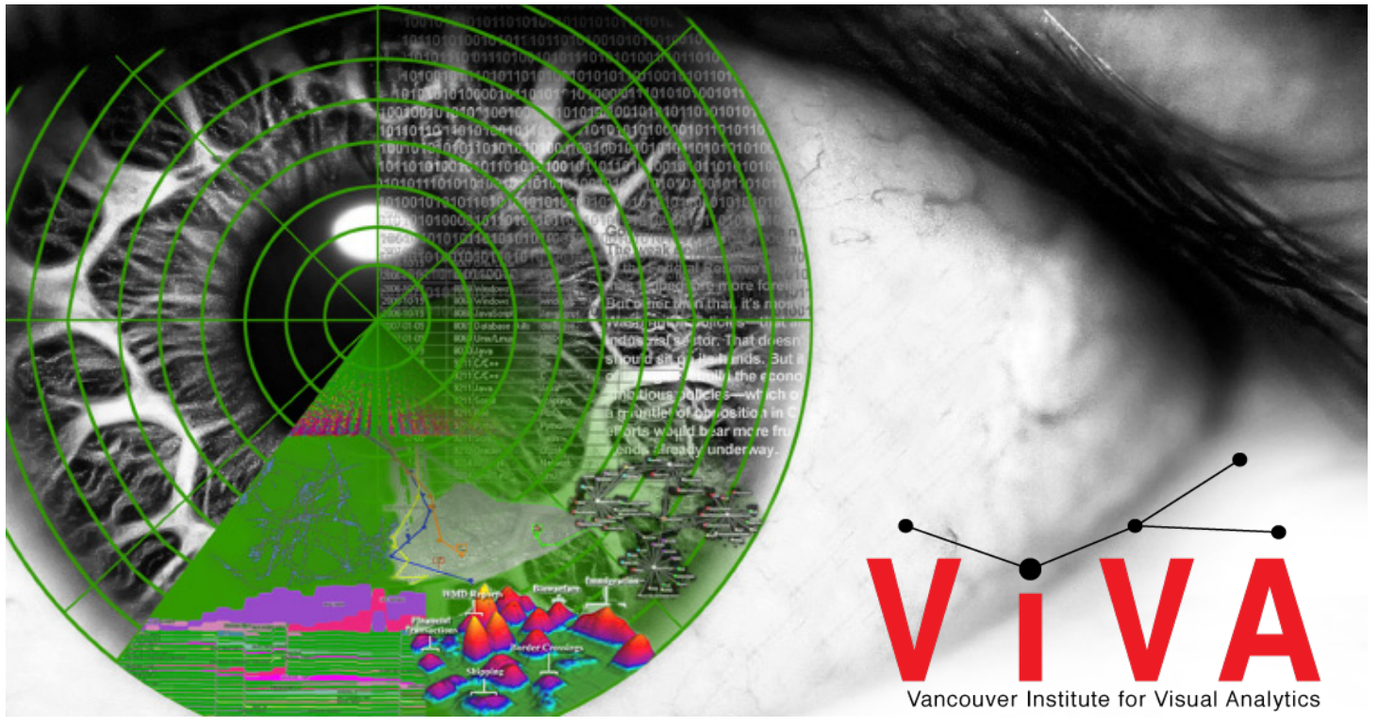
STAT 545A
Class meeting #5
Wednesday, September 19, 2012

Dr. Jennifer (Jenny) Bryan

Department of Statistics and Michael Smith Laboratories



The **Vancouver Institute for Visual Analytics** invites all UBC students and faculty to attend the Open Doors event of the Andrew Wade Visual Analytics Challenge program on Monday, October 1st, 12:00pm, at the Irving K. Barber Dodson Room.

The Vancouver Institute for Visual Analytics (VIVA) is a joint SFU/UBC initiative to promote Visual Analytics ("the science of analytical reasoning facilitated by interactive visual interfaces").

VA is a multidisciplinary field intended to help people interactively explore and synthesize information in order to derive insights from massive, dynamic, and often ambiguous and conflicting data.

VA draws on research from areas such as information visualization, human-computer interaction, machine learning, statistics, decision making and problem solving, communication, and the cognitive and social sciences.

