structures (do-while, foreach, loops)
  - instead, powerful data-driven computations replace iteration
  - e.g., calculate cumulative proportions by rows or columns

```
      ⎕←M←3 4⍴⍳12
1  2  3  4
5  6  7  8
9 10 11 12

      +/[2]M                  ⍝sum over columns
10 26 42

      M÷[1]+/[2]M             ⍝proportions of row totals
0.1    0.2    0.3    0.4
0.1923 0.2308 0.2692 0.3077
0.2143 0.2381 0.2619 0.2857

      +\M÷[1]+/[2]M           ⍝cumulative proportions
0.1    0.3    0.6    1
0.1923 0.4231 0.6923 1
0.2143 0.4524 0.7143 1
```

APL2, APL2STAT Features

- operators: functions of functions ("closures")
  - familiar example: derivatives

$$f(x) = 3x^2 - 2x + 3$$

$$f'(x) \equiv \frac{d}{dx} f(x) = 6x - 2$$

$$f''(x) \equiv \frac{d}{dx} f'(x) = 6$$

  - APL primitive operators : f/ (reduction), f\ (scan) °.f (outer product)

```
      ⍝-- GENERALIZED OUTER PRODUCTS
      1 2 3∘.+10 20          ⍝ sum of each with each
11 21
12 22
13 23
      1 2 3∘.*1 2 3          ⍝ ⍳3 raised to each power
1 1  1
2 4  8
3 9 27
      1 2 3∘.≥1 2 3          ⍝ lower triangular matrix
1 0 0
1 1 0
1 1 1
```

- APL2STAT operators: e.g., BOOTSTRAP

replications

function of data

data

Internal documentation

HOW ‘BOOTSTRAP’

```
      ∇
[0]   r←reps(fn BOOTSTRAP)data;n;bootstrap
[1]   ⍝ Operator for computing bootstrapped sample estimates.
[2]   ⍝.L (reps) Number of bootstrap replications. Default: 100
[3]   ⍝.L <fn> A function that takes a data matrix as a right
[4]   ⍝.L      argument and returns a vector or estimates.
[5]   ⍝.R <data> Input data matrix or vector.  Rows of the data
[6]   ⍝.R          matrix are sampled repeatedly with replacement.
[7]   ⍝.Z <r> A matrix containing (by rows) the result of fn for
[8]   ⍝.Z      each bootstrap replication.
[9]    r←⎕FX 'r←(fn bootstrap)r' 'r←,fn data[?n⍴n;]'
[10]   uses¨'default' 'select' 'get_data' 'null' 'COL'
[11]   'reps' default 100
[12]   n←1↑⍴data←COL select get_data data
[13]   ('Result for full sample: ')(fn data)
[14]   'Beginning BOOTSTRAP replications '
[15]   r←fn bootstrap¨⍳reps
[16]   'BOOTSTRAP replications completed.'
[17]   r←⊃[2]r
     ∇
```

return result

apply to each replication

local operator: 1 bootstrap sample

- APL2STAT operators: e.g., BOOTSTRAP

```
      ⍝ Define a function to return mean and variance
     ∇
[0]   R←STATS X
[1]   R←(MEAN X),VARIANCE X
     ∇

      DATA←* NORMAL_RAND 50          ⍝ 50 lognormal values

      R←STATS BOOTSTRAP DATA         ⍝ bootstrap means and variances
 Result for full sample:    1.5357 3.5881
Beginning BOOTSTRAP replications
BOOTSTRAP replications completed.

      ('MEAN' 'VAR'),[1]5↑[1]R       ⍝ first 5 bootstrap replications
 MEAN   VAR
 1.5824 3.8957
 1.4367 2.6835
 2.1826 7.8176
 1.9824 5.5228
  1.174 1.1709

      DESCRIBE R                     ⍝ summarize
                          Col_1   Col_2
 Mean                     1.5651 3.7373
 Standard deviation      0.25711 1.3982
 Minimum                   1.126 1.1541
 Lower hinge (Q1)          1.381  2.651
 Median                   1.5329  3.652
 Upper hinge (Q3)         1.7305 4.6877
 Maximum                  2.3215 7.8176
 N (selected, ≠'.')          100    100
```

Statistical computing: common tasks

- Data summaries by individual or group(s)
  - Find mean, sd, Q25, Q75 for each group
  - Group-wise transformations (scale, standardize)
  - Fit same model to each S or group
    - Longitudinal data: individual trajectories
    - Multilevel models: level 1 models
  - What's the pattern?
    - How does your software (SAS or R) help you think about solutions?

More common tasks

- Simulation studies
  - Effect of violation of constant variance on {*p*-value/power} in one-way ANOVA
  - Determining power for the new Cribbie-Mara multiple comparison procedure
- Experiment:
  - Generate multiple datasets with varying parameters
  - Analyze each: empirical *p*-value or power
  - Summarize collection
- What's the pattern?
  - How does your software (SAS or R) help you think about solutions?

Some *less* common tasks

- Computer-intensive methods
  - Multiple-imputation for missing data
    - Generate *m* imputed complete data sets
    - Analyze each using standard methods
    - Combine to give tests taking missing into account
  - Bootstrapping, when parametric methods fail
    - Generate *B* bootstrap samples from the data
    - Obtain standard estimates for each
    - Combine to give bootstrap estimate & CI
  - What's the pattern?
    - How does your software (SAS or R) help you think about solutions?

Some *less* common tasks

- Implement a new statistical procedure
  - Cribbie-Mara procedure; Flora's Λ
  - Make them publicly available
- Implement a new graphical method
  - Fox: effect plots
  - mosaic displays
  - HE plots
- Write a paper using reproducible research methods
  - All data, results, graphs verifiable & public
- How does your software (SAS or R) help?

SAS thinking

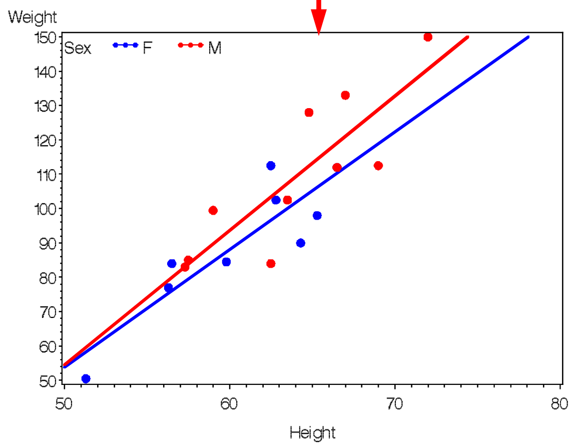
- PROC steps, DATA steps, ODS & more

```
data class;
 input age sex ht wt;
datalines;
 20 M 75 152
 22 F 69 132
```

```
proc glm data=class;
  class sex;
  model wt = ht sex;
  output out=results
    p=predict r=resid;
```

results

```
proc gplot data=results;
  plot wt * ht = sex;
  symbol1 ...
  symbol2 ...
```

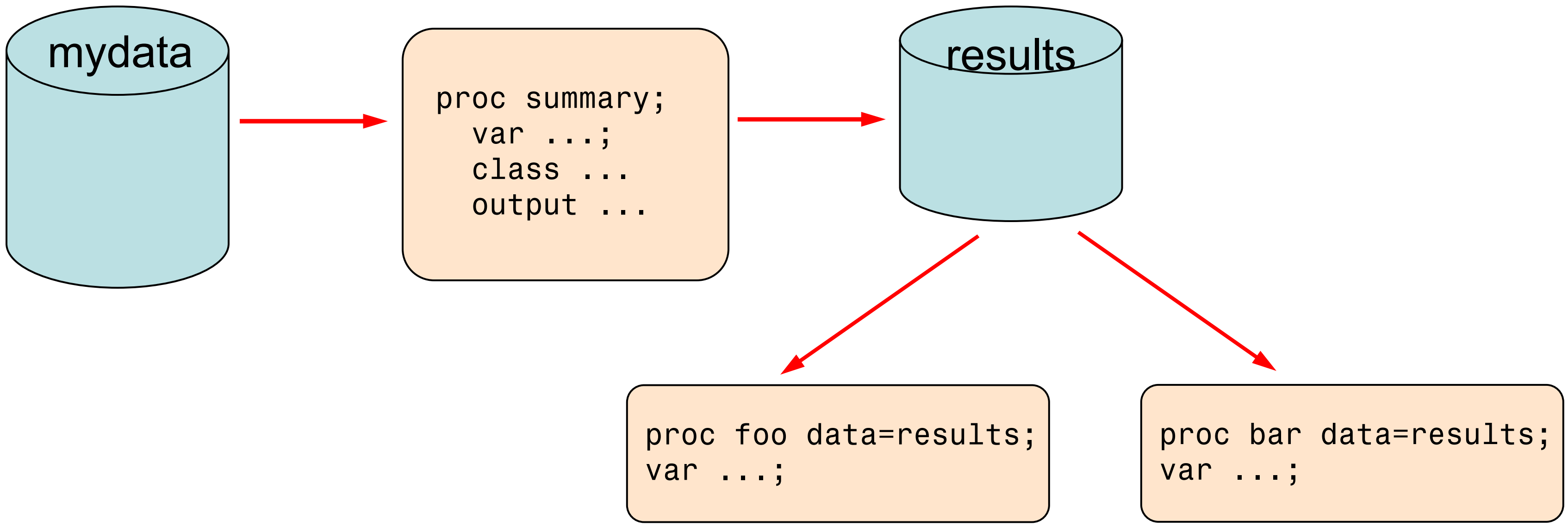


But, wait: there's more:

