R2

Ashish Mahabal
AyBi199, Caltech
24 May 2011

Running R

1. Create a separate sub-directory, say `work`, to h
   particular problem.

```
$ mkdir work
$ cd work
```

2. Start the R program with the command

```
$ R
```

3. At this point R commands may be issued (see
4. To quit the R program the command is

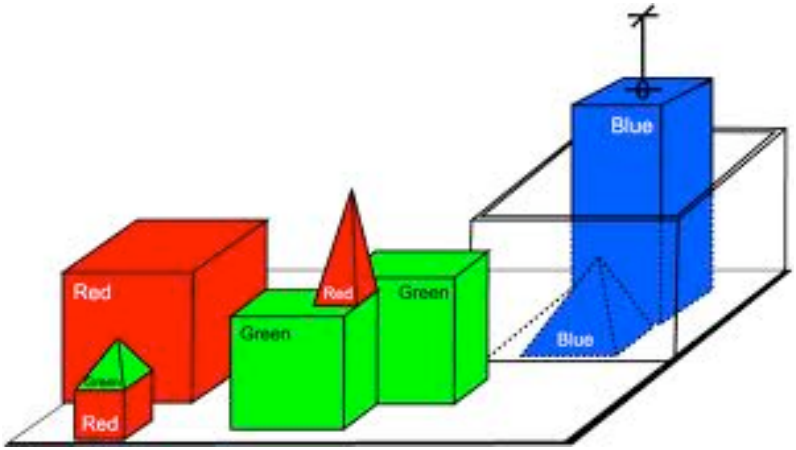
```
> q()
```

You can optionally save data.

Dealing with tables/objects (R frame)

- X = read.table("foo",header=TRUE)
- objects()
- objets(X)
- names(X)
- X
- Name1
- X$Name1
- attach(X)
- Name1

Num	Name1	Name2
1	1.1	3.3
2	4.4	5.4



Assignments

- x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
- assign("x", c(10.4, 5.6, 3.1, 6.4, 21.7))
- c(10.4, 5.6, 3.1, 6.4, 21.7) -> x (!)
- 1/x # 0.09,0.17,0.32,0.15,0.04
- y <- c(x,0,x) # 10.4,.. ,21.7,0,10.4,..,21.7
- v <- rep(x) + y + 1 # x repeated 2.2 times, 1 11
- var = sum((x-mean(x))^2)/(length(x)-1)

Lists (and data.frames=restricted)

- Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))
- Always numbered
- Lst[[4]] # returns [1] 4 7 9
- Lst[[4]][2] # returns 7
- Lst[4][2] # returns Null
- Lst[3] # returns $no.children [1] 3

Reading from files

- HousePrice <- read.table("houses.data", header=TRUE)
- read.table(file, header = FALSE, sep = "", quote = "\"'",
  dec = ".", row.names, col.names, as.is = !stringsAsFactors,
  na.strings = "NA", colClasses = NA, nrows = -1, skip = 0, check.names = TRUE, fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE,
  comment.char = "#", allowEscapes = FALSE, flush = FALSE,
  stringsAsFactors = default.stringsAsFactors(), fileEncoding = "",
  encoding = "unknown")

read.csv, read.csv2, read.delim, read.delim2

Accessing built-in datasets

- data()
- data(AirPassengers)
- ?AirPassengers
- new <- edit(AirPassengers)
- x<-array(c(AirPassengers[1:144]),dim=c(12,12))
- pairs(x)

- ? To get help
- # for comments
- -> or <- (or =) for assignments
- == for comparison

- Concatenate with c # x = c(a,b)
- seq to create sequence # z = seq(-30,30)
- is.na(x) to look for notavailableness

- summary(x)
- data(x)
- read.table(file)
- save(file)
- source(file) # input
- sink(file) # output

Common routines

- length(a) # length of vector
- max(a) # similarly min, mean
- sort(a), or sort(a, decreasing=T)
- unique(a)
- duplicated(a) # returns a logical array!

Tom Short's R ref card

http://www.rpad.org/Rpad/R-refcard.pdf

R Reference Card

by Tom Short, EPRI Solutions, Inc., tshort@episolutions.com 2005-07-12

Help and basics

Most R functions have online documentation.

help(topic) documentation on topic
?topic id.
help.search("topic") search the help system
apropos("topic") the names of all objects in the search list matching the regular expression "topic"
help.start() start the HTML version of help
str(a) display the internal *str*ucture of an R object
summary(a) gives a "summary" of a, usually a statistical summary but it is *generic* meaning it has different operations for different classes of a
ls() show objects in the search path; specify pat="pat" to search on a pattern
ls.str() str() for each variable in the search path
dir() show files in the current directory
methods(a) shows S3 methods of a
methods(class=class(a)) lists all the methods to handle objects of class a
options(...) set or examine many global options; common ones: width, digits, error
library(x) load add-on packages; library(help=x) lists datasets and functions in package x.
attach(x) database x to the R search path; x can be a list, data frame, or R data file created with save. Use search() to show the search path.
detach(x) x from the R search path; x can be a name or character string of an object previously attached or a package.

Input and output

load() load the datasets written with save
data(x) loads specified data sets
read.table(file) reads a file in table format and creates a data frame from it; the default separator sep="" is any whitespace; use header=TRUE to read the first line as a header of column names; use as.is=TRUE to prevent character vectors from being converted to factors; use comment.char="" to prevent "#" from being interpreted as a comment; use skip=n to skip n lines before reading data; see the help for options on row naming, NA treatment, and others
read.csv("filename",header=TRUE) id. but with defaults set for reading comma-delimited files
read.delim("filename",header=TRUE) id. but with defaults set for reading tab-delimited files
read.fwf(file,widths,header=FALSE,sep="",as.is=FALSE) read a table of *f*ixed width *f*ormatted data into a 'data.frame'; widths is an integer vector, giving the widths of the fixed-width fields
save(file,...) saves the specified objects (...) in the XDR platform-independent binary format
save.image(file) saves all objects
cat(..., file="", sep=" ") prints the arguments after coercing to character; sep is the character separator between arguments
print(a, ...) prints its arguments; generic, meaning it can have different methods for different objects
format(x,...) format an R object for pretty printing
write.table(x,file="",row.names=TRUE,col.names=TRUE, sep=" ") prints x after converting to a data frame; if quote is TRUE, character or factor columns are surrounded by quotes ("); sep is the field separator; eol is the end-of-line separator; na is the string for missing values; use col.names=NA to add a blank column header to get the column headers aligned correctly for spreadsheet input
sink(file) output to file, until sink()

Most of the I/O functions have a file argument. This can often be a character string naming a file or a connection. file="" means the standard input or output. Connections can include files, pipes, zipped files, and R variables.

On windows, the file connection can also be used with description = "clipboard". To read a table copied from Excel, use
x <- read.delim("clipboard")
To write a table to the clipboard for Excel, use
write.table(x,"clipboard",sep="\t",col.names=NA)
For database interaction, see packages RODBC, DBI, RMySQL, RPgSQL, and ROracle. See packages XML, hdf5, netCDF for reading other file formats.

Data creation

c(...) generic function to combine arguments with the default forming a vector; with recursive=TRUE descends through lists combining all elements into one vector
from:to generates a sequence; ":" has operator priority; 1:4 + 1 is "2,3,4,5"
seq(from,to) generates a sequence by= specifies increment; length= specifies desired length
seq(along=x) generates 1, 2, ..., length(x); useful for for loops
rep(x,times) replicate x times; use each= to repeat "each" element of x each times; rep(c(1,2,3),2) is 1 2 3 1 2 3; rep(c(1,2,3),each=2) is 1 1 2 2 3 3
data.frame(...) create a data frame of the named or unnamed arguments; data.frame(v=1:4,ch=c("a","B","c","d"),n=10); shorter vectors are recycled to the length of the longest
list(...) create a list of the named or unnamed arguments; list(a=c(1,2),b="hi",c=3i);
array(x,dim=) array with data x; specify dimensions like dim=c(3,4,2); elements of x recycle if x is not long enough
matrix(x,nrow=,ncol=) matrix; elements of x recycle
factor(x,levels=) encodes a vector x as a factor
gl(n,k,length=n*k,labels=1:n) generate levels (factors) by specifying the pattern of their levels; k is the number of levels, and n is the number of replications
expand.grid() a data frame from all combinations of the supplied vectors or factors
rbind(...) combine arguments by rows for matrices, data frames, and others
cbind(...) id. by columns

Slicing and extracting data

Indexing lists

x[n]	list with elements n
x[[n]]	n^{th} element of the list
x[["name"]]	element of the list named "name"
x$name	id.

Indexing vectors

x[n]	n^{th} element
x[-n]	all *but* the n^{th} element
x[1:n]	first n elements
x[-(1:n)]	elements from n+1 to the end
x[c(1,4,2)]	specific elements
x["name"]	element named "name"
x[x > 3]	all elements greater than 3
x[x > 3 & x < 5]	all elements between 3 and 5
x[x %in% c("a","and","the")]	elements in the given set

Indexing matrices

x[i,j]	element at row i, column j
x[i,]	row i
x[,j]	column j
x[,c(1,3)]	columns 1 and 3
x["name",]	row named "name"

Indexing data frames (matrix indexing plus the following)

x[["name"]]	column named "name"
x$name	id.

Variable conversion

as.array(x), as.data.frame(x), as.numeric(x), as.logical(x), as.complex(x), as.character(x), ... convert type; for a complete list, use methods(as)

Variable information

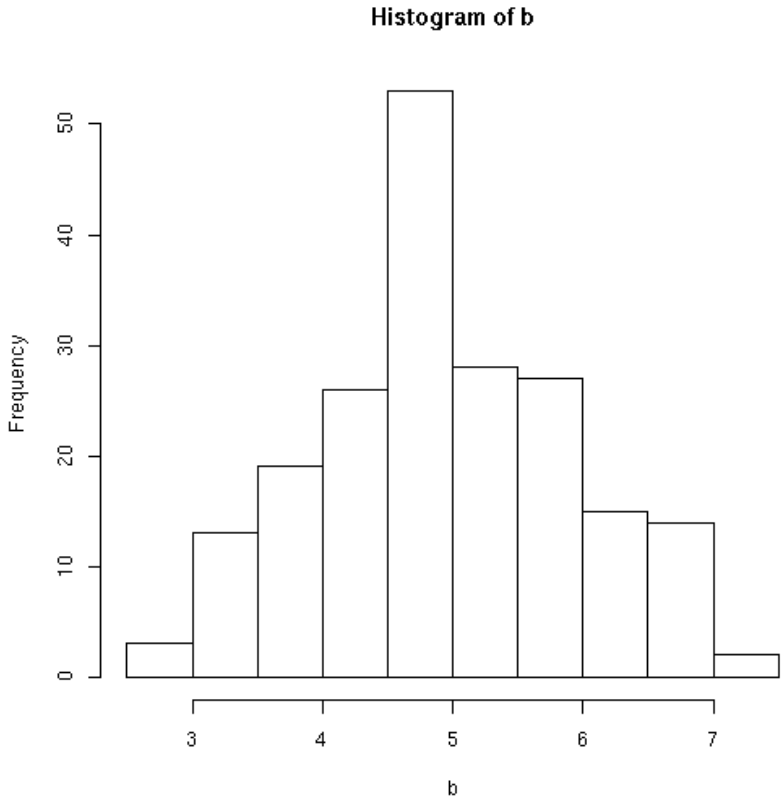
is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ... test for type; for a complete list, use methods(is)
length(x) number of elements in x
dim(x) Retrieve or set the dimension of an object; dim(x) <- c(3,2)
dimnames(x) Retrieve or set the dimension names of an object
nrow(x) number of rows; NROW(x) is the same but treats a vector as a one-row matrix
ncol(x) and **NCOL(x)** id. for columns
class(x) get or set the class of x; class(x) <- "myclass"
unclass(x) remove the class attribute of x
attr(x,which) get or set the attribute which of x
attributes(obj) get or set the list of attributes of obj

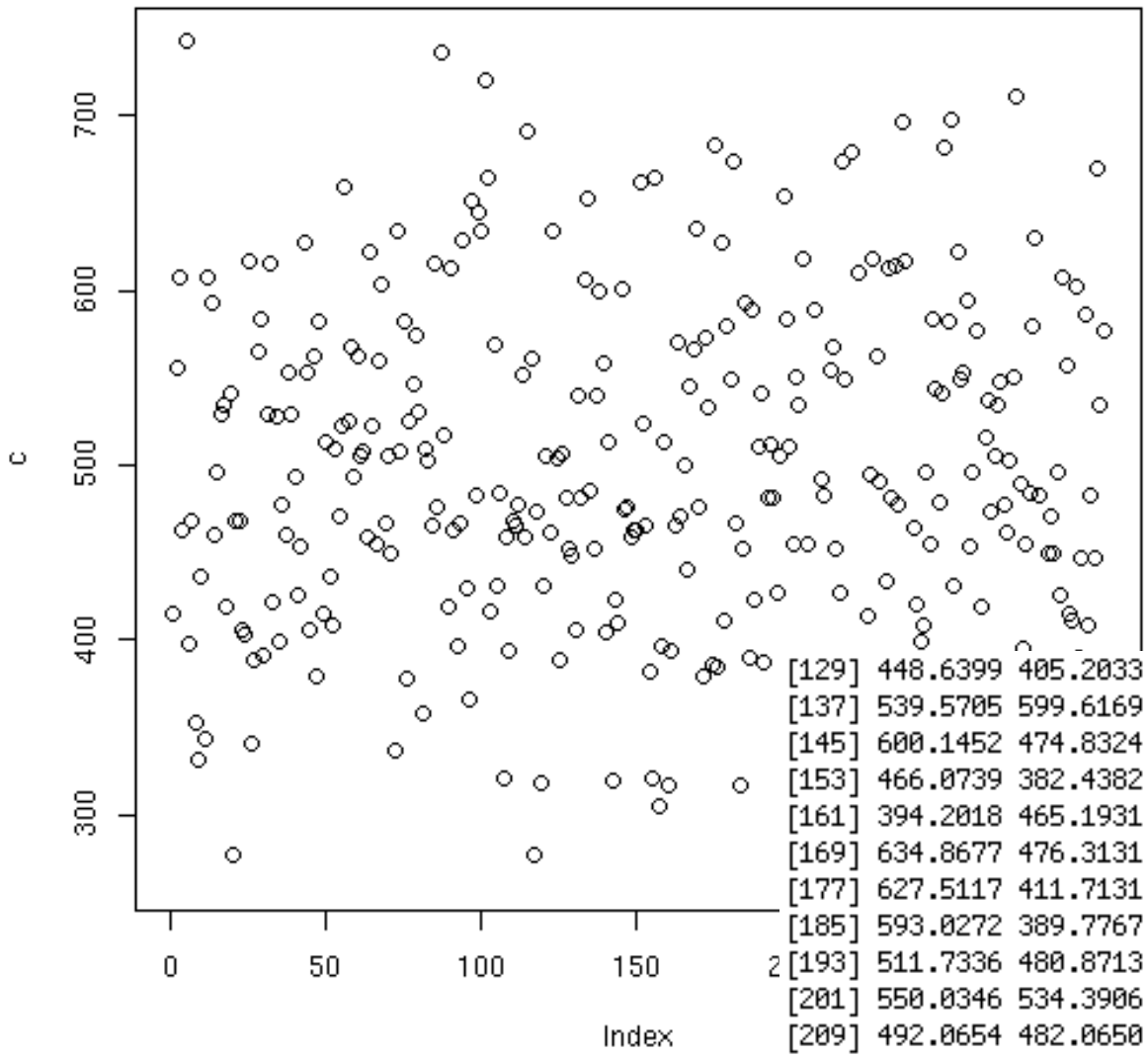
Data selection and manipulation

which.max(x) returns the index of the greatest element of **x**
which.min(x) returns the index of the smallest element of **x**
rev(x) reverses the elements of x
sort(x) sorts the elements of x in increasing order; to sort in decreasing order: rev(sort(x))
cut(x,breaks) divides x into intervals (factors); breaks is the number of cut intervals or a vector of cut points