VA can prepare you for internships leading to high-demand positions of data scientists and analysts.

VIVA offers students an opportunity to participate in VA research projects and work with cutting edge VA tools, using real-world data, in collaboration with domain subject matter experts.

You will learn:
- how you can get involved in VA through the Andrew Wade Visual Analytics Challenge program
- about the new Graduate Certificate in Visual Analytics at SFU

Pizza and soft drinks will be provided.

Registration is free at:http://awva.eventbrite.com (limit 60)
Location: University of British Columbia – Vancouver Campus, Dodson Room, Irving K. Barber Learning Center
Date: Monday, October 1st, 2012
Time: 12:00 – 13:00

For more information: challenge@viva-viva.ca

SFU SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

UBC a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

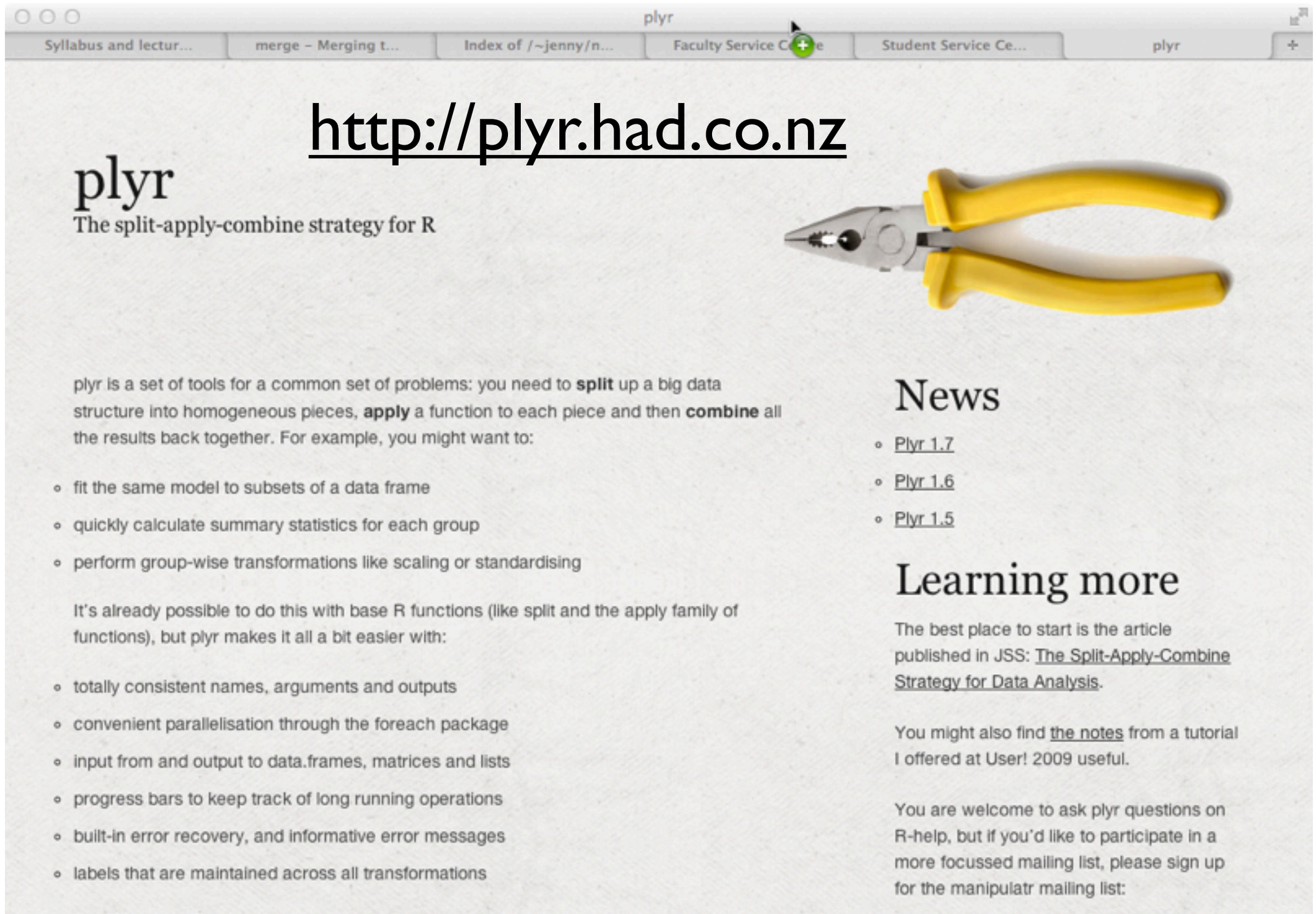
Review of last class

How to isolate bits of R objects for inspection, modification, graphing, modelling.