- ODS: capture *any* results to a data set
- ODS graphics: automatic plots from procs
- SAS/IML: matrix computations
- Macro language: custom graphics
- proc SQL, …

SAS thinking

- PROC SUMMARY
  - General procedure for univariate summaries
  - Output dataset for further processing



Cars data:

```
Obs make     model      mpg cylinder engine horse weight accel year origin

  1 chev     chevelle    18      8       307    130    3504   12.0  70     A
  2 buick    skylark     15      8       350    165    3693   11.5  70     A
  3 plymouth satellite   18      8       318    150    3436   11.0  70     A
  4 amc      rebel       16      8       304    150    3433   12.0  70     A
  5 ford     torino      17      8       302    140    3449   10.5  70     A
  6 ford     galaxie     15      8       429    198    4341   10.0  70     A
  7 chev     impala      14      8       454    220    4354    9.0  70     A
  8 plymouth fury        14      8       440    215    4312    8.5  70     A
    ...
```

Univariate, multiple summary statistics

```
proc summary data=cars;
   var mpg;
   class origin;
   output out=results mean=mean stddev=stddev qrange=qrange q1=q1 q3=q3;
   run;
```

```
_TYPE_    origin    _FREQ_      mean      stddev     qrange      q1       q3

   0                  406     23.5146    7.81598      11.5      17.5     29.0

   1        A         254     20.0835    6.40289       9.0      15.0     24.0
            E          73     27.8914    6.72393       6.7      24.0     30.7
            J          79     30.4506    6.09005       8.7      25.4     34.1
```

Multivariate: one summary measure

```
proc summary data=cars;
  var mpg accel weight;
  class origin;
  output out=means mean=;
  run;
```

TYPE	origin	_FREQ_	mpg	accel	weight
0		406	23.5146	15.5197	2979.41
1	A	254	20.0835	14.9425	3372.70
	E	73	27.8914	16.8219	2431.49
	J	79	30.4506	16.1722	2221.23

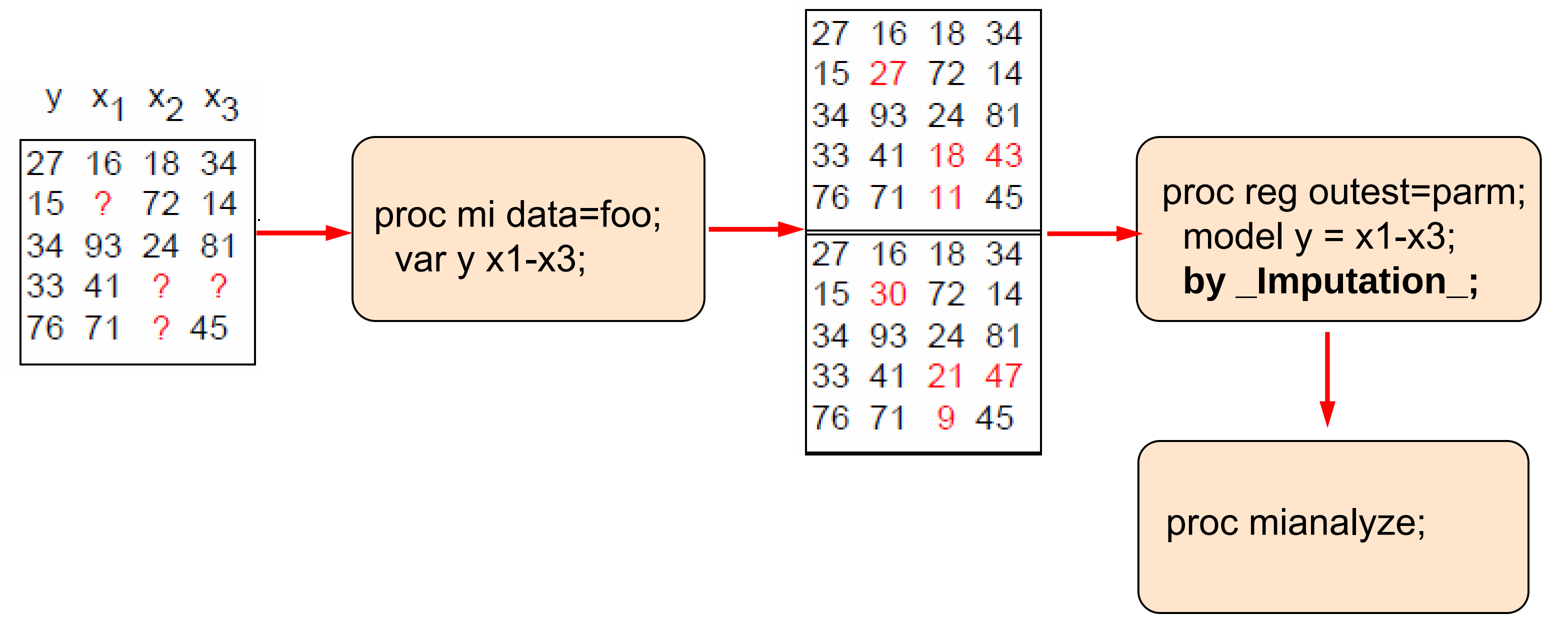
Univariate: two+ class variables;

```
proc summary data=cars;
  var mpg;
  class origin cylinder;
  output out=means mean=;
  run;
```

TYPE	origin	cylinder	_FREQ_	mpg
0		.	406	23.5146
1		3	4	20.5500
		4	207	29.2868
		5	3	27.3667
		6	84	19.9857
		8	108	14.9631
2	A	.	254	20.0835
	E	.	73	27.8914
	J	.	79	30.4506
3	A	4	72	27.8403
	A	6	74	19.6635
	A	8	108	14.9631
	E	4	66	28.4111
	E	5	3	27.3667
	E	6	4	20.1000
	J	3	4	20.5500
	J	4	69	31.5957
	J	6	6	23.8833

SAS thinking: BY processing

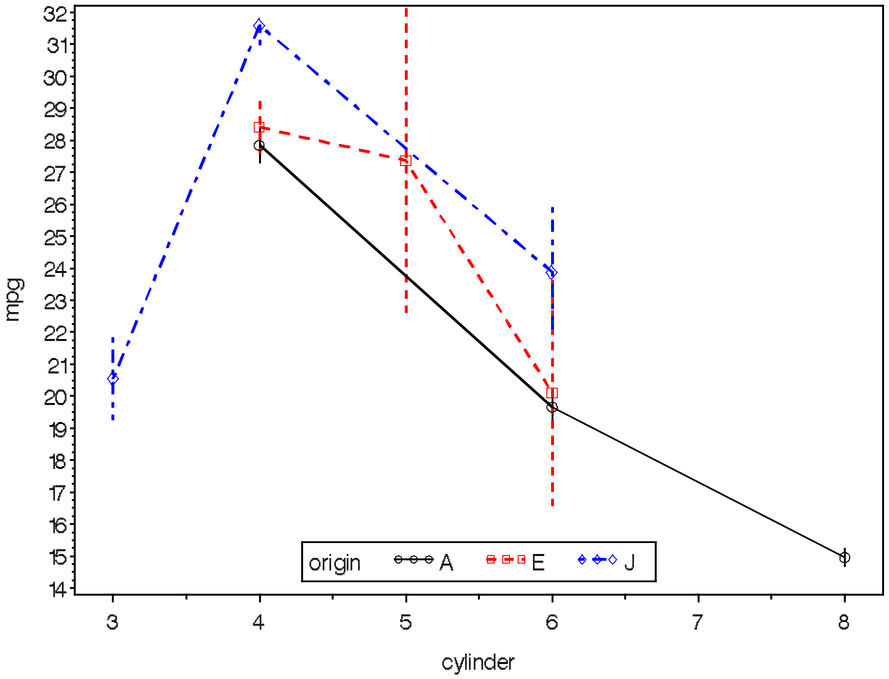
- BY processing
  - All (almost) SAS procedures accept BY stmt
  - Do the procedure for all values of BY variables