Example

- a=rnorm(100,mean=5,sd=1)
- b=rnorm(200,mean=5,sd=1)
- c=c(100*a,100*b)
- length(c)
- c





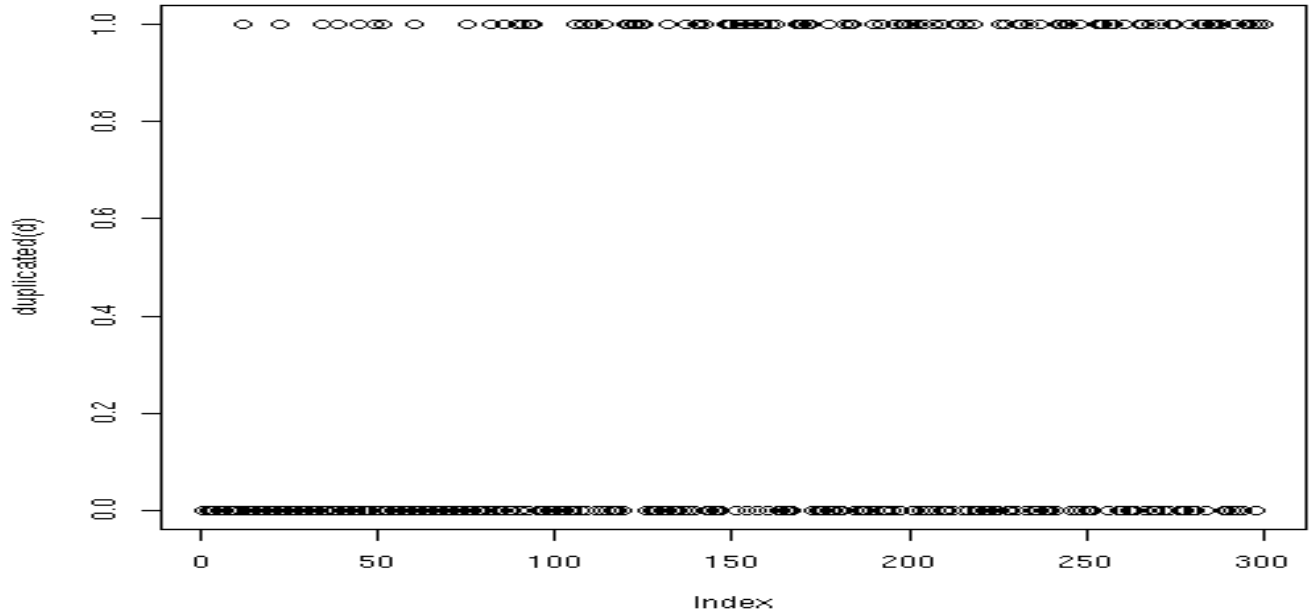

```
[129] 448.6399 405.2033 539.6339 481.5814 605.8925 652.4077 485.9140 452.6308
[137] 539.5705 599.6169 558.6897 403.9841 513.5366 320.0797 423.3735 409.3903
[145] 600.1452 474.8324 475.7574 458.8818 462.2530 462.1789 662.1314 524.2215
[153] 466.0739 382.4382 320.3050 664.2152 304.6201 396.7627 513.1186 316.9793
[161] 394.2018 465.1931 570.3309 471.2018 499.7137 439.7604 545.3611 565.8577
[169] 634.8677 476.3131 378.7135 573.0420 533.1097 386.3176 683.4459 383.8697
[177] 627.5117 411.7131 579.6656 548.9674 673.4191 466.5636 316.8249 451.9777
[185] 593.0272 389.7767 588.8479 422.8503 510.6999 541.5611 387.2153 481.9646
[193] 511.7336 480.8713 426.9614 504.6129 653.7176 583.9883 510.9250 454.2808
[201] 550.0346 534.3906 618.4641 389.1341 455.4347 351.0636 588.9437 373.3234
[209] 492.0654 482.0650 302.8964 554.8883 567.3488 451.8892 427.2502 673.1516
[217] 549.4872 340.9521 678.7345 324.3261 609.4050 271.7941 347.2002 413.5027
[225] 494.1476 618.3846 561.6571 490.4388 311.9638 433.9501 612.2427 481.6557
[233] 613.3592 477.4250 695.8766 616.0761 379.0668 307.3831 464.5840 420.0470
[241] 398.5331 408.9758 496.4556 454.2031 583.3175 543.1616 478.5739 541.4109
[249] 681.4943 581.5734 697.4596 431.3465 622.0025 549.1973 552.8481 594.3782
[257] 454.1116 496.1235 576.2675 369.9352 419.1967 515.2750 537.4536 473.0187
[265] 505.1537 534.9228 547.8853 476.7180 461.7504 502.4479 550.4487 710.9494
[273] 489.0445 394.7242 455.1865 484.4981 580.1016 630.0980 482.1229 264.0277
```

- d=as.integer(c)
- d
- mean(d); max(d); min(d); sd(d)

```
> a
  [1] 415 556 607 462 742 397 468 352 331 435 343 607 592 459 496 528 534 419
 [19] 541 277 467 467 406 403 617 340 388 565 583 391 529 614 421 528 399 476
 [37] 460 553 528 492 425 453 627 552 406 562 379 582 415 513 435 408 509 470
 [55] 522 658 524 567 493 562 504 507 458 622 521 454 559 603 466 505 449 336
 [73] 633 508 582 378 525 546 573 530 358 508 501 465 615 476 735 516 419 612
 [91] 462 396 467 627 429 366 650 483 643 634 719 664 416 569 430 483 320 458
[109] 394 467 465 477 551 458 690 561 276 472 317 430 505 460 633 504 388 506
[127] 481 451 448 405 539 481 605 652 485 452 539 599 558 403 513 320 423 409
[145] 600 474 475 458 462 462 662 524 466 382 320 664 304 396 513 316 394 465
[163] 570 471 499 439 545 565 634 476 378 573 533 386 683 383 627 411 579 548
[181] 673 466 316 451 593 389 588 422 510 541 387 481 511 480 426 504 653 583
[199] 510 454 550 534 618 389 455 351 588 373 492 482 302 554 567 451 427 673
[217] 549 340 678 324 609 271 347 413 494 618 561 490 311 433 612 481 613 477
[235] 695 616 379 307 464 420 398 408 496 454 583 543 478 541 681 581 697 431
[253] 622 549 552 594 454 496 576 369 419 515 537 473 505 534 547 476 461 502
[271] 550 710 489 394 455 484 580 630 482 264 359 449 470 450 496 425 607 556
[289] 414 410 601 389 446 585 408 482 446 670 533 576
```

```
[109] FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE F
[121]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE F
[133] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE
[145] FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE F
[157] FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE F
[169]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE F
[181] FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
[193] FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
[205] FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE
[217] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[229] FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE F
[241] FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE F
```

- unique(d)
- duplicated(d)
- plot(duplicated(d)) # coercion

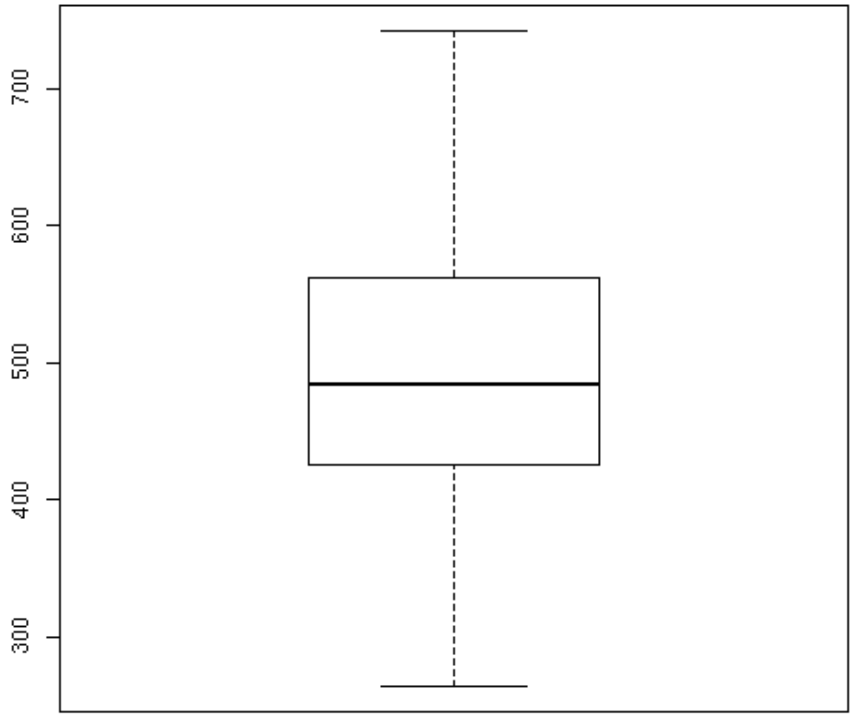


- duplicated(unique(d))
- five(d) # whats the five-of-d
- ?five # nothing? Do you know five?
- ??five # at least something that has five?
- fivenum(d) # ah, so its fivenum. I'll take that
- ?fivenum # what is it BTW?

… minimum, lower-hinge, median,upper-hinge, maximum …

- fivenum(x, na.rm = TRUE) # options!
- g=d
- g[duplicated(d)]<-NA
- fivenum(d)
  - [1] 264.0 425.5 483.5 561.5 742.0
- fivenum(g,na.rm = TRUE)
  - [1] 264.0 419.5 492.0 571.5 742.0

- boxplot.stats(x, coef = 1.5, do.conf = TRUE, do.out = TRUE)
- quantile, range, bxp



- which selects indices:
- a=which(d<500) # returns indices
- a=d[which(d<500)] # returns the elements
- which.max(d) # index of max element

```
 [1]   1   4   6   7   8   9  10  11  14  15
[19]  30  33  35  36  37  40  41  42  45  47
[37]  71  72  76  81  84  86  89  91  92  93
[55] 109 110 111 112 114 117 118 119 120 122 1
[73] 140 142 143 144 146 147 148 149 150 153 1
[91] 165 166 170 171 174 176 178 182 183 184 1
[109] 205 206 208 209 210 211 214 215 218 220 2
[127] 234 237 238 239 240 241 242 243 244 247 2
[145] 273 274 275 276 279 280 281 282 283 284 2
[163] 297
```

```
  [1] 415 462 397 468 352 331 435 343 459 496 419
 [19] 391 421 399 476 460 492 425 453 406 379 415
 [37] 449 336 378 358 465 476 419 462 396 467 429
 [55] 394 467 465 477 458 276 472 317 430 460 388
 [73] 403 320 423 409 474 475 458 462 462 466 382
 [91] 499 439 476 378 386 383 411 466 316 451 389
[109] 455 351 373 492 482 302 451 427 340 324 271
[127] 477 379 307 464 420 398 408 496 454 478 431
[145] 489 394 455 484 482 264 359 449 470 450 496
[163] 446
```

- match(x,y) # elements of x in y, else NA
- merge(a,b) # using common columns/rows

```
> t1=read.table('table1',header=TRUE)
> t1
  id    ra   dec
1 101 200.1  33.1
2 102 199.3 -13.3
3 103 200.2  19.1
> t2=read.table('table2',header=TRUE)
> t2
  id mag
1 101  17
2 102  18
3 104  19
```

```
> merge(t1,t2)
  id    ra   dec mag
1 101 200.1  33.1  17
2 102 199.3 -13.3  18
```

choose(n,k) # combinations of k from n

choose(5,3) returns 10

sample(x,size) # resamples size elements from x (with the option of replacement)

```
> cards= paste(c("C","D","H","S"), rep(1:13,times=4), sep="")
> cards
 [1] "C1"  "D2"  "H3"  "S4"  "C5"  "D6"  "H7"  "S8"  "C9"  "D10" "H11" "S12"
[13] "C13" "D1"  "H2"  "S3"  "C4"  "D5"  "H6"  "S7"  "C8"  "D9"  "H10" "S11"
[25] "C12" "D13" "H1"  "S2"  "C3"  "D4"  "H5"  "S6"  "C7"  "D8"  "H9"  "S10"
[37] "C11" "D12" "H13" "S1"  "C2"  "D3"  "H4"  "S5"  "C6"  "D7"  "H8"  "S9"
[49] "C10" "D11" "H12" "S13"
```

- > sample(cards,5)

[1] "S9" "H4" "D5" "C3" "C9"

- > sample(cards,2)

[1] “S1" “S11”

> sample(cards, 60, replace=TRUE)

- Conj(4+3i)
- fft(x)
- solve(a) # matrix inverse of a
- solve(a,b) # solves a %*% x = b for x

Graphics in R

R graphics, comparison

- Base graphics (Ross Ihaka, S graphics driver)
  - Can not be modified
- Grid graphics (Paul Murrell, 2000)
  - Low level, but no support for statistical graphics
- Lattice package (Sarkar, 2008)
  - Based on grid. Detailed. Lacks a formal model
- ggplot2 (Hedly, 2005)
  - Layered, based on grammar of graphics, low level, static plots, extensions easy, flow like that of analysis
- ggobi, rggobi (Cook, Swayne, Wickham 2007,8)
  - Interactive, grammar of graphics

http://cran.r-project.org/web/views/Graphics.html **(online list)**

Grammar of graphics (Wilkinson, 2005)

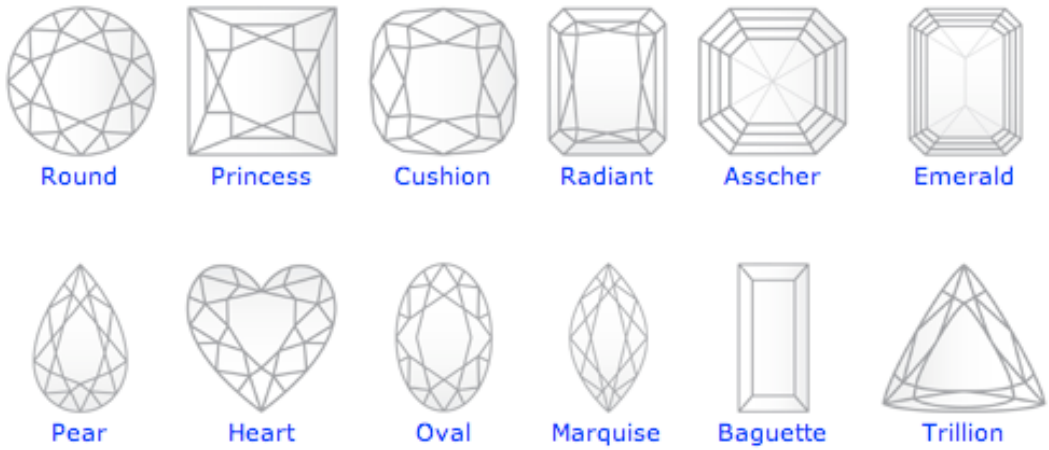
- Mapping (***mappings***) from data (***data***) to aesthetic attributes of geometric objects (***geoms***).
- Including statistical transformations (***stats***) in specific coordinate systems
- Faceting (***facet***) to allow easy subsetting (synonyms: conditioning, latticing, trellising)
- ***scale, coord***