Data aggregation: doing something repetitive for various bits of your data. Top-level for loops authored by YOU are rarely necessary or even desirable. Exploit apply, sapply, lapply, tapply, by, etc.

do.call() trick and other strategies for "tidying up" the results returned after data aggregation.

The plyr package may be worth adopting for data aggregation. JB intends to make the switch! Still good to know about the base R functions, though.....



Review of last class

Data presentation strategies:

build up your own confidence and that of your audience with (boring) facts (how many observations? how many variables? overview of missing data and how you address) ... make some figures illustrating this (boring) stuff

give your audience a good sense of the whole dataset, at a high level

do some bulk processing / data reduction, e.g. linear regression of lifeExp on year for each of 142 countries

now transition to highlighting trends and facilitating comparisons (e.g. distribution of life expectancy rates of change by continent)

use analytical results to identify interesting cases, e.g. countries with worst and best life expectancy gains, and revisit raw data ... end with an interesting story ... tie to outside events or knowledge, etc.

Before we go on ...
My advice re: assignment operator in R

```
(jYear <- max(gDat$year))
xyplot(lifeExp ~ gdpPercap, gDat,
       subset = year == jYear)
```

I **strongly** recommend you use '<–' for assignment , instead of '='.

It shows and enforces better discipline. Say exactly what you mean, mean exactly what you say.

Assignment, argument passing, and testing for equality are distinct concepts. Thus, your syntax should be distinct too.

In my head, I read '`x <- rnorm(10)`' as "x gets 10 random normal variates".

I reserve the single equals sign ('=') for providing values for funtion arguments.

The double equals sign ('==') is a comparison operator.

Don't just take my word for it, look at R itself and examples in documentation. Look at Google's R style guide.

```
(jYear <- max(gDat$year))
plot(lifeExp ~ gdpPercap, gDat,
     subset = year == jYear)
```

Today!

exploring the numeric variables:
population
life expectancy
GDP per capita

exploring and checking quantitative
univariate data

Consider observations of one quantitative variable X ... possibly in the presence of one or two categorical variables Y and Z, that take on a small number of values

X might be ... life expectancy in Gapminder

Y, Z might be ... country or continent or year

X might be ... life expectancy in Gapminder

Y, Z might be ... country or continent or year

What would you most like to know about the observed distribution of the X's (ignore Y, Z)?

Now focus on the possible relationship between X and Y, Z. What would you most like to know?

<make a list>

Key foundational concepts

Let's say that random variable X has cumulative distribution function F and density f, i.e.

$F(x) = P(X \leq x),\ F'(x) = f(x)$

Quantile function is the inverse of the CDF F

$F(x_0) = p_0 \Leftrightarrow F^{-1}(p_0) = x_0$

Specific functionals of the distribution are of special interest

$E(X) = \int x\, dF$ "expectation" "the mean" (measure of location)

$F^{-1}(0.5)$ "the median" (robust measure of location)

$V(X) = \mathrm{var}(X) = \sigma^2 = \int (x - E(X))^2\, dF$ "variance" (measure of spread)

σ "standard deviation"

"median absolute deviation" "MAD" (measure of spread)

$\mathrm{IQR} = F^{-1}(0.75) - F^{-1}(0.25)$ "interquartile range" (measure of spread)

Key concepts -- less ‘tidy’

- Unimodal? If not, how many modes? Where?
- Symmetric? If not, what’s the shape? Which tail is long?
- If considering Y, is the distribution of X meaningfully different ... in location, spread, shape, etc. ... for different values of Y?