SAS thinking: Macros

- Macro language
  - Combine any number of PROC and DATA steps into a general procedure

%***meanplot***(data=cars, var=mpg, class=cylinder origin);



SAS thinking: Macros

```
%macro meanplot (data=_last_,
var=, class=, out=, z=1, ...);
```

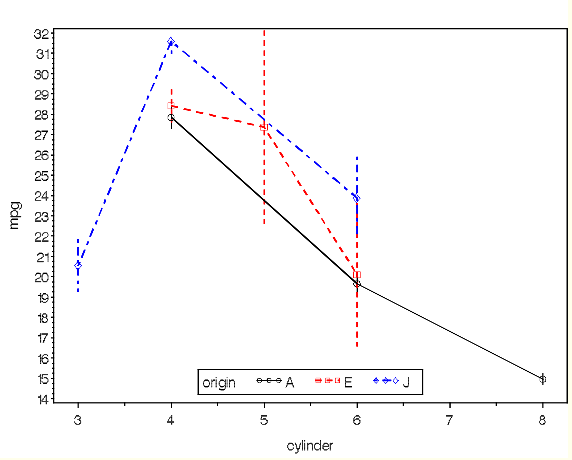
```
proc summary data=&data;
  var &var;
  class &class;
  output out=&out mean= stderr=se;
```

```
proc glm data=&data outstat=_stat_;
  class &class;
  model &var = &class;
  /* extract MSE and dfe */
```

```
/* annotate data set to draw err bars
*/
data _bars_; set &out;
  x=&xvar; y=mean+se; function='move';
  x=&xvar; y=mean-se; function='draw';
  ...
```

```
proc gplot data=&out;
  by &panels;
  plot &var * &xvar = &sym /
    anno=_bars_ &haxis &vaxis ...;
  ...
```

```
/* clean up */
proc datasets;
  delete _work_ _bars_ ...;
```



SAS thinking: Macros

- What’s wrong with macros?
  - Another language to learn
  - Text substitution, not computation
  - Difficult to use other macros as building blocks
    - No require() feature
    - No version control
  - No standards for documentation, examples, dissemination

SAS thinking: many languages

SAS/Graph:

• procs, Annotate language

Base SAS, SAS/STAT

• data step, proc steps

SAS thinking: many languages

%macro language

proc iml
• matrix language, graphics

SAS/Graph:
• procs, Annotate language

Base SAS, SAS/STAT
• data step, proc steps

SAS thinking : many languages

ODS graphics
• template language

Output delivery system (ODS)

%macro language

proc iml
• matrix language, graphics

SAS/Graph:
• procs, Annotate language

Base SAS, SAS/STAT
• data step, proc steps

R thinking: Features

- What features contribute to the power of R for statistical computing & graphics?
- Language & data
  - Data objects: arrays, lists, data frames, ...
  - Object methods (S3, S4)
  - Formulas: compact notation for models and graphs
  - grammars for graphics: base, lattice, ggplot2
  - all of the above are extensible!
- R environment
  - documentation: .Rd format, executable examples, vignettes
  - packages: now over 2000 contributed packages
  - CRAN: easy upgrade, task views, ...
  - Social: newsgroups (R-help), blogs, galleries, ...

thinking: Objects

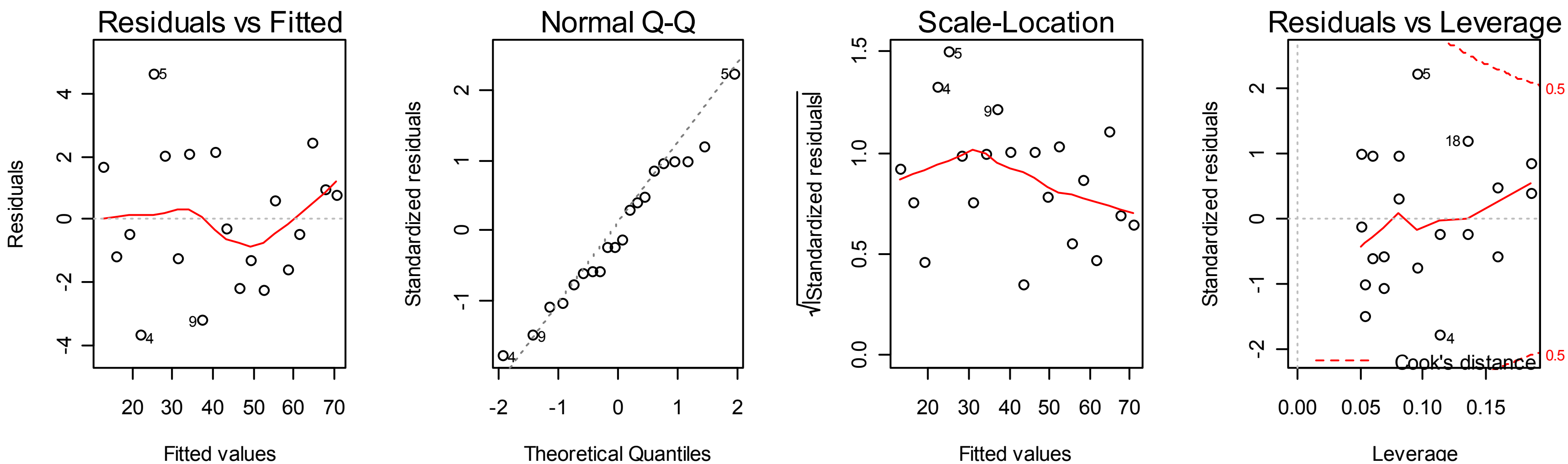
- Data objects: data.frame, matrix, array, list, …
- *Everything* in R is an object!
- Objects have *methods*

```
> x <- 1:20
> y <- 10 + 3*x + 2*rnorm(20)
> mymod <- lm(y ~ x)

> class(mymod)
[1] "lm"
> methods(class = "lm")
 [1] add1.lm*           alias.lm*           anova.lm            case.names.lm*
 [5] confint.lm*        cooks.distance.lm*  deviance.lm*        dfbeta.lm*
 [9] dfbetas.lm*        drop1.lm*           dummy.coef.lm*      effects.lm*
[13] extractAIC.lm*     family.lm*          formula.lm*         hatvalues.lm
[17] influence.lm*      kappa.lm            labels.lm*          logLik.lm*
[21] model.frame.lm     model.matrix.lm     plot.lm             predict.lm
[25] print.lm           proj.lm*            residuals.lm        rstandard.lm
[29] rstudent.lm        simulate.lm*        summary.lm          variable.names.lm*
[33] vcov.lm*

   Non-visible functions are asterisked
```

thinking: Methods

```
> coefficients(mymod)
(Intercept)           x
      9.831       3.039
> residuals(mymod)
      1       2       3       4       5       6       7       8       9      10
 1.6690 -1.1390 -0.4393 -3.6188  4.6582  2.0145 -1.1890  2.0841 -3.1287  2.1334
     11      12      13      14      15      16      17      18      19      20
-0.2631 -2.1389 -1.2886 -2.2416  0.6391 -1.5430 -0.4538  2.4677  0.9650  0.8128

> op <- par(mfrow=c(1,4))
> plot(mymod)
> par(op)
```