Hadley Wickham's ggplot2
http://had.co.nz/ggplot2/

- install.packages("ggplot2")
- library(ggplot2)

Diamonds
http://www.diamondse.info/

- help(diamonds)
- data(diamonds)
- dsmall <- diamonds[sample(nrow(diamonds), 100),]
- dsmall
- help(ggsave)



Aesthetics: color, size, shape

ggsave(filename="diamond_smooth.png", plot=last_plot(), height=3, width=5, dpi=100)

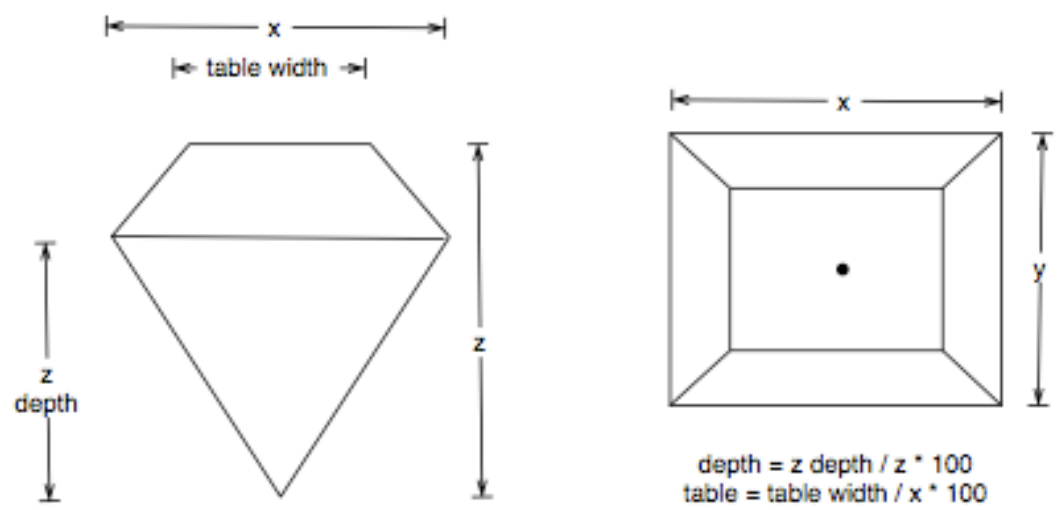
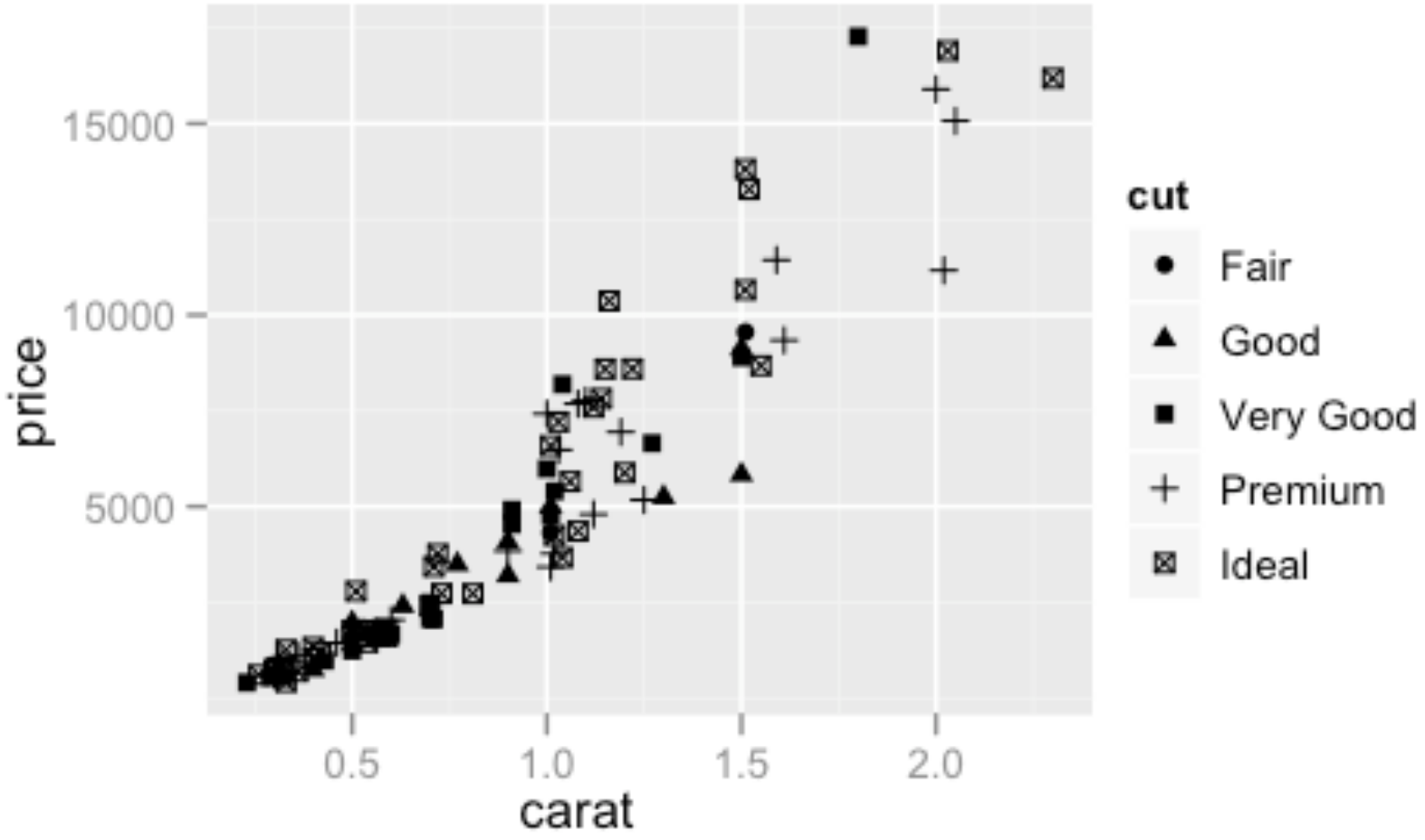


Fig. 2.1: How the variables x, y, z, table and depth are measured.

Carat
cut
color
clarity
(categories)

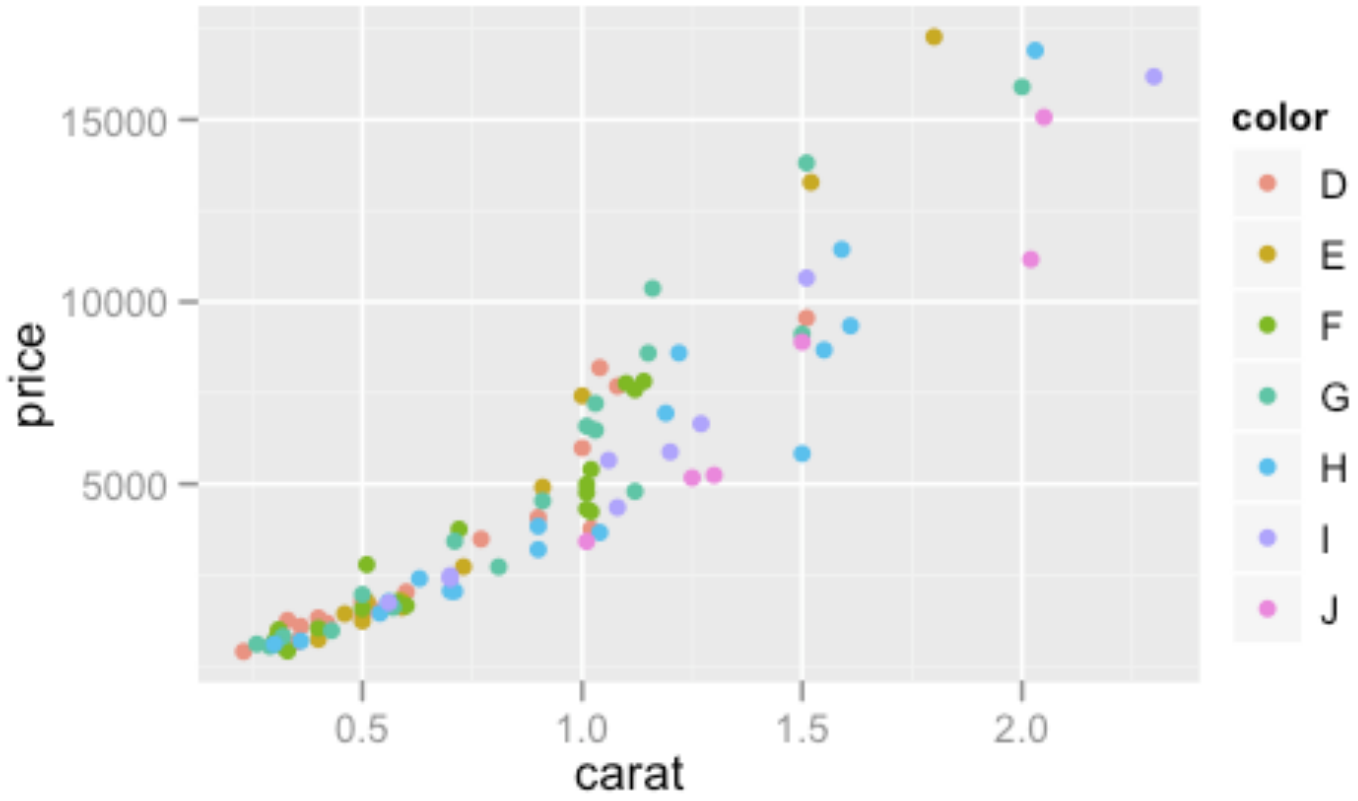
	carat	cut	color	clarity	depth	table	price	x	y	z
35183	0.31	Ideal	G	IF	62.0	54.0	891	4.36	4.38	2.71
205	0.98	Fair	H	SI2	67.9	60.0	2777	6.05	5.97	4.08
52175	0.70	Premium	E	SI2	62.8	59.0	2475	5.69	5.64	3.56
39992	0.41	Ideal	F	VS2	61.1	56.0	1107	4.83	4.80	2.94
2776	0.90	Good	G	SI2	61.8	56.0	3253	6.11	6.16	3.79
22560	1.43	Premium	G	VS2	62.2	58.0	10609	7.18	7.15	4.46
22586	1.35	Ideal	H	VVS1	61.9	57.0	10639	7.06	7.09	4.38
44761	0.52	Very Good	E	VS2	62.8	55.0	1621	5.11	5.15	3.22
47749	0.54	Ideal	D	VS2	61.3	56.0	1892	5.22	5.26	3.21
34734	0.37	Ideal	D	VS2	61.0	56.0	876	4.65	4.70	2.85
4107	0.78	Ideal	F	VS1	61.6	56.0	3537	5.90	5.92	3.64

qplot(carat, price, data=dsmall, shape=cut)



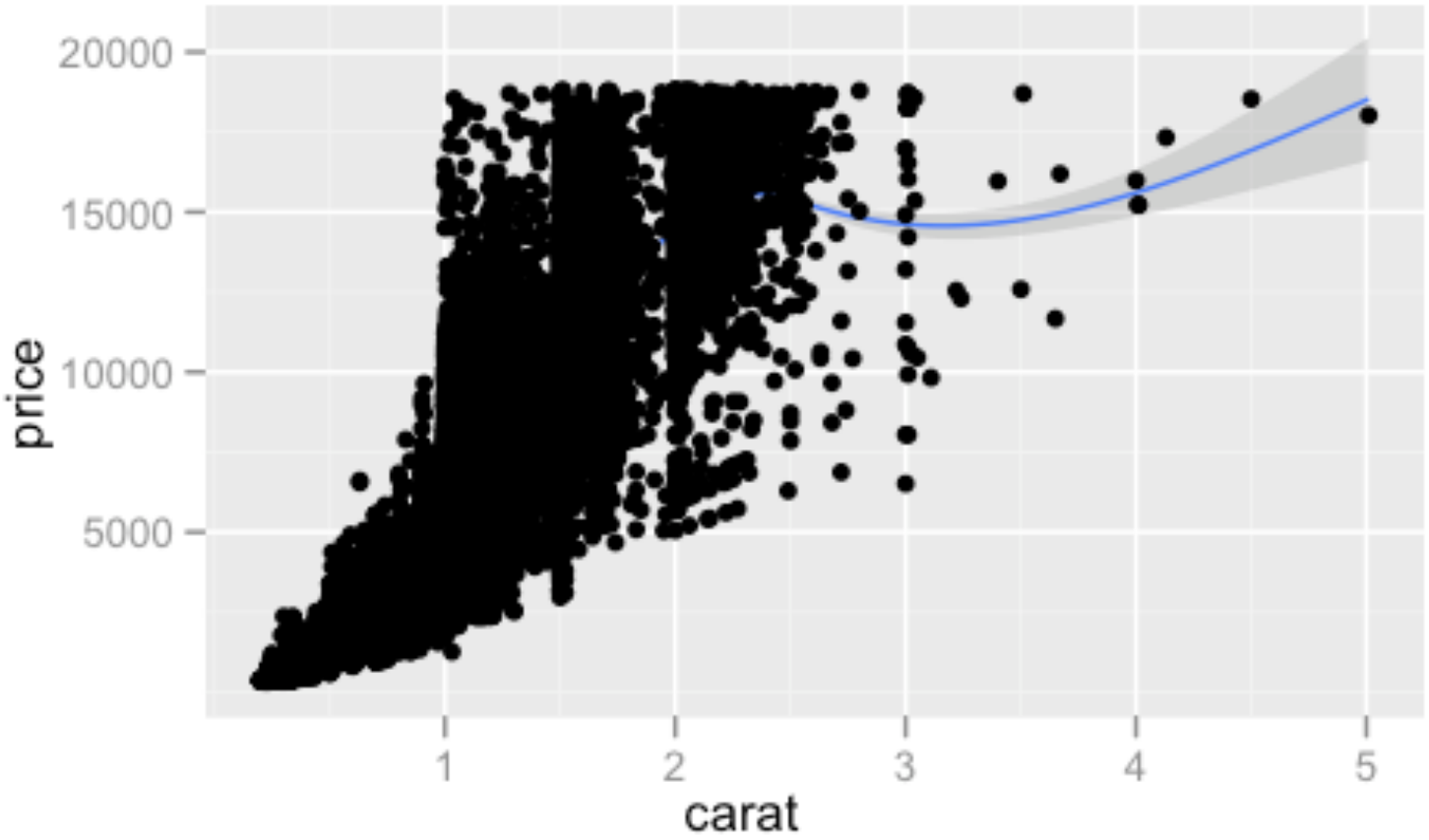
Shape is the aesthetic here and a categorical variable

qplot(carat, price, data=dsmall, colour=color)