Summaries computed from observed data are *empirical versions* of those "key" concepts

I.e. the average of a sample is an estimate -- and merely an estimate -- of the true mean

Clear statistical thinkers make a <u>*big distinction*</u> between these concepts, though we often speak casually about it

In this exploratory data analysis class we will be fairly relaxed but don't ever forget these distinctions are real

Numerical summaries, esp. location

```
> summary(gDat$lifeExp)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  23.60   48.20   60.71   59.47   70.85   82.60 

> fivenum(gDat$lifeExp)
[1] 23.5990 48.1850 60.7125 70.8460 82.6030

> mean(gDat$lifeExp)
[1] 59.47444

> median(gDat$lifeExp)
[1] 60.7125
```

Numerical summaries, esp. spread

```
> var(gDat$lifeExp)
[1] 166.8517

> sd(gDat$lifeExp)
[1] 12.91711

> mad(gDat$lifeExp)
[1] 16.10104

> IQR(gDat$lifeExp)
[1] 22.6475
```

Numerical summaries, esp. extremes

```
> min(gDat$lifeExp)
[1] 23.599

> max(gDat$lifeExp)
[1] 82.603

> quantile(gDat$lifeExp, probs = c(0.05, 0.95))
     5%     95%
38.4924 77.4370

> range(gDat$lifeExp)
[1] 23.599 82.603

> which.min(gDat$lifeExp)
[1] 1293

> gDat[which.min(gDat$lifeExp), ]
     country year     pop continent lifeExp gdpPercap
1293  Rwanda 1992 7290203    Africa  23.599  737.0686

> which.max(gDat$lifeExp)
[1] 804

> gDat[which.max(gDat$lifeExp), ]
    country year       pop continent lifeExp gdpPercap
804   Japan 2007 127467972      Asia  82.603  31656.07
```

Data aggregation returns!

Summarizing X for the different levels of Y

```
> with(gDat,
+      tapply(lifeExp, continent, median))
  Africa Americas     Asia   Europe  Oceania
 47.7920  67.0480  61.7915  72.2410  73.6650
```

```
> (foo <- with(gDat,
+              tapply(lifeExp, continent, summary)))
$Africa
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  23.60   42.37   47.79   48.87   54.41   76.44 

$Americas
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  37.58   58.41   67.05   64.66   71.70   80.65 

$Asia
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  28.80   51.43   61.79   60.06   69.51   82.60 

$Europe
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  43.58   69.57   72.24   71.90   75.45   81.76 

$Oceania
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  69.12   71.20   73.66   74.33   77.55   81.24 

> (leByContinent <- do.call(cbind, foo))
        Africa Americas  Asia Europe Oceania
Min.     23.60    37.58 28.80  43.58   69.12
1st Qu.  42.37    58.41 51.43  69.57   71.20
Median   47.79    67.05 61.79  72.24   73.66
Mean     48.87    64.66 60.06  71.90   74.33
3rd Qu.  54.41    71.70 69.51  75.45   77.55
Max.     76.44    80.65 82.60  81.76   81.24
```

Data aggregation returns!