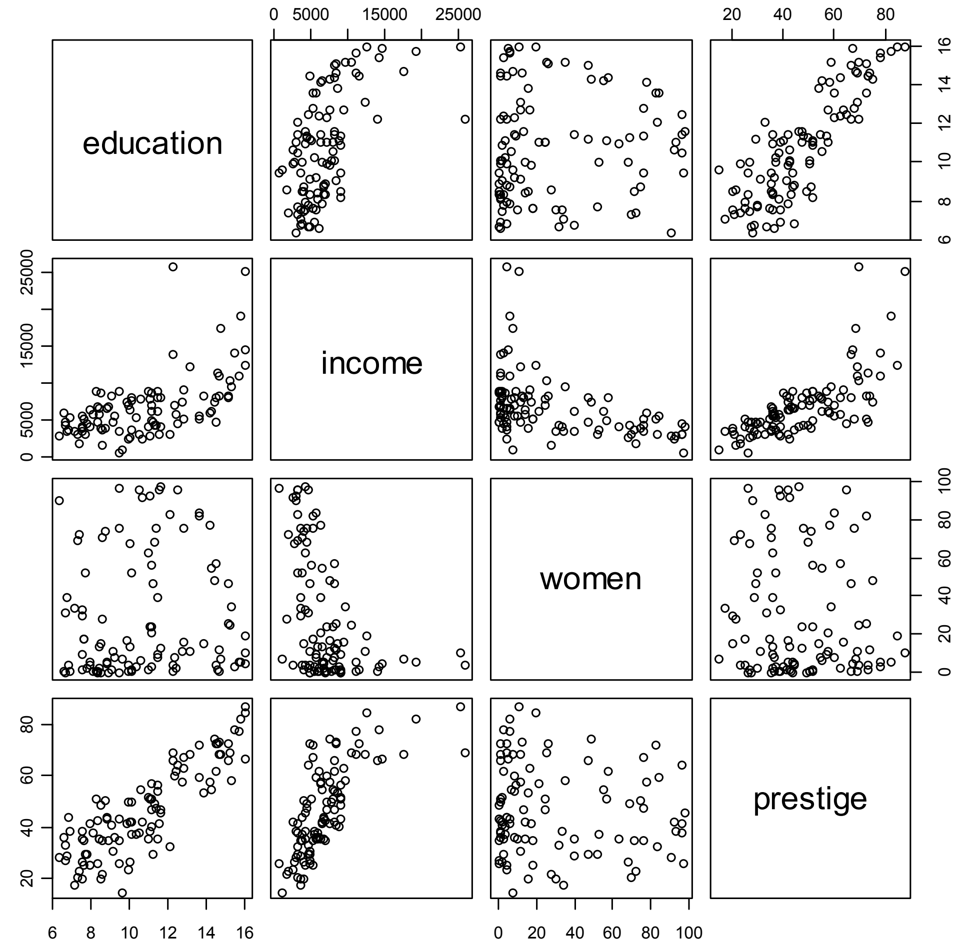
plot(lm) objects: ‘regression quartet”



thinking: Methods

plot(data.frame) objects: scatterplot matrix

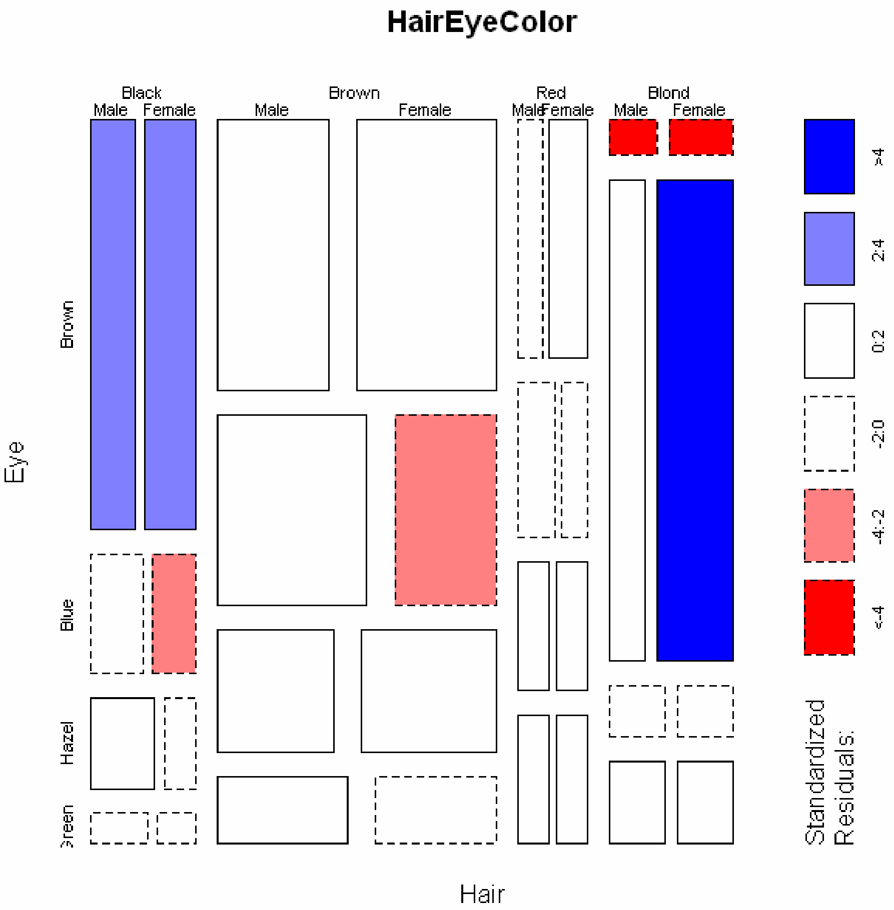
```
> library(car)
> class(Prestige)
[1] "data.frame"
> plot(Prestige[,1:4])
```



thinking: Methods

plot(table) objects: mosaic display

```
> library(vcd)
> class(HairEyeColor)
[1] "table"
> str(HairEyeColor)
 table [1:4, 1:4, 1:2] 32 53 10 3 11 50 10 ...
 - attr(*, "dimnames")=List of 3
  ..$ Hair: chr [1:4] "Black" "Brown" "Red"
"Blond"
  ..$ Eye : chr [1:4] "Brown" "Blue" "Hazel"
"Green"
  ..$ Sex : chr [1:2] "Male" "Female"
> plot(HairEyeColor, shade=TRUE)
```



thinking: Formulas

- Model formulas: response ~ predictor(s)
  - + adds new terms: y `~ x1 + x2 + x3`
  - - omits terms: y `~ -1 + x`
  - `:` interactions between terms: y `~ x1 + x2 + x1:x2`
  - `*` expands to interactions + terms (y `~ a*b` → `y ~ a+b+a:b`)
  - `^n` all terms and interactions up to order n:
    - y ~ (a+b+c)^2 → y ~ a + b + c + a:b + a:c + b:c
    - y ~ (a+b+c)^3 → y ~ a + b + c + a:b + a:c + b:c + a:b:c
  - functions: `log(x)`, `I(x^2)`, `poly(x, 4)`, ...
  - multivariate responses: `cbind(y1, y2, y3) ~ x1 + x2 + x3`
  - short-hands:
    - Use everything else: `y ~ .`
    - Update methods: `update(model1, . ~ . + x5)`

thinking: Formulas

- **generality**: applies to *all* model functions (with extensions)
  - Linear models: lm()
  - Generalized linear models: glm()
    - `glm(Freq ~ (row+col+layer)^2, family=poisson)`
  - Nonlinear models: nls()
    - `nls(y ~ Asym/(1 + exp((Xmid - log(conc))/Scale)))`
  - Generalized non-linear models: gnm()
    - `gnm(Freq ~ row+col + Diag(row,col) + Mult(row,col))`
  - Robust linear models: MASS::rlm()
  - Mixed models: nmle