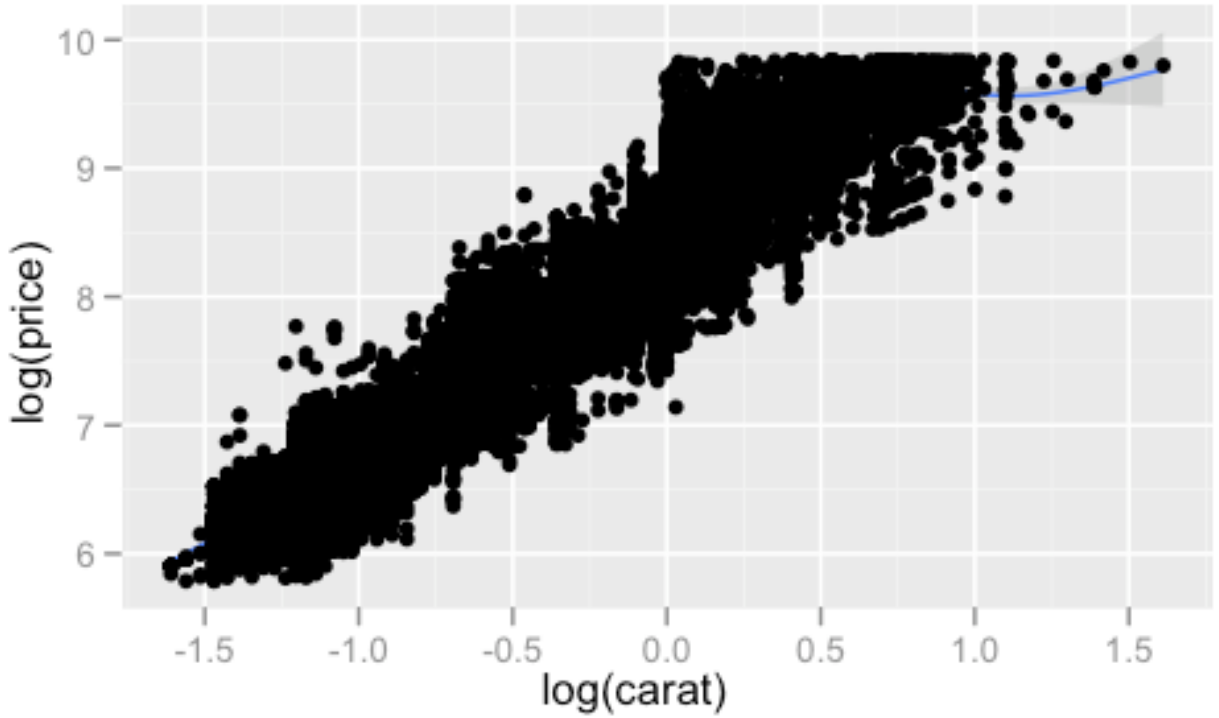
color is the aesthetic here and a categorical variable

qplot(carat, price, data=diamonds, geom=c("smooth","point"))



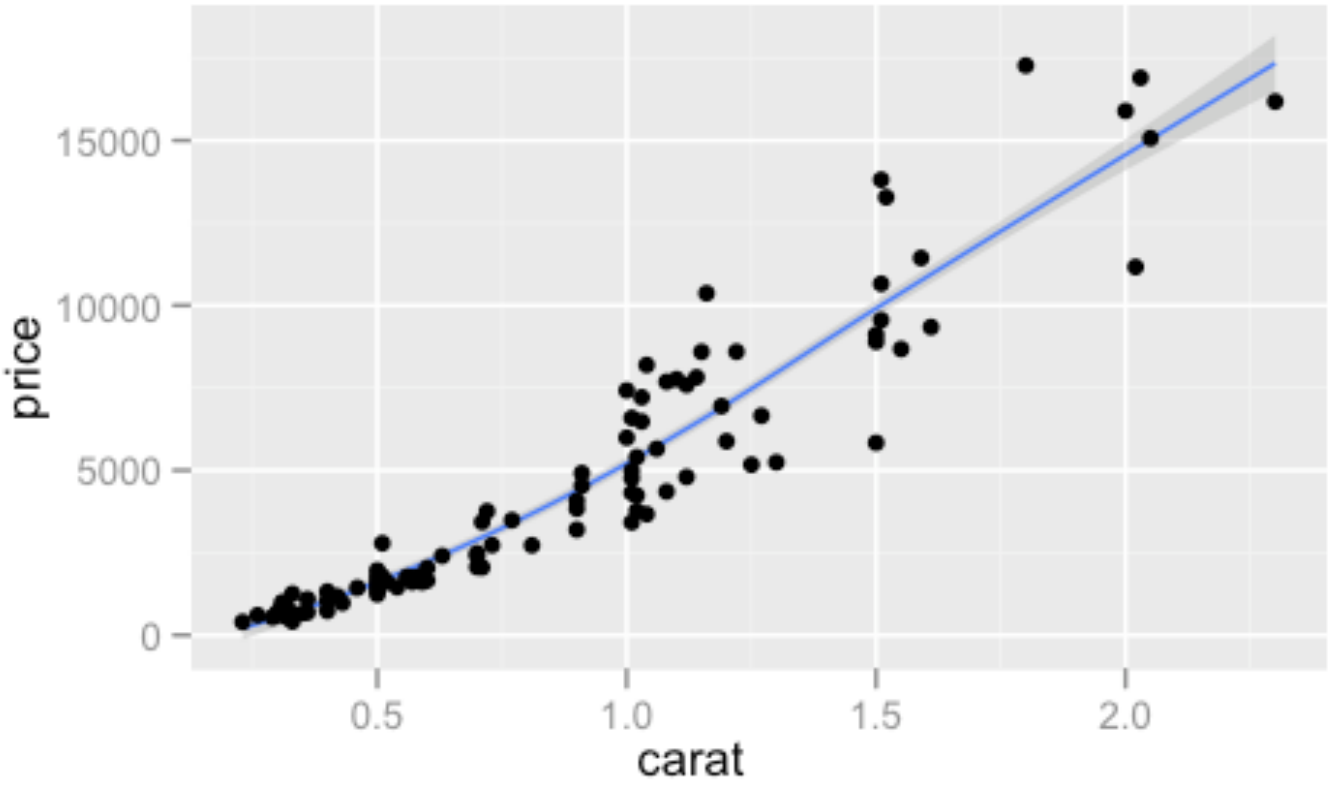
An example of a statistical plot

qplot(log(carat), log(price), data=diamonds, geom=c("smooth","point"))

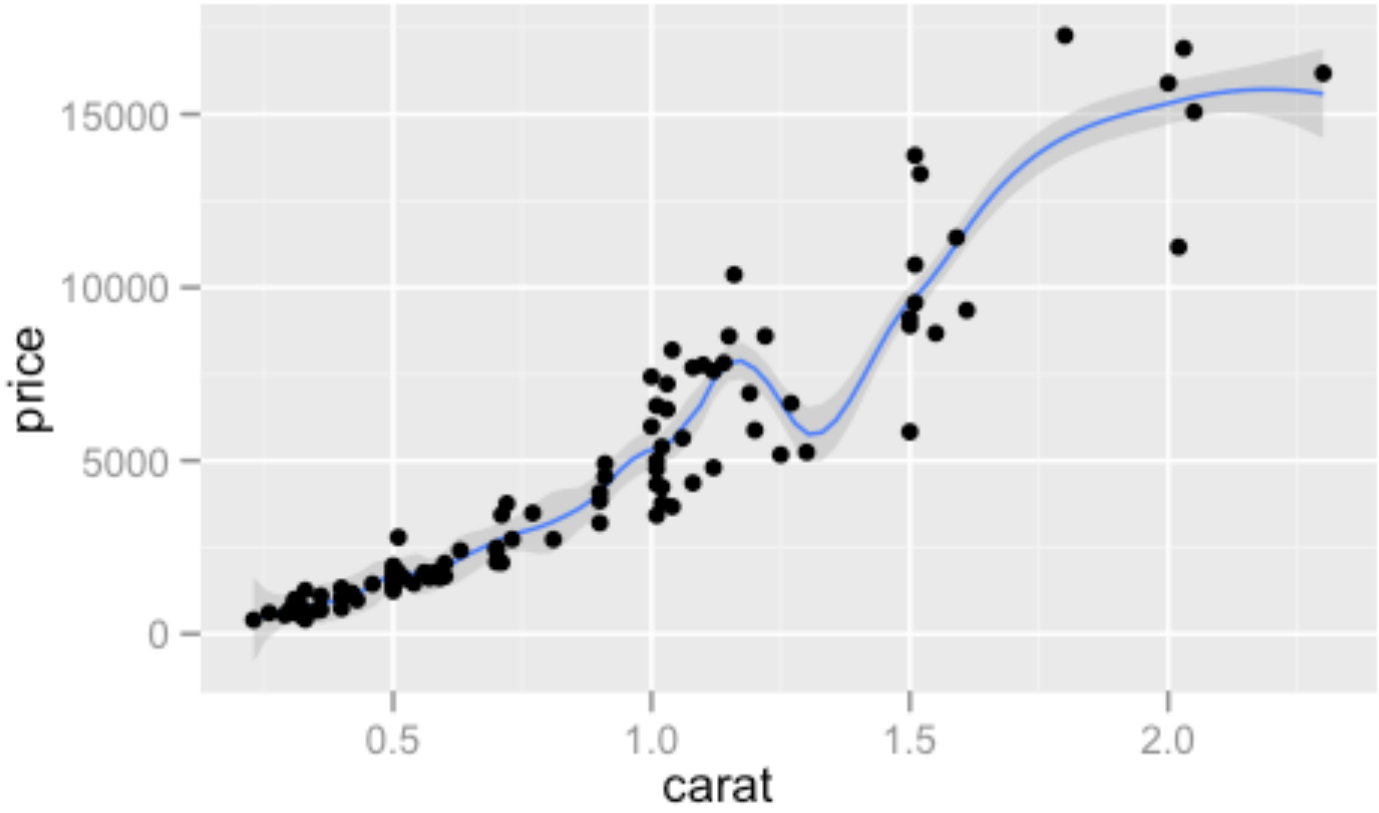


se=False to turn confidence level off

qplot(carat, price, data=dsmall, geom=c("smooth","point"), span=1)

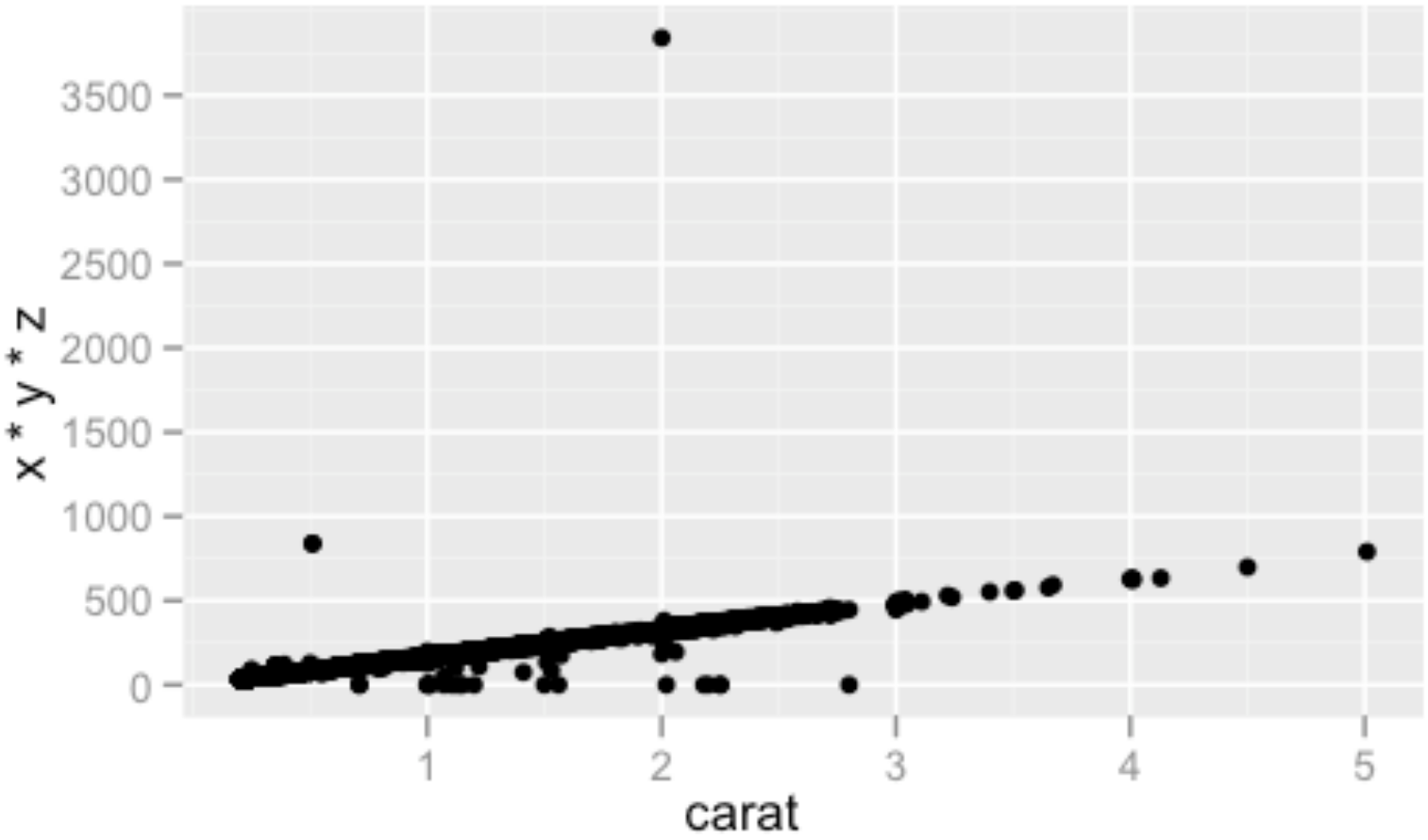


qplot(carat, price, data=dsmall, geom=c("smooth","point"), span=0.2)