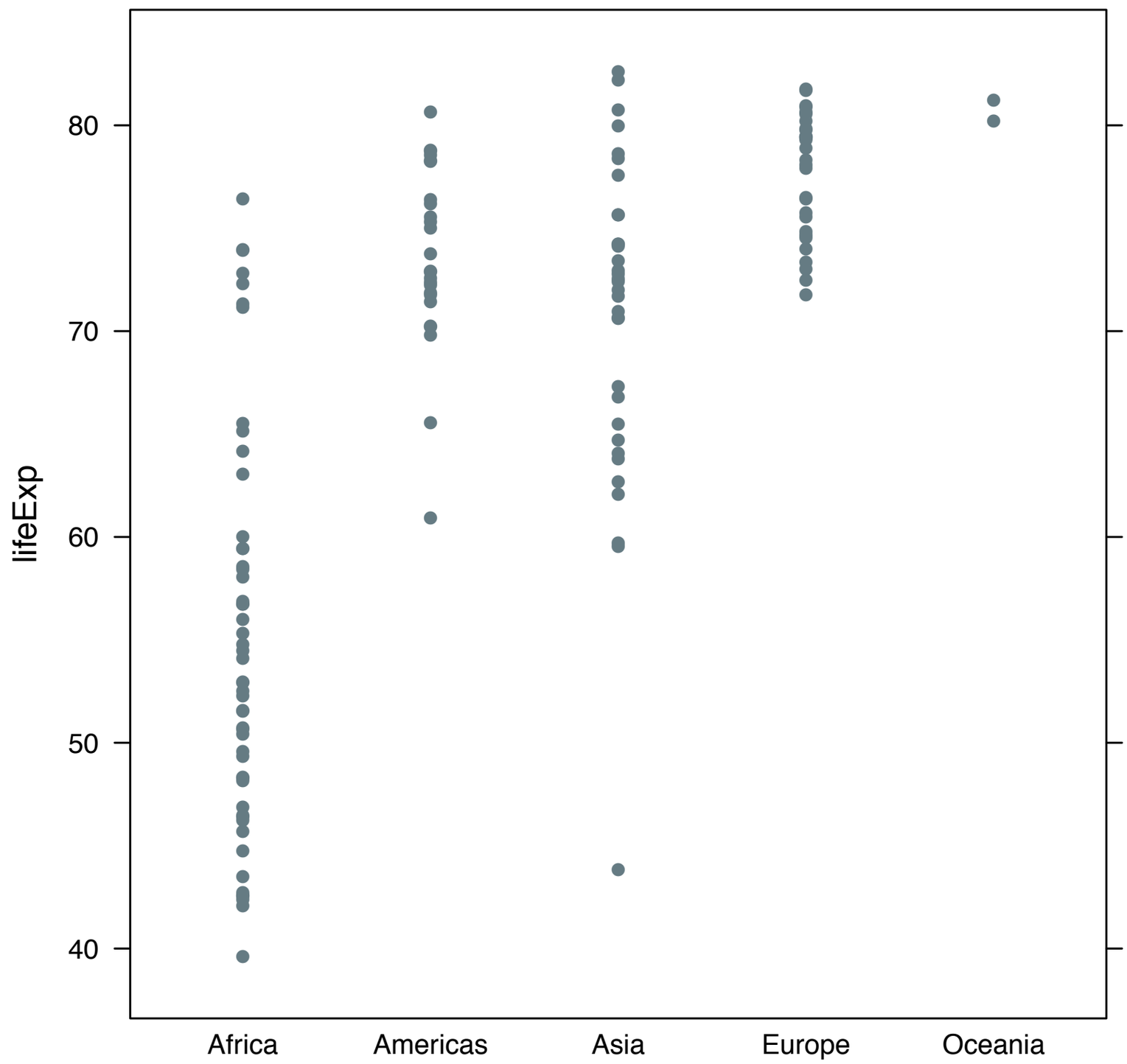
do.call() trick to tidy up the result

Summarizing X for the different levels of (Y, Z)

```
> with(gDat,
+      tapply(lifeExp, list(year, continent), median))
      Africa Americas   Asia  Europe Oceania
1952 38.8330   54.745 44.869 65.9000 69.2550
1957 40.5925   56.074 48.284 67.6500 70.2950
1962 42.6305   58.299 49.325 69.5250 71.0850
1967 44.6985   60.523 53.655 70.6100 71.3100
1972 47.0315   63.441 56.950 70.8850 71.9100
1977 49.2725   66.353 60.765 72.3350 72.8550
1982 50.7560   67.405 63.739 73.4900 74.2900
1987 51.6395   69.498 66.295 74.8150 75.3200
1992 52.4290   69.862 68.690 75.4510 76.9450
1997 52.7590   72.146 70.265 76.1160 78.1900
2002 51.2355   72.047 71.028 77.5365 79.7400
2007 52.9265   72.899 72.396 78.6085 80.7195
```

but who wants to look at tables of numbers all day?