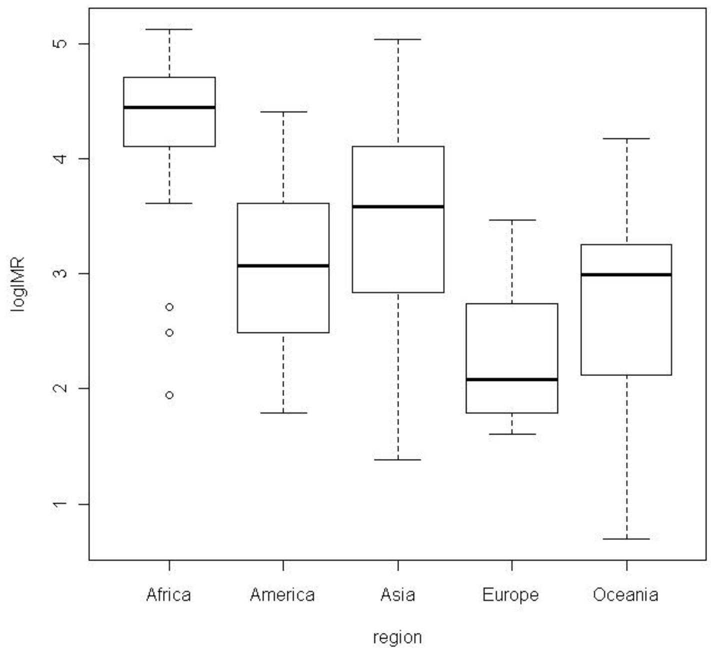
thinking: Formulas

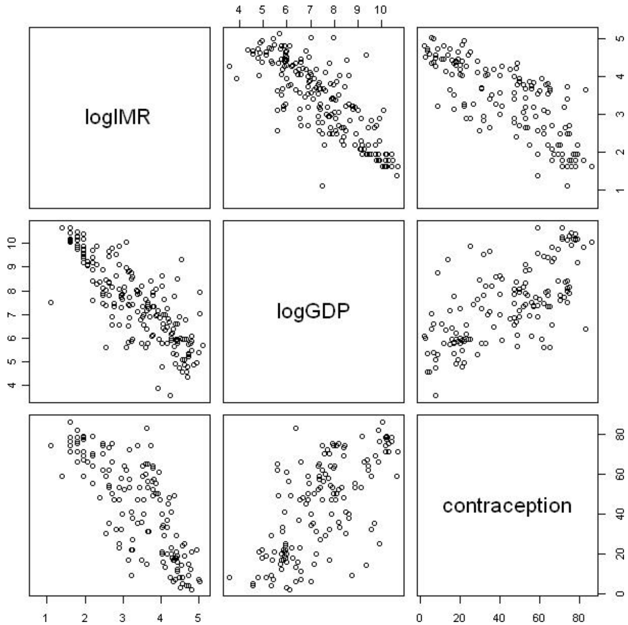
- **suggestivity**: a notation for other things
  - Crosstabs: xtabs(), vcd::structable(), ...

```
> DF <- as.data.frame(UCBAdmissions) # make a data frame
> ## Nice for taking margins ...
> xtabs(Freq ~ Gender + Admit, DF)
        Admit
Gender   Admitted Rejected
  Male       1198     1493
  Female      557     1278
> ## And for testing independence ...
> summary(xtabs(Freq ~ ., DF))
Call: xtabs(formula = Freq ~ ., data = DF)
Number of cases in table: 4526
Number of factors: 3
Test for independence of all factors:
        Chisq = 2000.3, df = 16, p-value = 0
```

- suggestivity: ...
  - Formulas for graphs

```
plot(logIMR ~ logGDP, data=UN)                                    # scatterplot
plot(logIMR ~ region, data=UN)                                    # boxplots
plot(logIMR ~ logGDP + contraception + educationFemale, data=UN) # 3 scatterplots
plot(~ logIMR + logGDP + contraception, data=UN)                  # scatterplot matrix
```

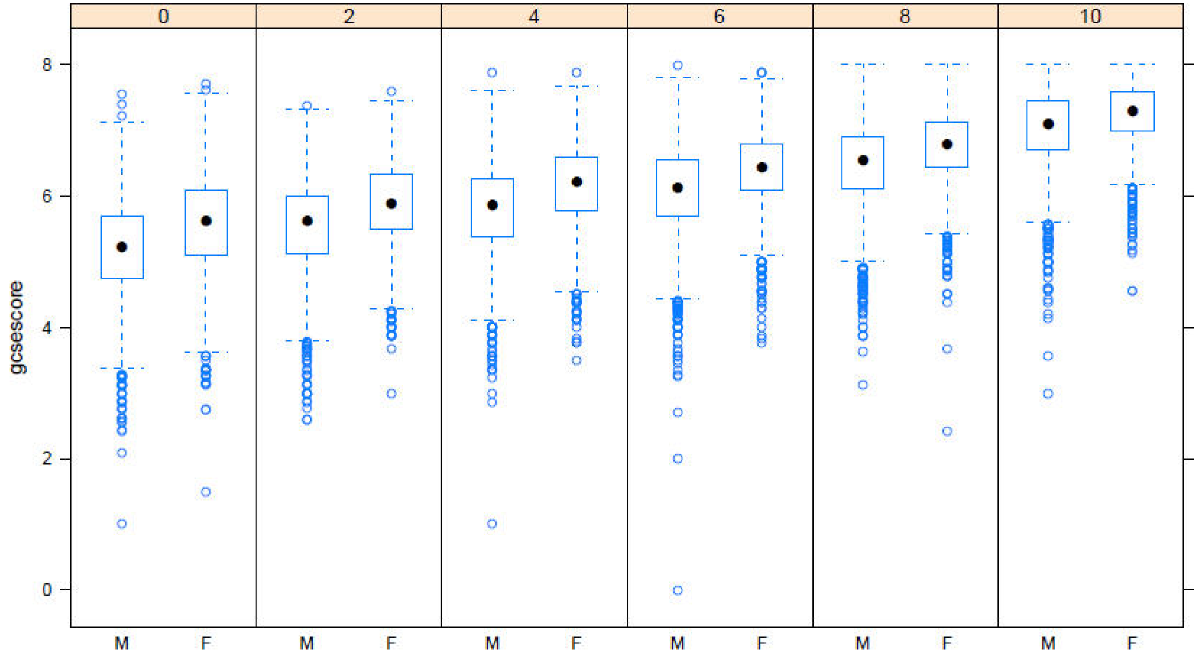




Graph formulas: Lattice extensions

- Conditioned plots: y ~ x | z

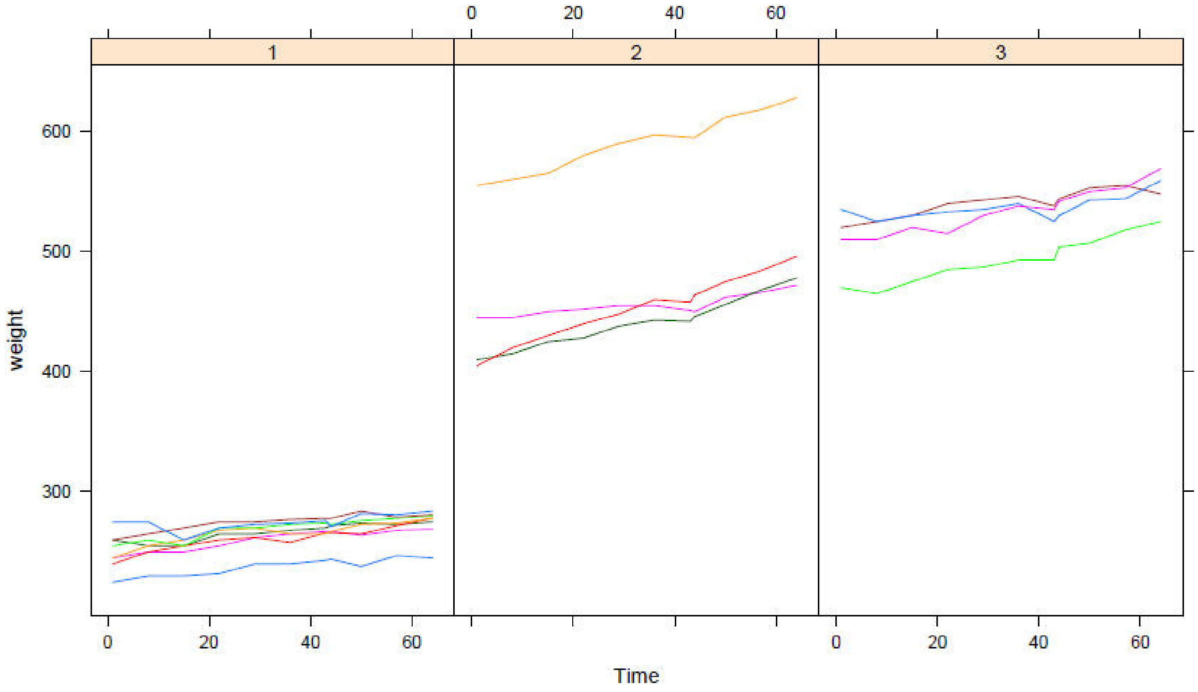
```
library(lattice)
data(Chem97, package = "mlmRev")
bwplot(gcsescore ~ gender | factor(score), Chem97, layout = c(6, 1))
```