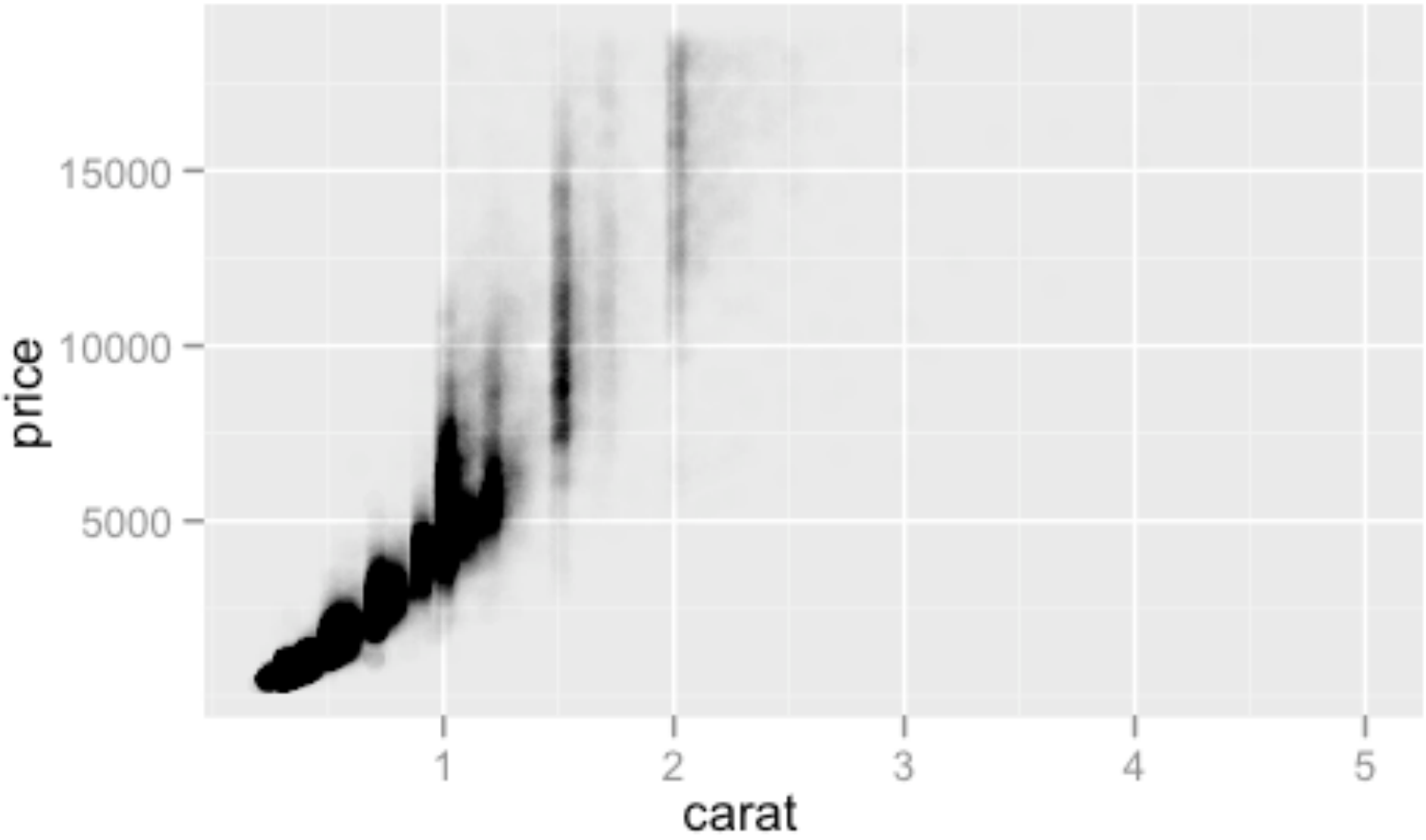
Span can be varied between 0 (very wiggly) and 1 (not wiggly)

qplot(carat, x*y*z, data=diamonds)



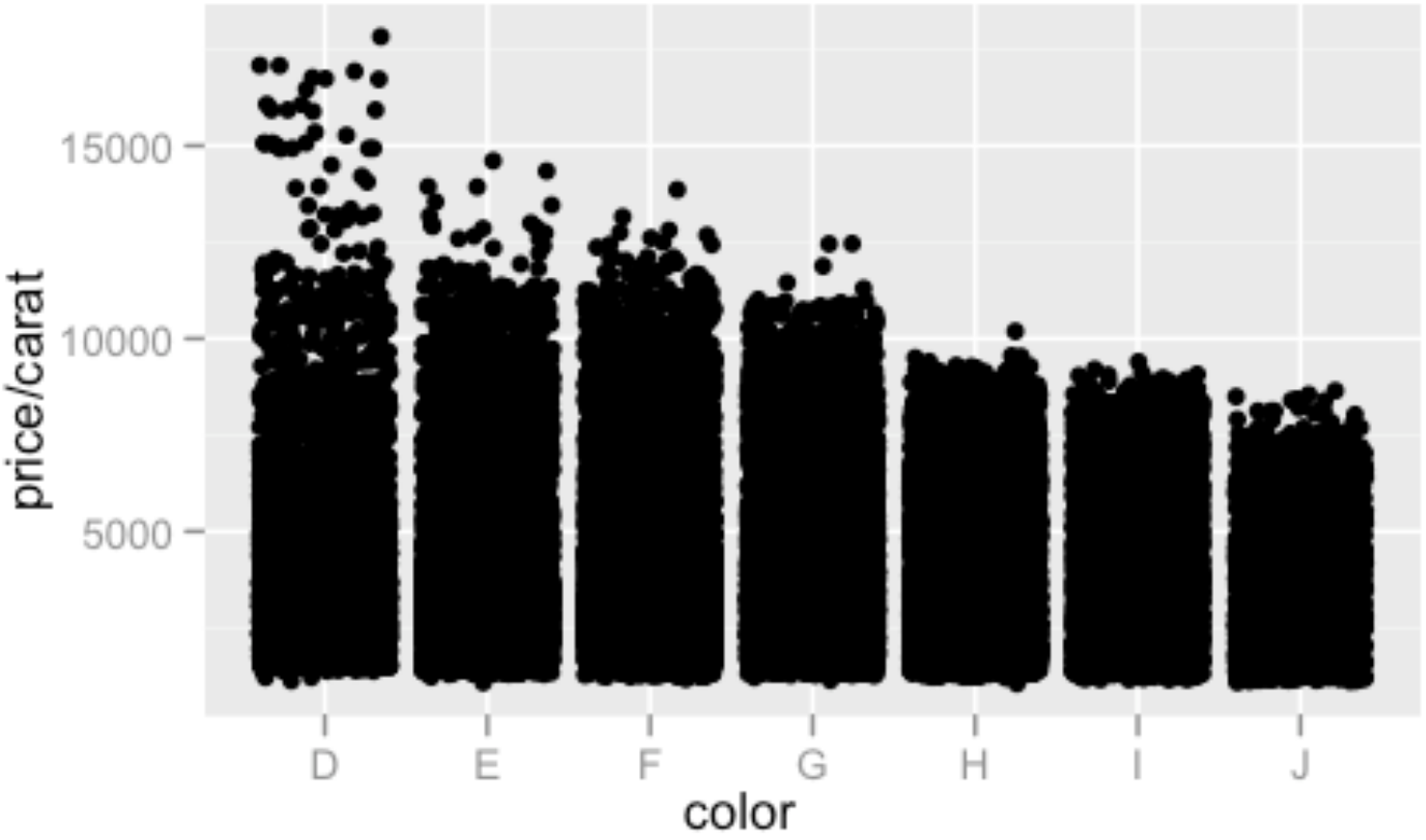
Easy to use multiple variables. Outliers by volume

qplot(carat, price, data=diamonds, alpha=I(1/500))

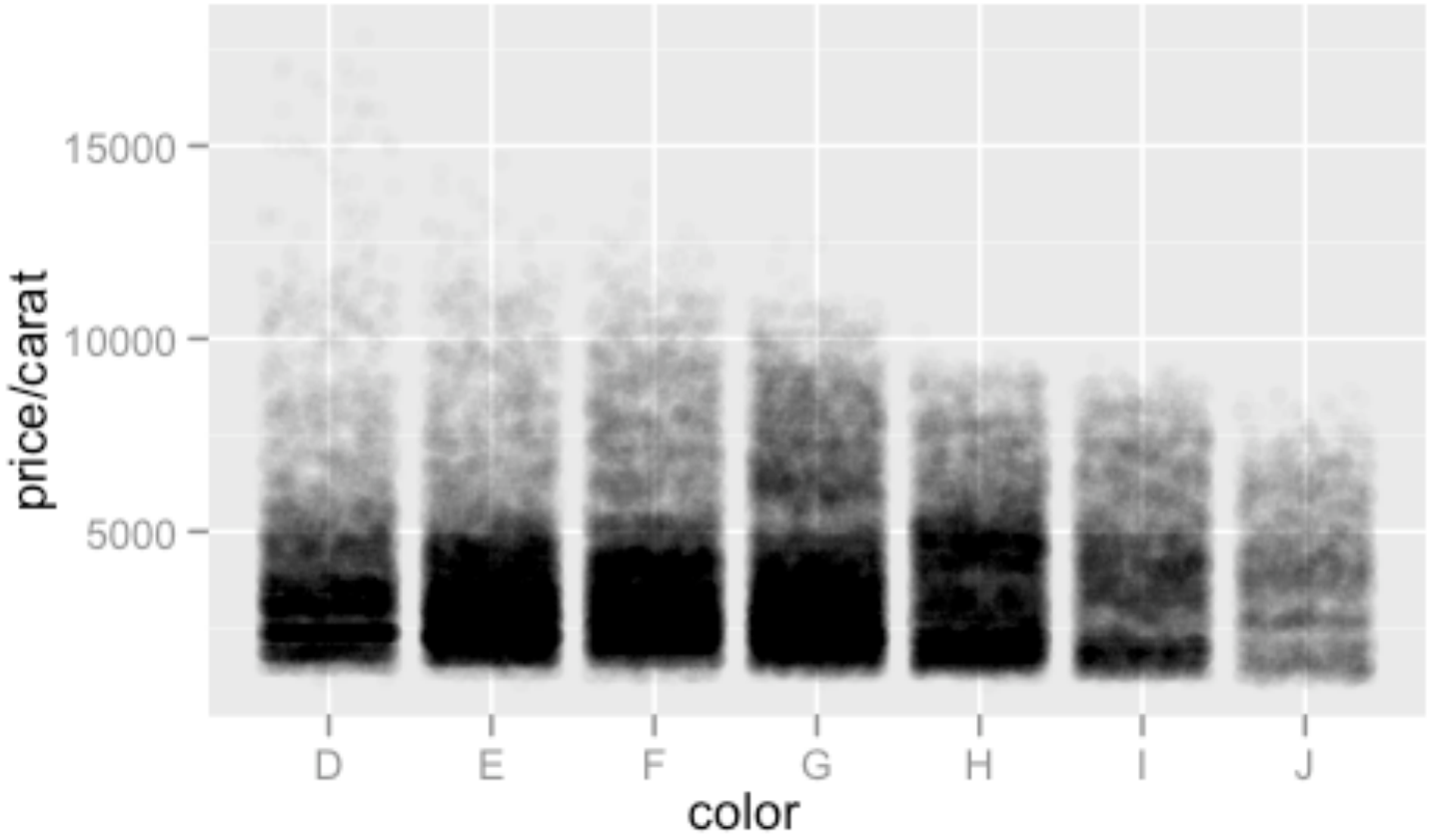


Transparency (scale 0 to 1): 1/1 => not transparent

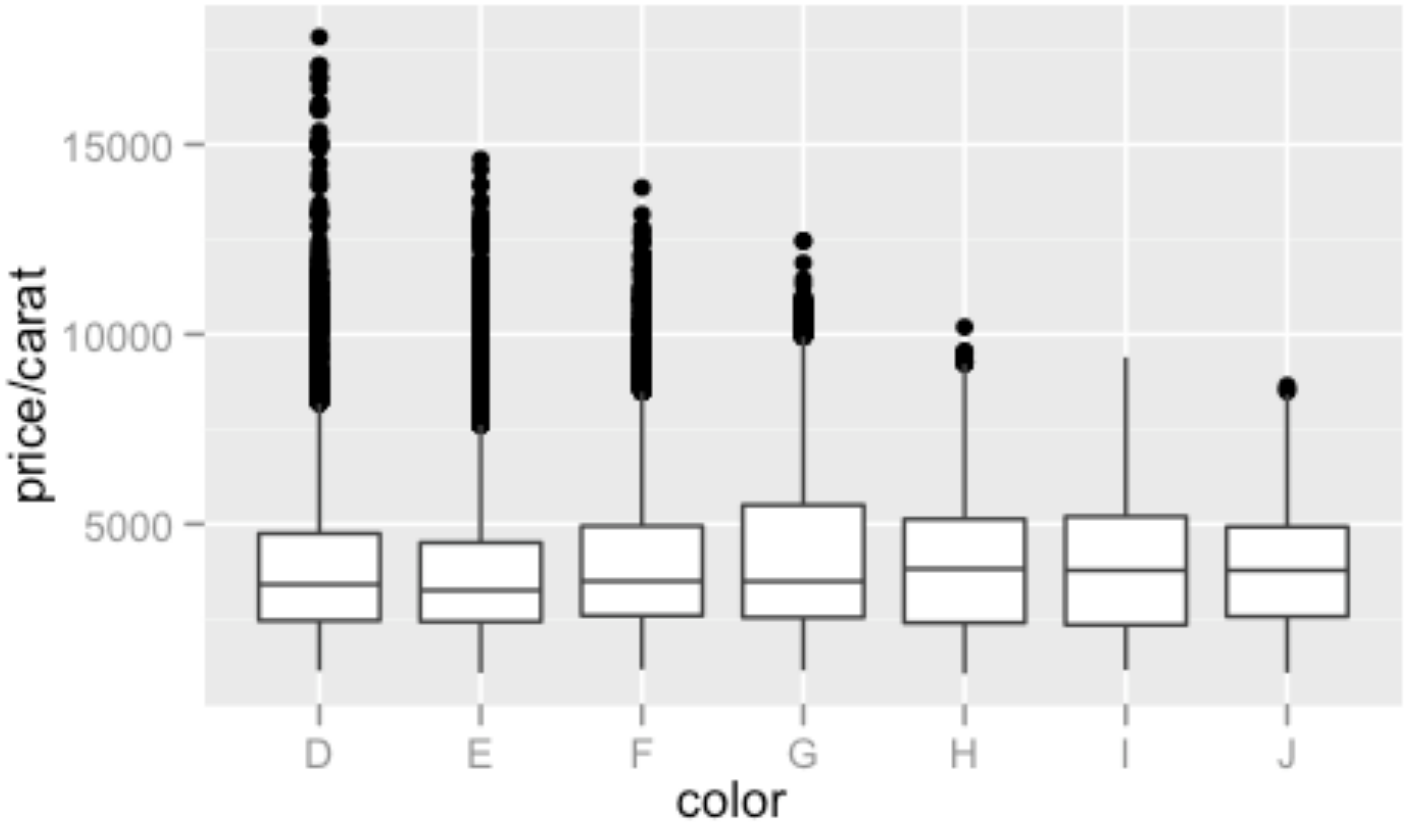
qplot(color, price / carat, data = diamonds, geom = "jitter")



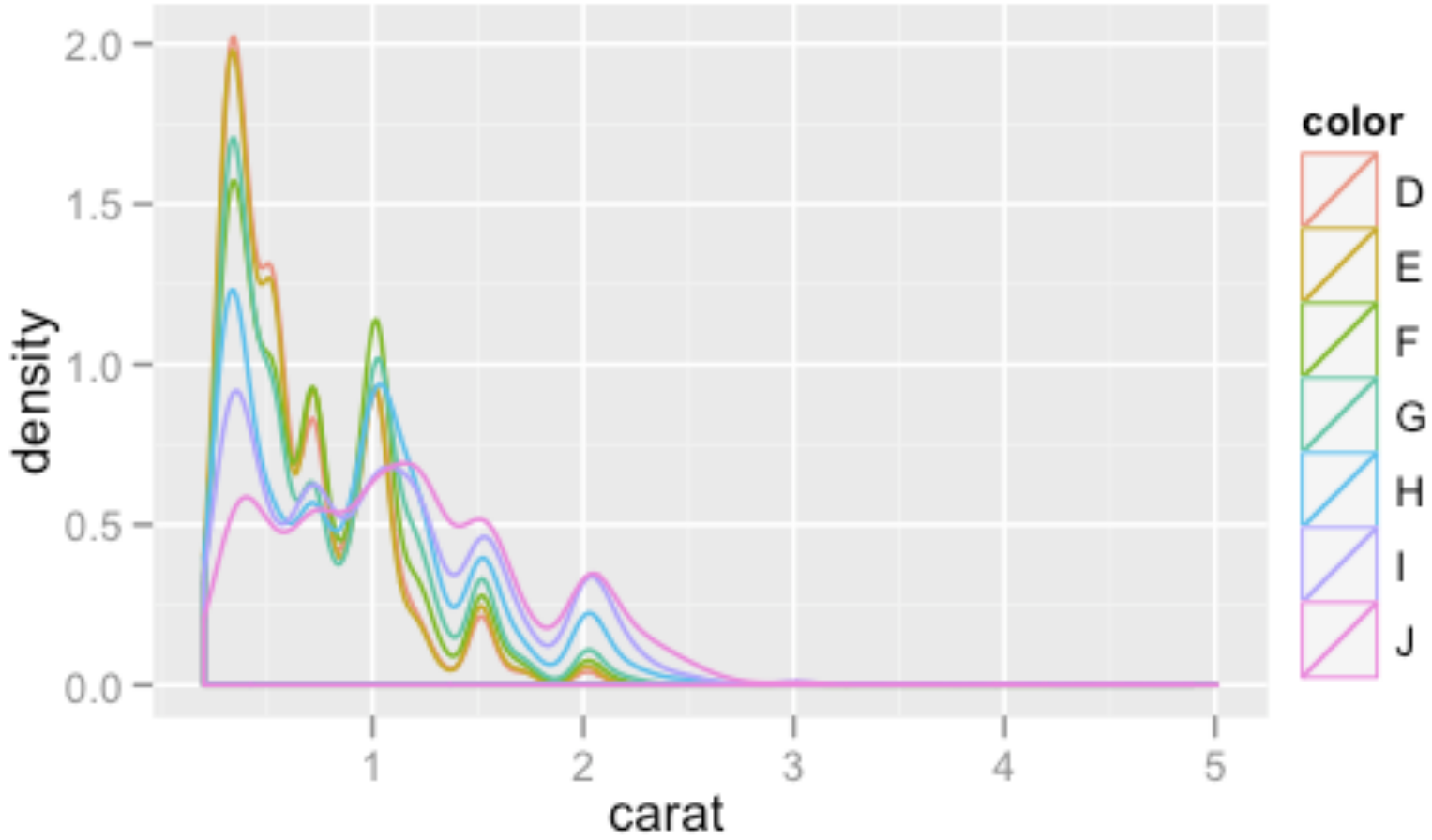
qplot(color, price / carat, data = diamonds, geom = "jitter", alpha=I(1/50))



qplot(color, price / carat, data = diamonds, geom = "boxplot")



qplot(carat, data=diamonds, geom="density", colour = color)



Combining aesthetic and geom

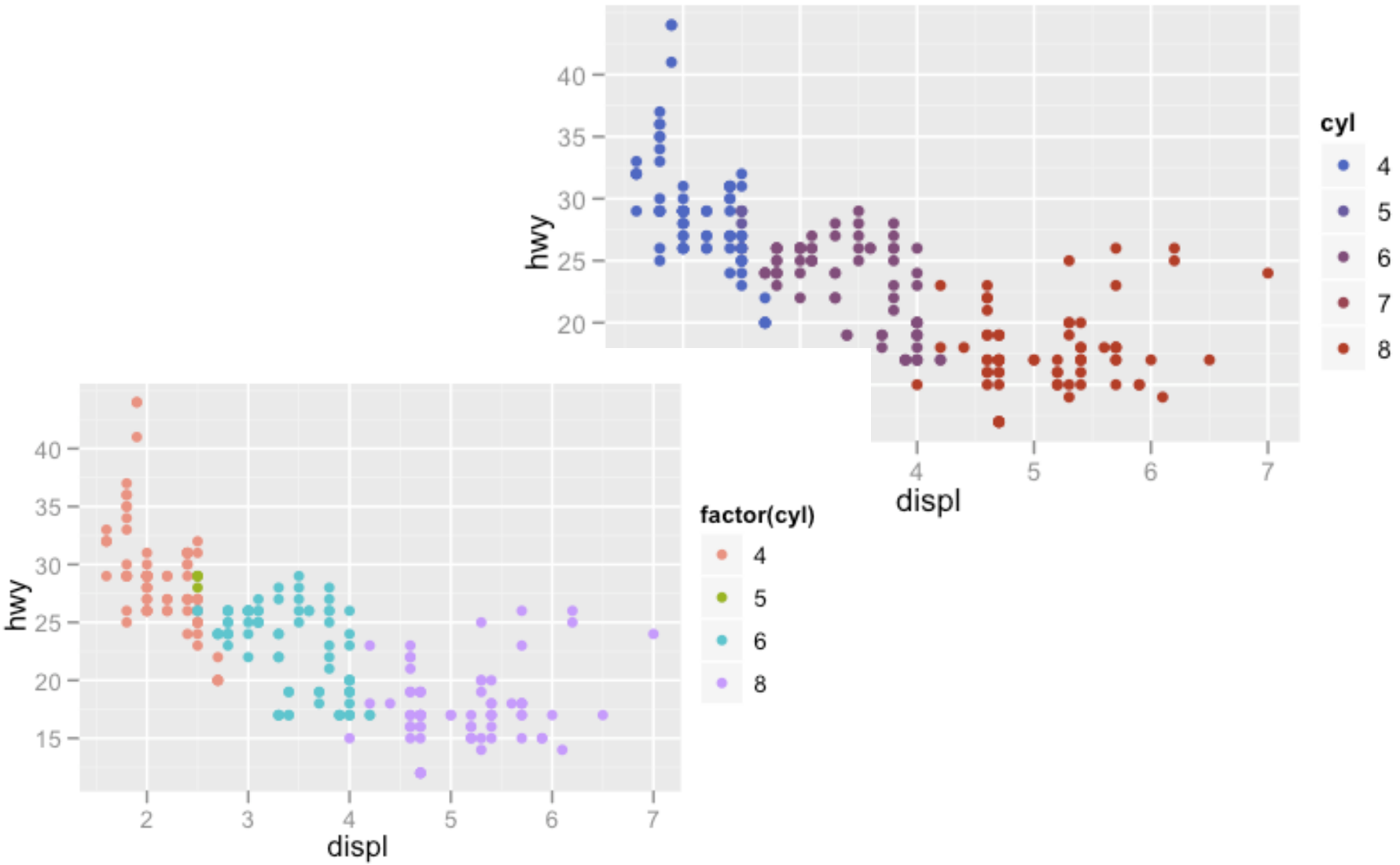
Fuel efficiency data http://fueleconomy.gov

- data(mpg)
- help(mpg)
- 234 rows, 11 vars
- 1999-2008 popular cars

Treehugger.com

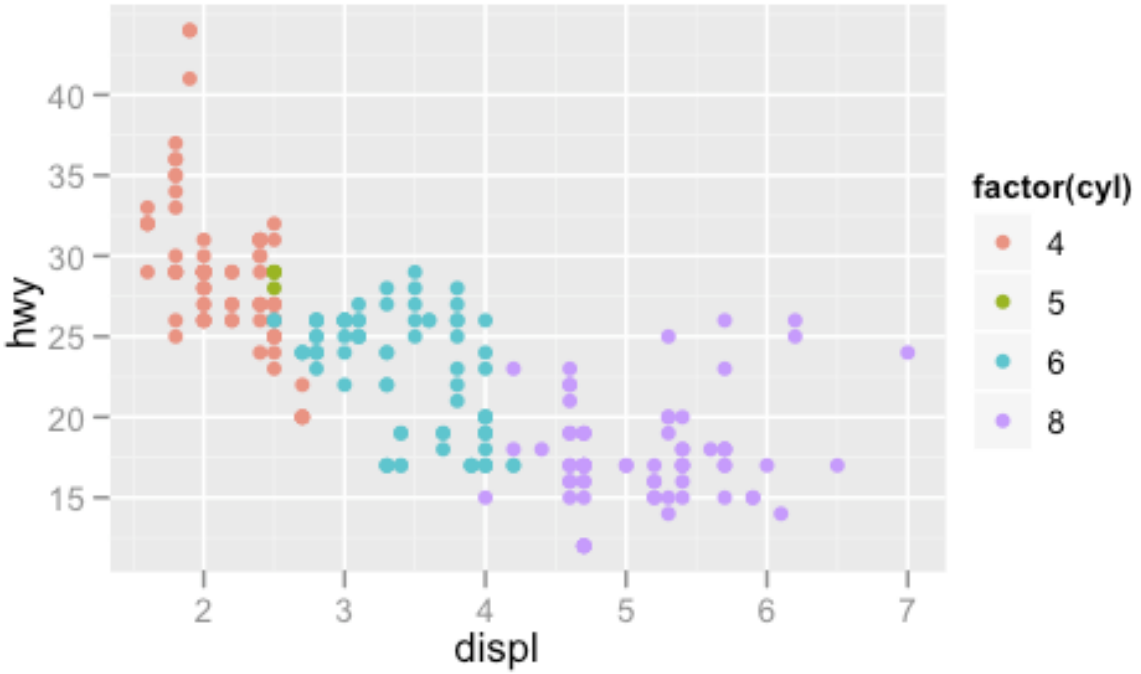
manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl
chevrolet	k1500 tahoe 4wd	5.3	2008	8	auto(l4)	4	14	19	r
chevrolet	corvette	7.0	2008	8	manual(m6)	r	15	24	p
audi	a4 quattro	2.0	2008	4	auto(s6)	4	19	27	p
dodge	ram 1500 pickup 4wd	4.7	2008	8	auto(l5)	4	13	17	r
toyota	camry solara	2.4	2008	4	manual(m5)	f	21	31	r
subaru	forester awd	2.5	2008	4	manual(m5)	4	20	27	r
honda	civic	1.6	1999	4	manual(m5)	f	28	33	r
chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(l4)	r	14	20	r
chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(l4)	r	11	15	e
dodge	dakota pickup 4wd	4.7	2008	8	auto(l5)	4	14	19	r

qplot(displ, hwy, data = mpg, colour = cyl)

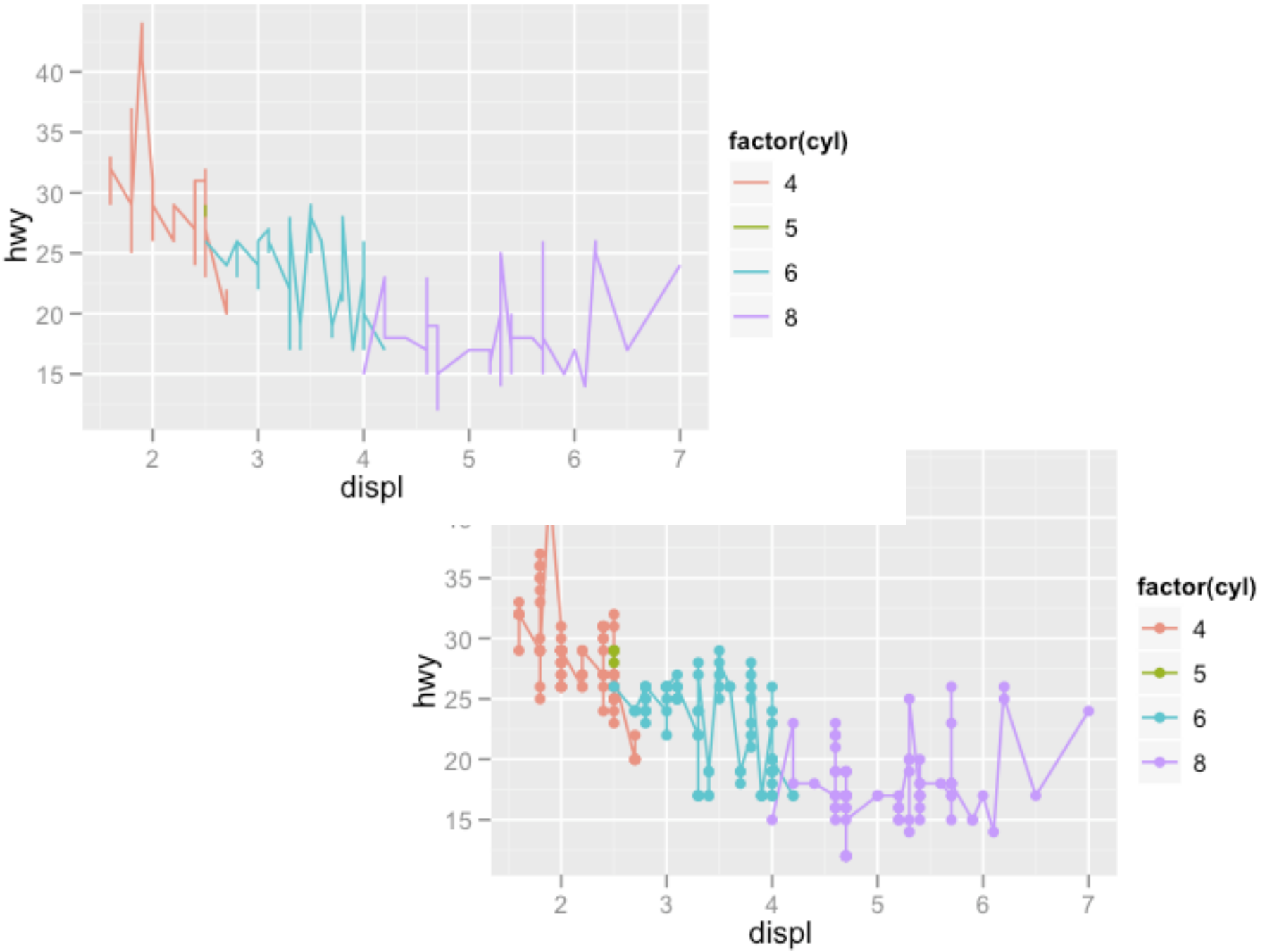


qplot(displ, hwy, data = mpg, colour = factor(cyl))