Graph formulas: Lattice extensions

- Conditional+grouping: y ~ x | z, groups=

```
data(BodyWeight, package = "nlme")
xyplot(weight ~ Time | Diet, BodyWeight, groups = Rat,
   type = "l", layout = c(3, 1))
```



thinking: *apply

- Functional programming: apply a function to subsets of an object

matrix or array

function applied

```
apply(X, MARGIN, FUN, ...)
```

subscript(s) applied over

other args to FUN()

array of dim(X)[MARGIN]

- Other variations:
  - lapply(X, FUN, …) – apply FUN(X[[i]], …) to lists
  - tapply(X, INDEX, FUN, ...) – tables indexed by factors
  - convience functions: sweep(), by(), aggregate(), replicate(), …

apply() examples

```
> # 100 random chisq(4) values in a 25x4 matrix
> dat <- matrix(rchisq(100, 4), ncol = 4)
> head(dat,3)
       [,1]   [,2]    [,3]   [,4]
[1,] 5.3343 3.2368  3.0089 7.7839
[2,] 3.8361 3.5756  6.2050 6.0195
[3,] 2.8049 1.6793  8.9917 5.2745
```

Create some data

```
> apply(dat, 2, mean)
[1] 4.1591 3.3582 4.7192 3.7962
> # trimmed means
> apply(dat, 2, mean, trim=0.05)
[1] 4.0448 3.2576 4.5819 3.7392
> # variances
> apply(dat, 2, function(x) sd(x)^2)
[1]  5.1259  3.7888 10.2617  5.4181
```

apply some functions

```
> skewness <-function (x) {
+     n <- length(x)
+     x <- x - mean(x)
+     skew <- sqrt(n) * sum(x^3)/(sum(x^2)^(3/2))  # std skewness
+     skew <- skew * sqrt(n * (n - 1))/(n - 2)     # SAS, SPSS form
+     skew}
>
> apply(dat, 2, skewness)
[1] 0.87816 0.85000 0.72138 0.41406
```

Custom functions

replicate() examples

simulate distribution of eigenvalues of a correlation matrix (Horn's method)

Do it once, by hand

```
> set.seed(1)
> N <- 100
> dat <- mvrnorm(N,rep(0,3),diag(3))
> cor(dat)
            [,1]        [,2]      [,3]
[1,]  1.00000000 -0.00099432  0.018382
[2,] -0.00099432  1.00000000 -0.049536
[3,]  0.01838219 -0.04953621  1.000000
> eigen(cor(dat))$values
[1] 1.05317 0.99935 0.94748
```

Wrap in a function

```
> eigensim <- function(N, mu, sigma) {
+   dat <- mvrnorm(N,mu,sigma)
+   eigen(cor(dat))$values
+ }
```

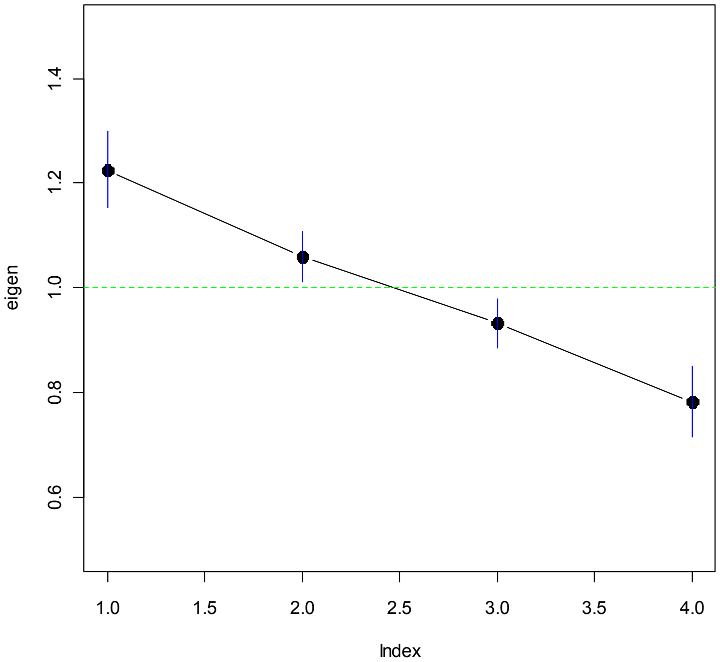
Replicate 5 times

```
> mu <- rep(0,4)
> sigma <- diag(4)
> replicate(5, eigensim(N, mu, sigma))
        [,1]    [,2]    [,3]    [,4]    [,5]
[1,] 1.16863 1.22323 1.26190 1.22898 1.23730
[2,] 1.10411 1.11780 1.03367 1.06498 1.12878
[3,] 0.88743 0.91680 0.92439 0.96007 0.91078
[4,] 0.83983 0.74217 0.78005 0.74597 0.72314
```

replicate() examples

Do 1000 replications, summarize and plot

```
> reps <- replicate(1000, eigensim(N, mu, sigma))
> (eigen <- apply(reps, 1, mean))
[1] 1.22589 1.05877 0.93298 0.78236
> (sdbars <- apply(reps, 1, sd))
[1] 0.073541 0.048007 0.046980 0.067464

> plot(eigen, type='b', pch=16, cex=1.5, ylim=c(0.5,1.5))
> abline(h=1, lty=2, col="green")
> segments(1:4, eigen+sdbars, 1:4, eigen-sdbars, col="blue")
```



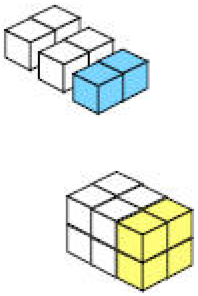
thinking: plyr

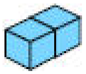
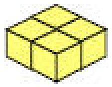
- General package for split-apply-combine

Split a data object into pieces

Apply a function to each piece

Combine the pieces back together

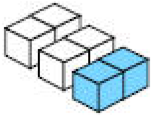


fun() -> 

Input		Output		
a (array)	+	a (array)	+	ply
d (data frame)		d (data frame)		
l (list)		l (list)		
		_ (nothing)		
What to split		How to combine		

thinking: plyr

Basic plyr functions



name	age	sex
John	13	Male
Mary	15	Female

input \ output	array	data frame	list	discarded
array	aaply	adply	alply	a_ply
data frame	daply	ddply	dlply	d_ply
list	laply	ldply	llply	l_ply

Arguments

```
a*ply(.data, .margins, .fun, …)
d*ply(.data, .variables, .fun, …)
l*ply(.data, .fun, …)
```

thinking: plyr

- Advantages over std *apply functions
  - consistent names, arguments and outputs
  - convenient parallelization through the `foreach` package (large simulations: multi processors)
  - input from and output to data.frames, matrices and lists
  - progress bars to keep track of long running operations
  - built-in error recovery, and informative error messages (failwith= argument)
  - labels that are maintained across all transformations

plyr: ddply()

```
new <-ddply(.data, .variables, .fun, ...)
```

- Arguments:
  - `.data`: data frame to process
  - `.variables`: combinations of variables to split by
  - `.fun`: function to call on each piece
  - `...` : extra args passed to `.fun()`
- Variable syntax:
  - Character: `c("sex", "year")`
  - Numeric: `1:3`
  - Formula: `~ sex + year`
  - Special
    - `.(sex, year)`
    - `.(first = substr(name, 1, 1))`

plyr: ddply()

Ways to split a data frame

DF

name	age	sex
John	13	Male
Mary	15	Female
Alice	14	Female
Peter	13	Male
Roger	14	Male
Phyllis	13	Female

`d*ply(DF, .(sex), ...)`

.(sex)

name	age	sex
John	13	*Male*
Peter	13	*Male*
Roger	14	*Male*

name	age	sex
Mary	15	*Female*
Alice	14	*Female*
Phyllis	13	*Female*

`d*ply(DF, .(age), ...)`

.(age)

name	age	sex
John	*13*	Male
Peter	*13*	Male
Phyllis	*13*	Female

name	age	sex
Alice	*14*	Female
Roger	*14*	Male

name	age	sex
Mary	*15*	Female

```
> ddply(DF, .(sex), "nrow")
     sex nrow
1 Female    3
2   Male    3
> ddply(DF, .(sex, age), "nrow")
     sex age nrow
1 Female  13    1
2 Female  14    1
3 Female  15    1
4   Male  13    2
5   Male  14    1
```

```
> ddply(DF, .(sex), summarize, mean.age=mean(age))
     sex mean.age
1 Female 14.00000
2   Male 13.33333
> ddply(DF, .(sex), summarize, mean.age=mean(age),
sd.age=sd(age))
     sex mean.age    sd.age
1 Female 14.00000 1.0000000
2   Male 13.33333 0.5773503
```

plyr: a*ply()