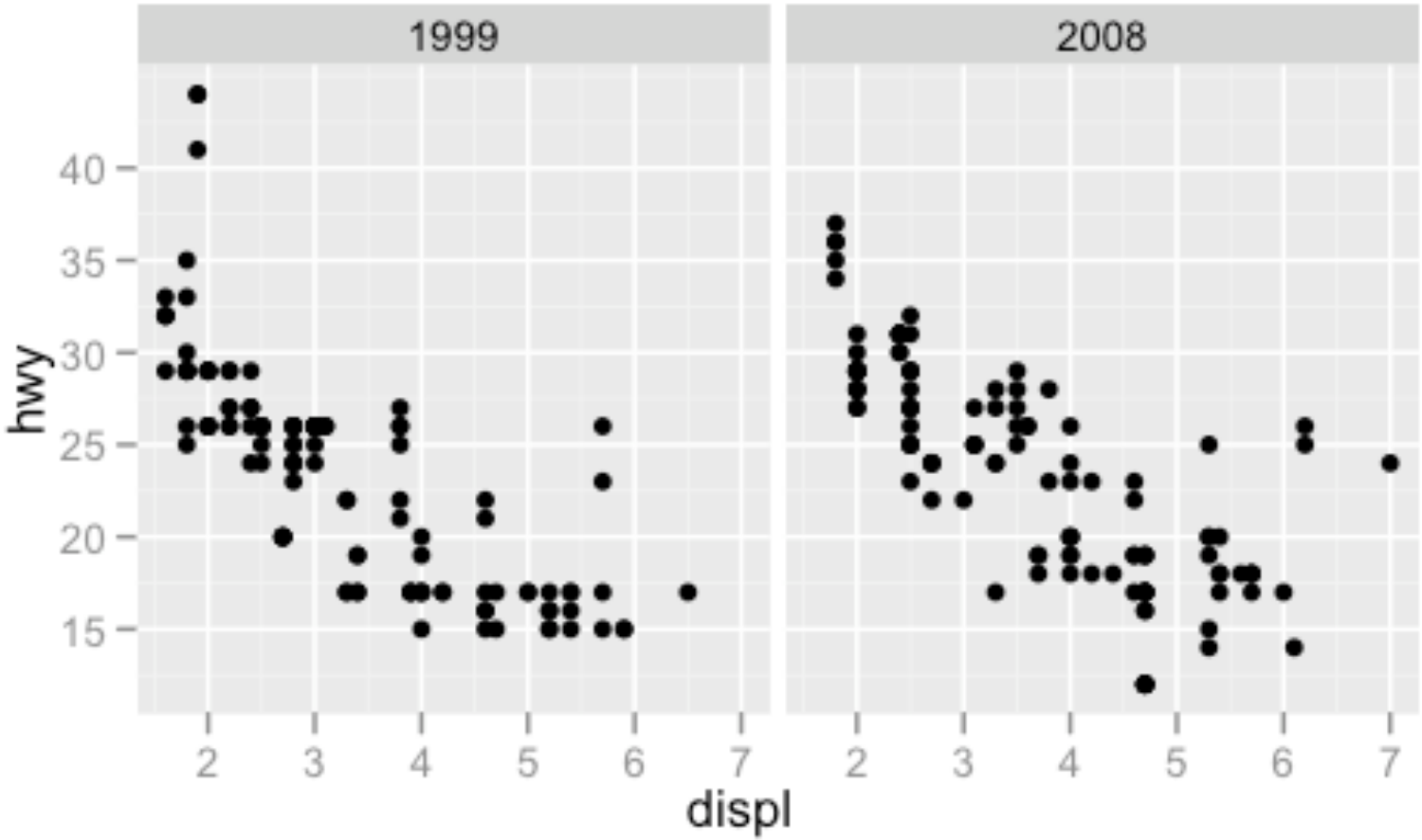
aesthetics: color; geom: point



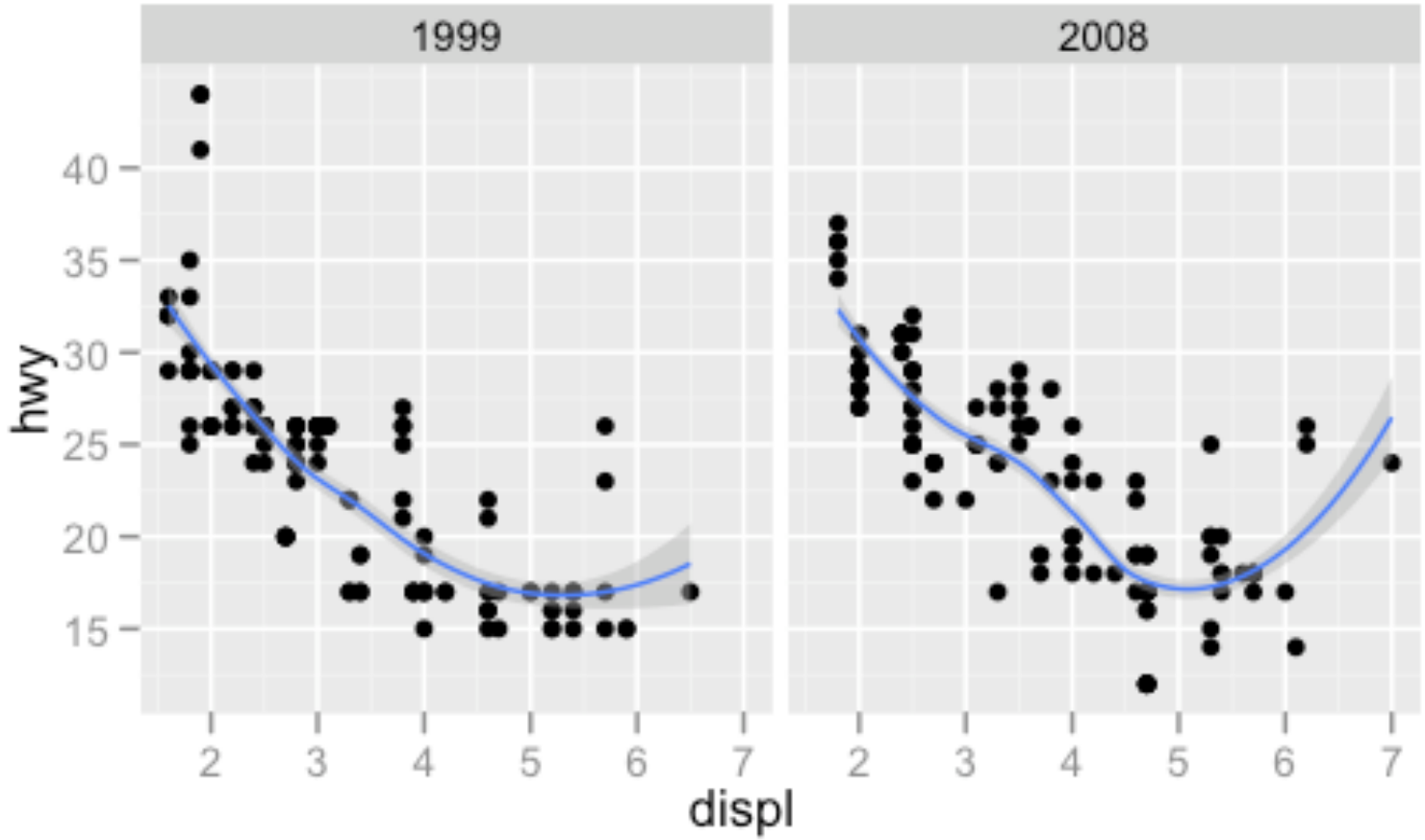
The geom could as easily have been “line” or even point and line!

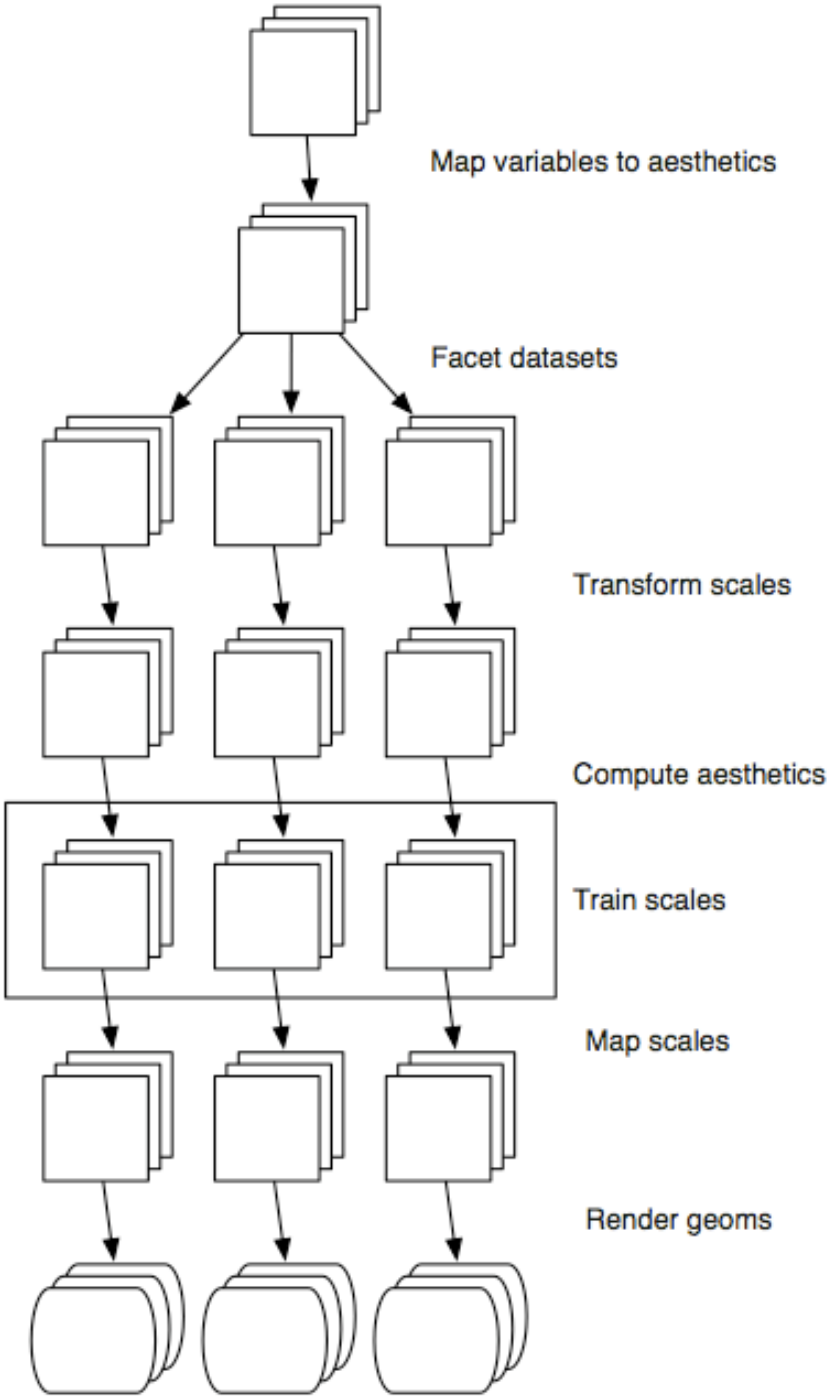

hwy
40
35
30
25
20
15
2
3
4
5
6
7
displ
factor(cyl)
4
5
6
8

qplot(displ, hwy, data=mpg, facets = . ~ year)



qplot(displ, hwy, data=mpg, facets = . ~ year) + geom_smooth()





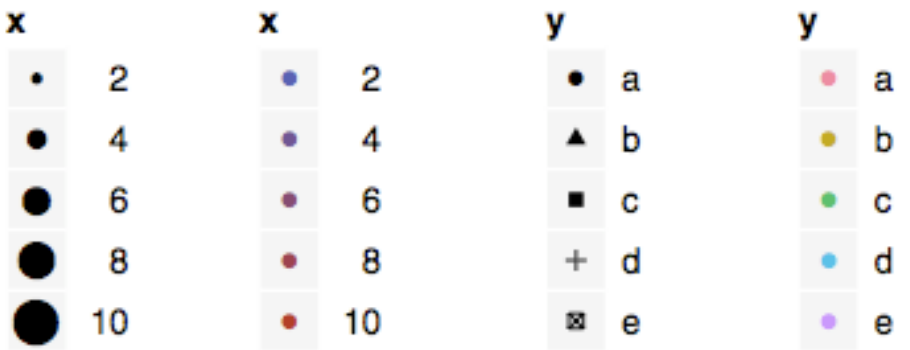
Anatomy of a plot

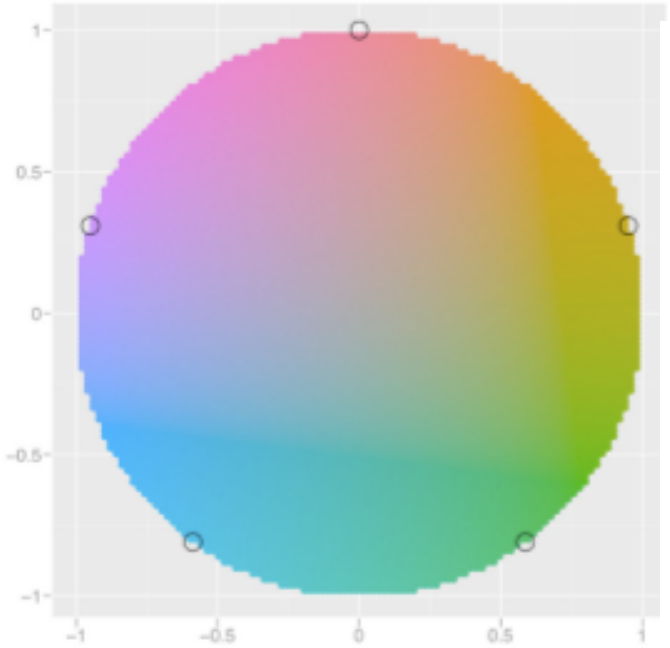
- scale transformations
- stat transformations
  - After scaling so that log/lin make sense
- train scales
  - Superset across different layers
- map data to aesthetics
- render geoms

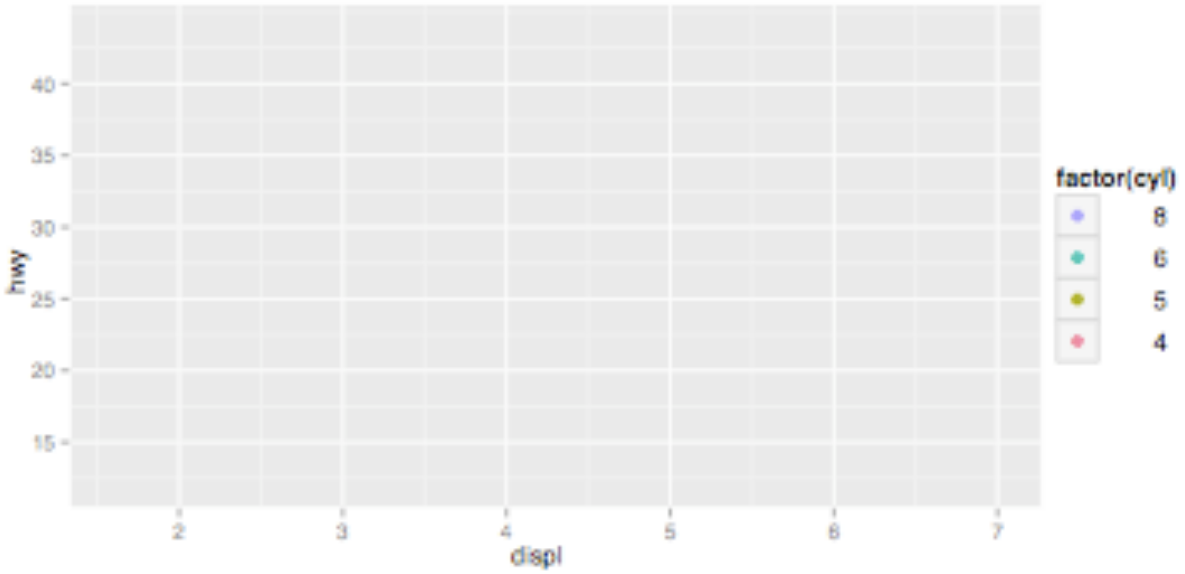
What is a plot (made of)?