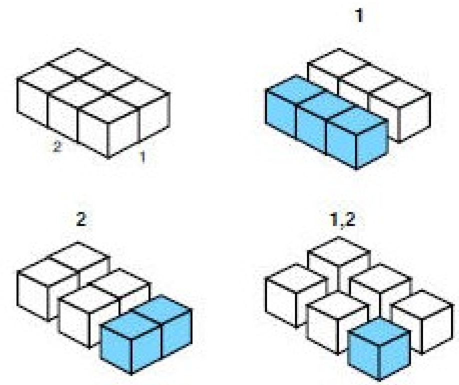
Ways to split a 2-way array

.margin=c()

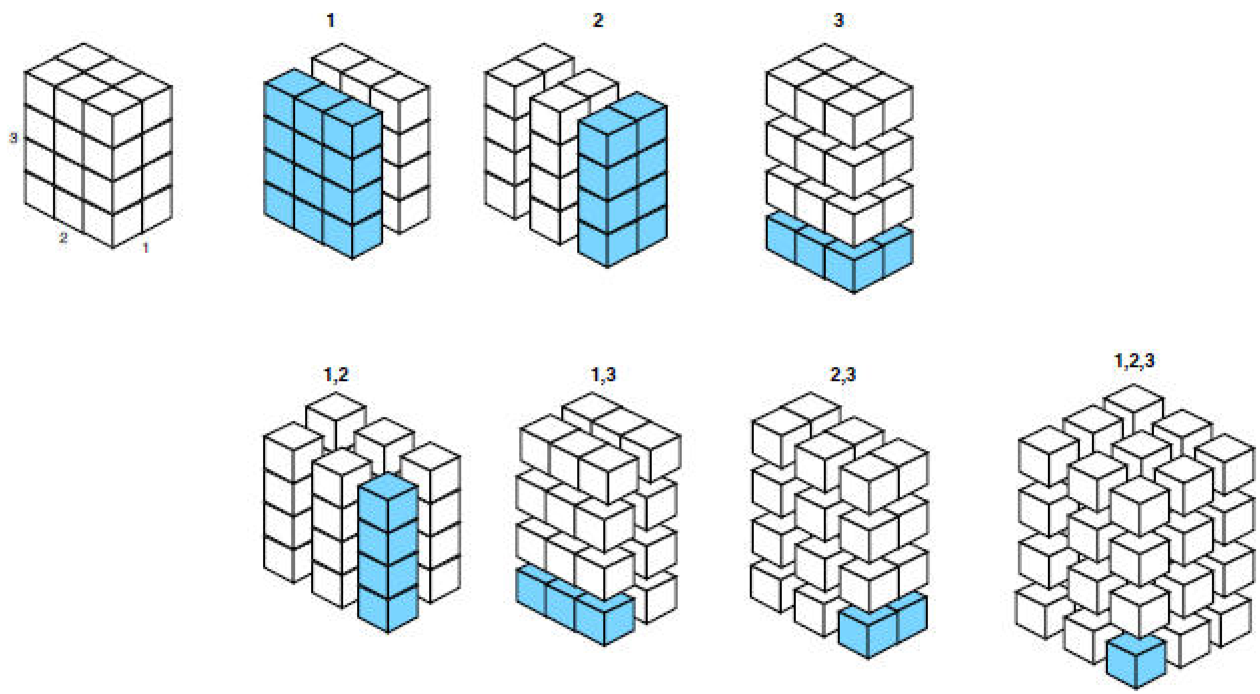
.margin=1

.margin=2

.margin=1:2



Ways to split a 3-way array



plyr: aaply() examples

```
> dim(HairEyeColor)
[1] 4 4 2
> str(HairEyeColor)
 table [1:4, 1:4, 1:2] 32 53 10 3 11 50 10 30 10 25 ...
 - attr(*, "dimnames")=List of 3
  ..$ Hair: chr [1:4] "Black" "Brown" "Red" "Blond"
  ..$ Eye : chr [1:4] "Brown" "Blue" "Hazel" "Green"
  ..$ Sex : chr [1:2] "Male" "Female"
```

```
> # one-way marginal frequencies
> aaply(HairEyeColor, 1, sum)
Black Blond Brown   Red
  108   127   286    71
> aaply(HairEyeColor, 2, sum)
 Blue Brown Green Hazel
  215   220    64    93
> aaply(HairEyeColor, 3, sum)
Female   Male
   313    279
> # collapse over Sex
> (HE <- aaply(HairEyeColor, 1:2, sum))
       Eye
Hair     Blue Brown Green Hazel
  Black    20    68     5    15
  Blond    94     7    16    10
  Brown    84   119    29    54
  Red      17    26    14    14
```

```
> percents <- function(x) x/sum(x)
> aaply(HE, 1, percents)

Hair          Blue      Brown     Green      Hazel
  Black 0.1851852 0.62962963 0.0462963 0.13888889
  Blond 0.7401575 0.05511811 0.1259843 0.07874016
  Brown 0.2937063 0.41608392 0.1013986 0.18881119
  Red   0.2394366 0.36619718 0.1971831 0.19718310
> rowSums(aaply(HE, 1, percents))
Black Blond Brown   Red
    1     1     1     1
> aaply(HE, 2, percents)

Eye           Black      Blond     Brown        Red
  Blue  0.09302326 0.43720930 0.3906977 0.07906977
  Brown 0.30909091 0.03181818 0.5409091 0.11818182
  Green 0.07812500 0.25000000 0.4531250 0.21875000
  Hazel 0.16129032 0.10752688 0.5806452 0.15053763
> rowSums(aaply(HE, 2, percents))
 Blue Brown Green Hazel
    1     1     1     1
```

plyr: strategies for data analysis & graphics

1. Extract a subset of the data for which it is easy to solve the problem
2. Solve the problem by hand, checking as you go
3. Write a function that encapsulates the solution
4. Use appropriate **ply function to
   - split the data into pieces,
   - apply the function to each piece
   - join the pieces back together

Example: baseball, 1871-2007

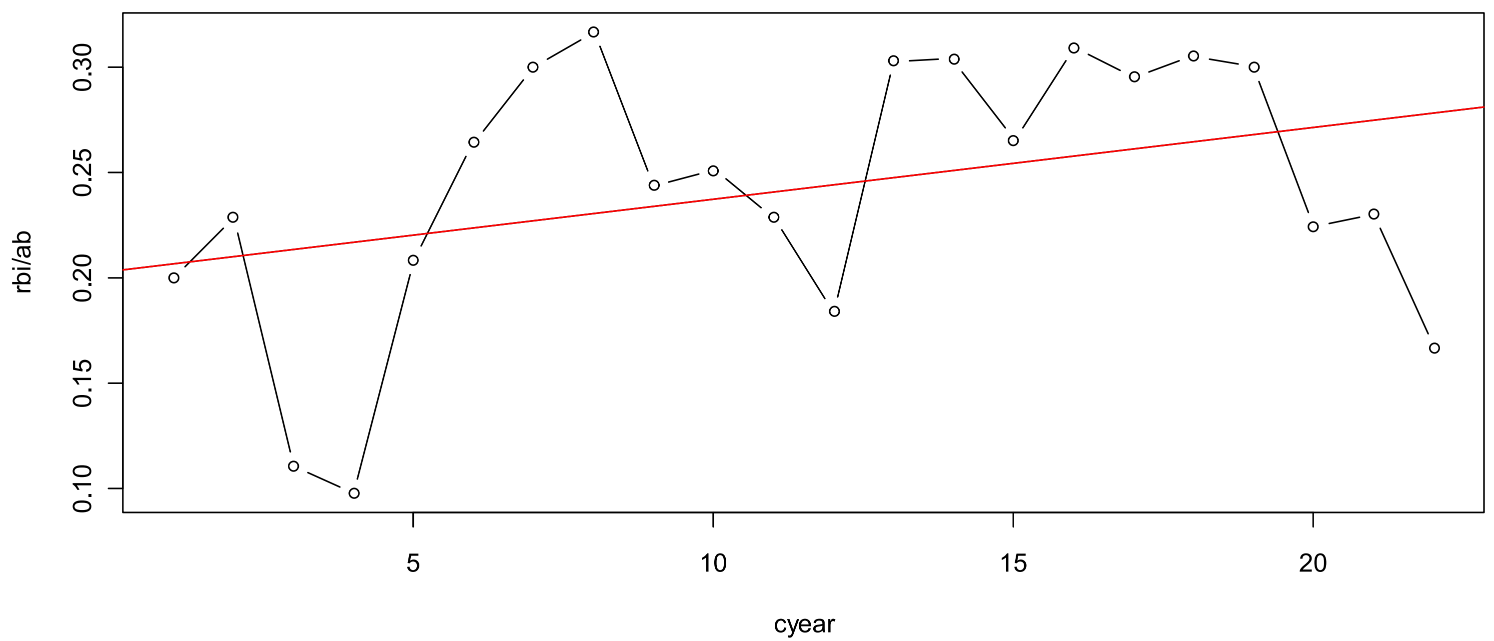
Batting records of all US players, 1871-2007 with 15+ years of data

Focus on year, rbi (runs batted in), ab (at bats): performance over career

```
> data(baseball)
> dim(baseball)
[1] 21699    22
> head(baseball)[,1:15]
           id year stint team lg  g  ab  r  h X2b X3b hr rbi sb cs
4   ansonca01 1871     1  RC1    25 120 29 39  11   3  0  16  6  2
44  forceda01 1871     1  WS3    32 162 45 45   9   4  0  29  8  0
68  mathebo01 1871     1  FW1    19  89 15 24   3   1  0  10  2  1
99  startjo01 1871     1  NY2    33 161 35 58   5   1  1  34  4  2
102 suttoez01 1871     1  CL1    29 128 35 45   3   7  3  23  3  1
106 whitede01 1871     1  CL1    29 146 40 47   6   5  1  21  2  2
> tail(baseball)[,1:15]
             id year stint team lg   g  ab  r   h X2b X3b hr rbi sb cs
89523 biggicr01 2007     1  HOU NL 141 517 68 130  31   3 10  50  4  3
89525 benitar01 2007     2  FLO NL  34   0  0   0   0   0  0   0  0  0
89526 benitar01 2007     1  SFN NL  19   0  0   0   0   0  0   0  0  0
89530 ausmubr01 2007     1  HOU NL 117 349 38  82  16   3  3  25  6  1
89533  aloumo01 2007     1  NYN NL  87 328 51 112  19   1 13  49  3  0
89534 alomasa02 2007     1  NYN NL   8  22  1   3   1   0  0   0  0  0
```

1,2: Extract subset, fit and graph model

```
> # how many unique players?
> length(unique(baseball$id))
[1] 1228
> # examine career trajectory of baseball players in terms of rbi/ab
> # look at one player: Babe Ruth
> baberuth <- subset(baseball, id == "ruthba01")
> baberuth <- transform(baberuth, cyear= year - min(year) + 1)

> plot(rbi/ab ~ cyear, data=baberuth, type='b')
> BRmodel <- lm(rbi/ab ~ cyear, data=baberuth)
> abline(BRmodel, col="red")
```



3,4: Encapsulate in function, use **ply()

```
# apply transform() for all players
baseball <- ddply(baseball, .(id), transform, cyear= year - min(year) + 1)
```

define plot function for one player, with common scale:

```
xlim <- range(baseball$cyear, na.rm=TRUE)
ylim <- range(baseball$rbi/baseball$ab, na.rm=TRUE)
plotfun <- function(df) {
  plot(rbi/ab ~ cyear, data=df, type='b', xlim=xlim, ylim=ylim)
  abline(lm(rbi/ab ~ cyear, data=df, col="red"))
  }
```

use d_ply(): make plots for all 1128 players:

```
pdf("bbplots.pdf", width=8, height=4)
d_ply(baseball, .(reorder(id, rbi/abi)), plotfun)
dev.off()
```

3,4: Encapsulate in function, use **ply()

```
> # restrict ourselves to players with > 25 at bats
> bb <- subset(baseball, ab >= 25)
> length(unique(bb$id))
[1] 1152
```

Fit models: linear

```
> # function to fit one model
> model1 <- function(df) {
+   lm(rbi / ab ~ cyear, data=df)
+ }
> model1(baberuth)

Call:
lm(formula = rbi/ab ~ cyear, data = df)

Coefficients:
(Intercept)        cyear
   0.203200     0.003413

> # apply to all
> model1s <- dlply(bb, .(id), model1)
```

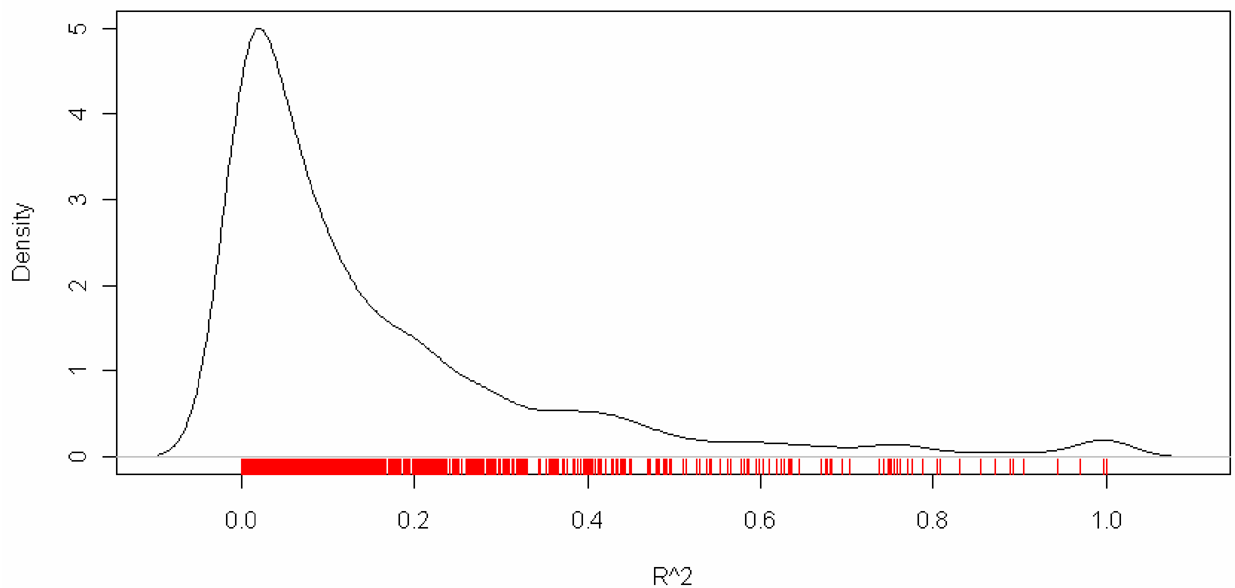
Fit models: quadratic

```
> # or, try a quadratic model
> model2 <- function(df) {
+   lm(rbi / ab ~ cyear + I(cyear^2), data=df)
+ }
> model2(baberuth)

Call:
lm(formula = rbi/ab ~ cyear + I(cyear^2), data
= df)

Coefficients:
(Intercept)        cyear   I(cyear^2)
  0.1267872    0.0225163   -0.0008306

> # apply to all
> model2s <- dlply(bb, .(id), model2)
```

We have a list of 1152 models, one for each player: summarize them:

```
> # extract an R^2 from a model
> rsq <- function(x) summary(x)$r.squared
>
> # summarize all
> summaries <- ldply(model1s, function(x) c(coef(x), rsquare = rsq(x)))
> names(summaries)[2:3] <- c("intercept", "slope")
> head(summaries,4)
         id  intercept         slope     rsquare
1 aaronha01 0.18329371  0.0001478121 0.000862425
2 abernte02 0.00000000            NA 0.000000000
3 adairje01 0.08670449 -0.0007118756 0.010230121
4 adamsba01 0.05905337  0.0012002168 0.030184694
```

```
> plot(density(summaries$rsquare, na.rm=TRUE), xlab="R^2", main="")
> rug(summaries$rsquare, col="red")
```



Where to go from here?

- This account (n=1) entirely impressionistic
  - Some features of programming languages
  - Characteristics: power, elegance, suggestivity, generality, economy, ...
- How to study empirically?
  - Experiment: tasks? population? language features? what needs to be controlled?
  - Survey: population? language features? what needs to be controlled?