- A ***data***set
- Set of ***mapping***s from variables to aesthetics
- At least one layer, each made of
  - A geometric object (***geom***)
  - A statistical transformation (***stat***)
  - A position adjustment
  - Perhaps a dataset and aesthetic mappings
- One ***scale*** per mapping
- A ***coord***inate system
- ***Faceting*** specification

Example of scale
continuous and discrete variables







```
p <- qplot(displ, hwy, data = mpg,
colour = factor(cyl))
```

Things to do with an object

- print() – automatically unless in a loop
- ggsave() – to write out an image
- summary() – summary about the makeup
- save() – to save the object

summary(p)

- data: manufacturer, model, displ, year, cyl, trans, drv, cty, hwy, fl, class [234x11]
- mapping: colour = factor(cyl), x = displ, y = hwy
- scales: colour, x, y
- faceting: facet_grid(. ~ ., FALSE)

- geom_point:
- stat_identity:
- position_identity: (width = NULL, height = NULL)

- > # Save plot object to disk
- > save(p, file = "plot.rdata")
- > # Load from disk
- > load("plot.rdata")
- > # Save png to disk
- > ggsave("plot.png", width = 5, height = 5)

Adding layers

- p <- ggplot(diamonds, aes(carat, price, colour = cut))
- p <- p + layer(geom = "point") #stat/pos defaults
- layer(geom, geom_params, stat, stat_params, data, mapping, position)
- p <- ggplot(diamonds, aes(x = carat))
- p <- p + layer(
  - geom = "bar",
  - geom_params = list(fill = "steelblue"),
  - stat = "bin",
  - stat_params = list(binwidth = 2))
- p

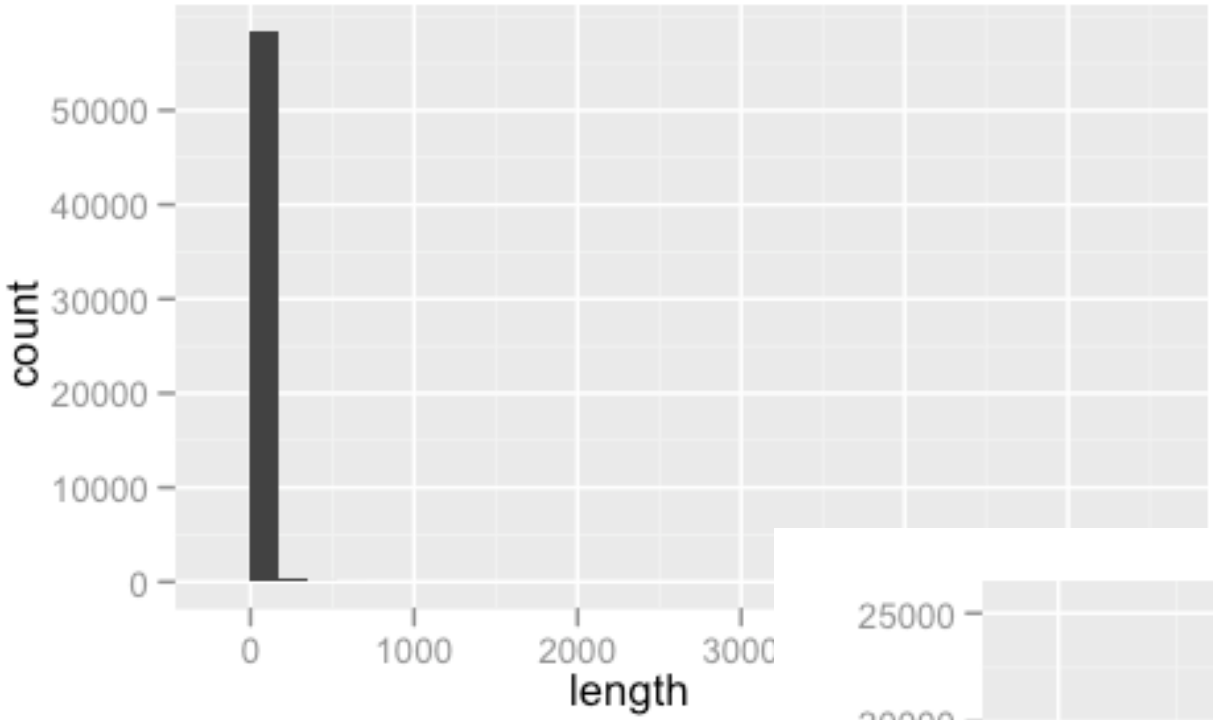
Every geom associated with a (default) stat

- geom_histogram(binwidth = 2, fill = "steelblue")

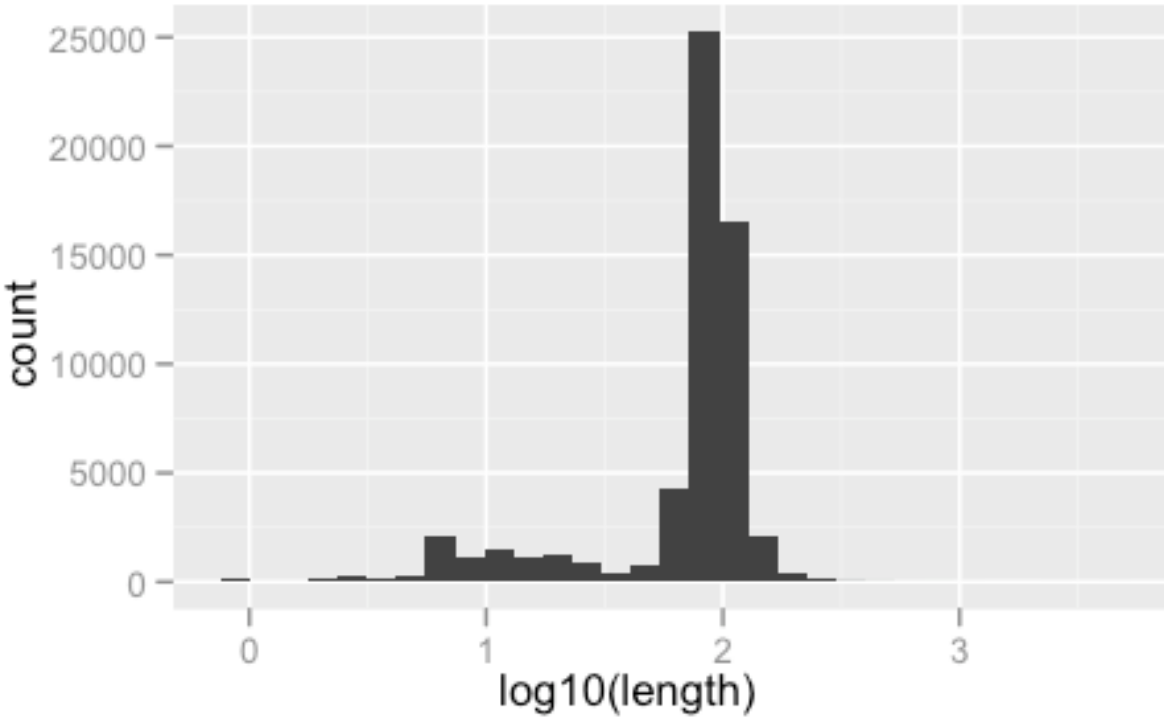
And every stat with a (default) geom

stat: bin

geom: bar

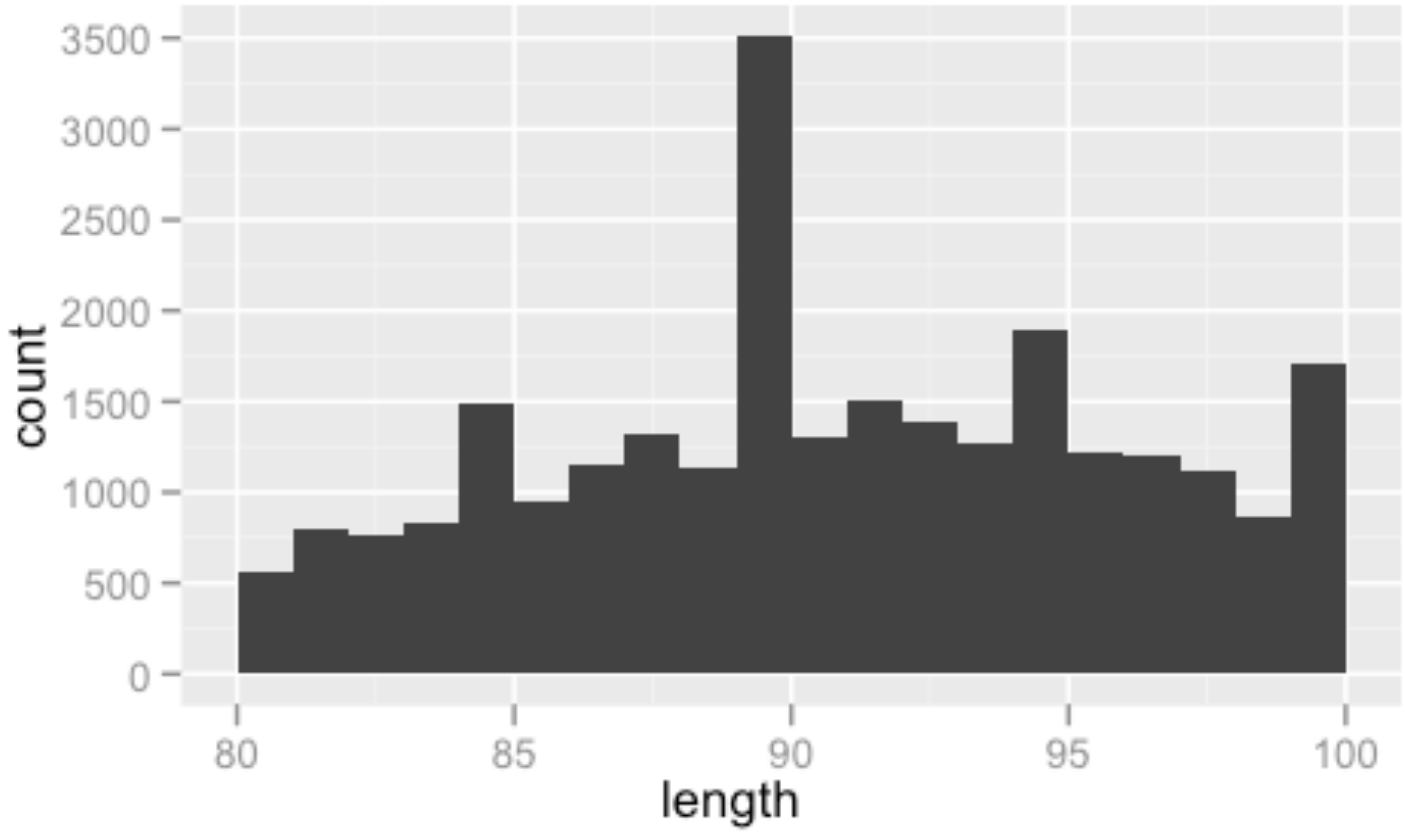


28819 movies from IMDB

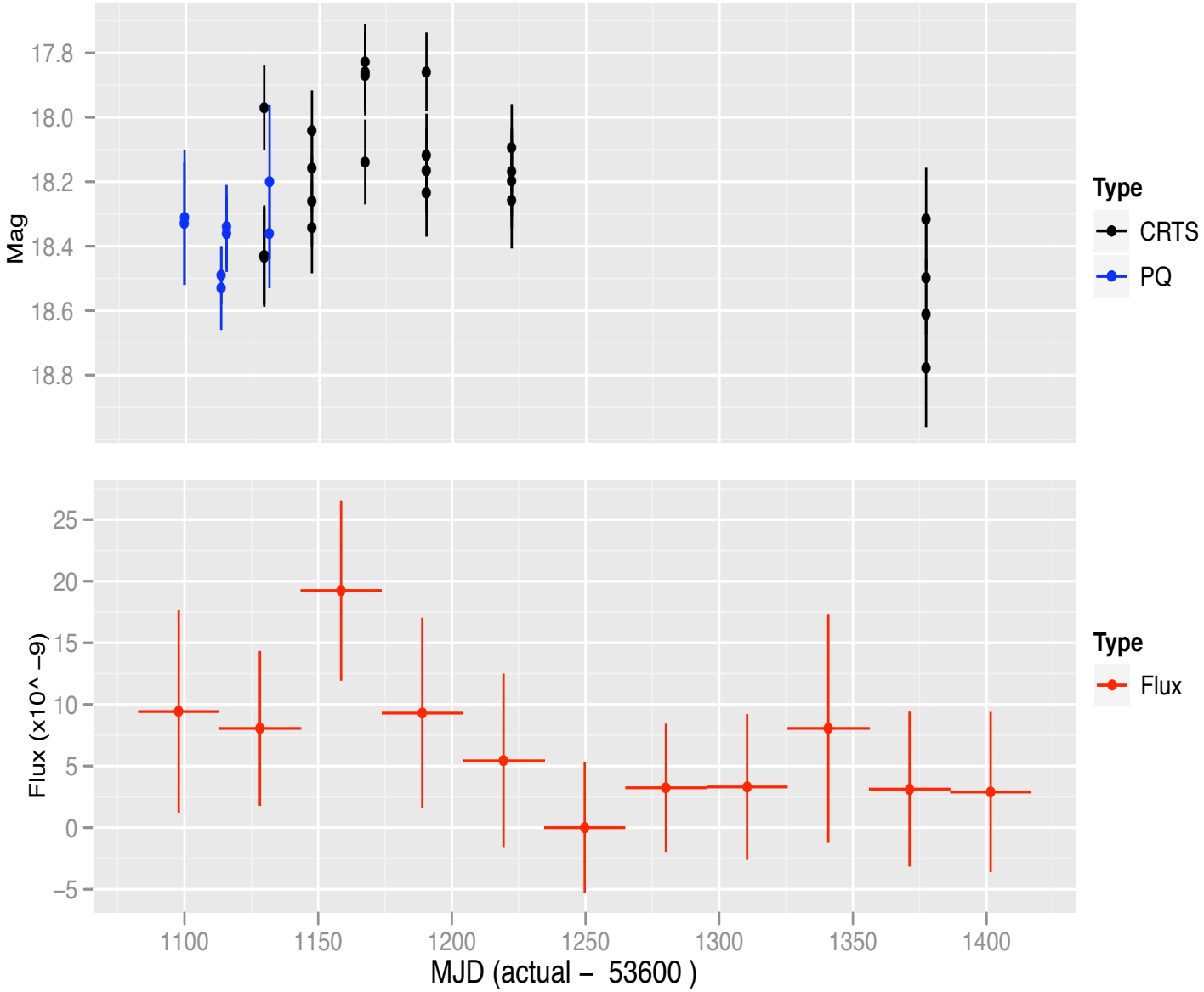


24 variables including Length of movie.
Linear and log plots

qplot(length, data=movies, geom="histogram", binwidth=1, xlim=c(80,100))



Blazar lightcurve example



Judy Mou

Some more links …

- http://vostat.org
- R GUI (R commander http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/)
- http://scc.stat.ucla.edu/mini-courses
- http://cran.r-project.org
- http://www.r-project.org
- http://en.wikipedia.org/wiki/R_(programming_language)
- GGobi (http://www.ggobi.org/) /rggobi

Homework

- Apply various graphing techniques learnt to the astronomy dataset
- http://avyakta.caltech.edu:8080/datasets/dataset2.Rframe
- Flux ratios aka Colors (u-g, g-r, r-I, i-z) and spectroscopic class

- num umg gmr rmi imz specClass
- 1 1.80 0.35 0.42 0.29 3
- 2 1.22 0.01 -0.05 -0.06 1

colors and classification

- d2 = read.table(“dataset2.Rframe”,header=TRUE)
- objects()
- names(d2)
- umg
- d2$umg
- attach(d2)
- umg
- plot(umg,gmr)
- pairs(d2)

Use ggplot2 methods instead of the above