“strip plot”, i.e. a univariate scatter plot




```
stripplot(lifeExp ~ continent, gDat,
          subset = year == 2007)
```

Digression: R's formula syntax

http://cran.r-project.org/doc/manuals/R-intro.html#Formulae-for-statistical-models

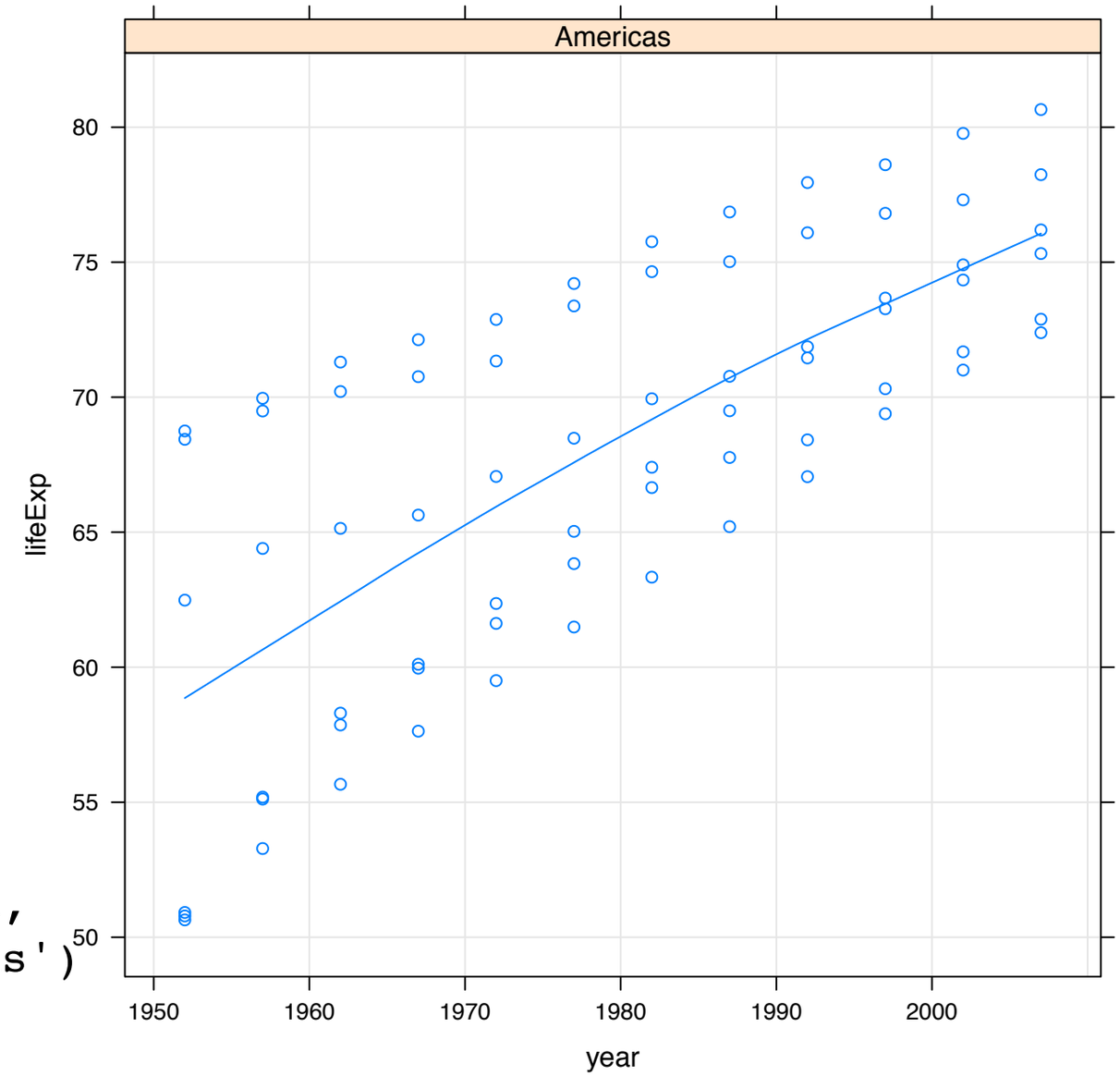
```
y ~ x
```

"y twiddle x"

In modelling functions, says y is response or dependent variable and x is the predictor or covariate or independent variable. More generally, the right-hand side can be much more complicated.

simple linear regression example you've seen before

x and y are quantitative



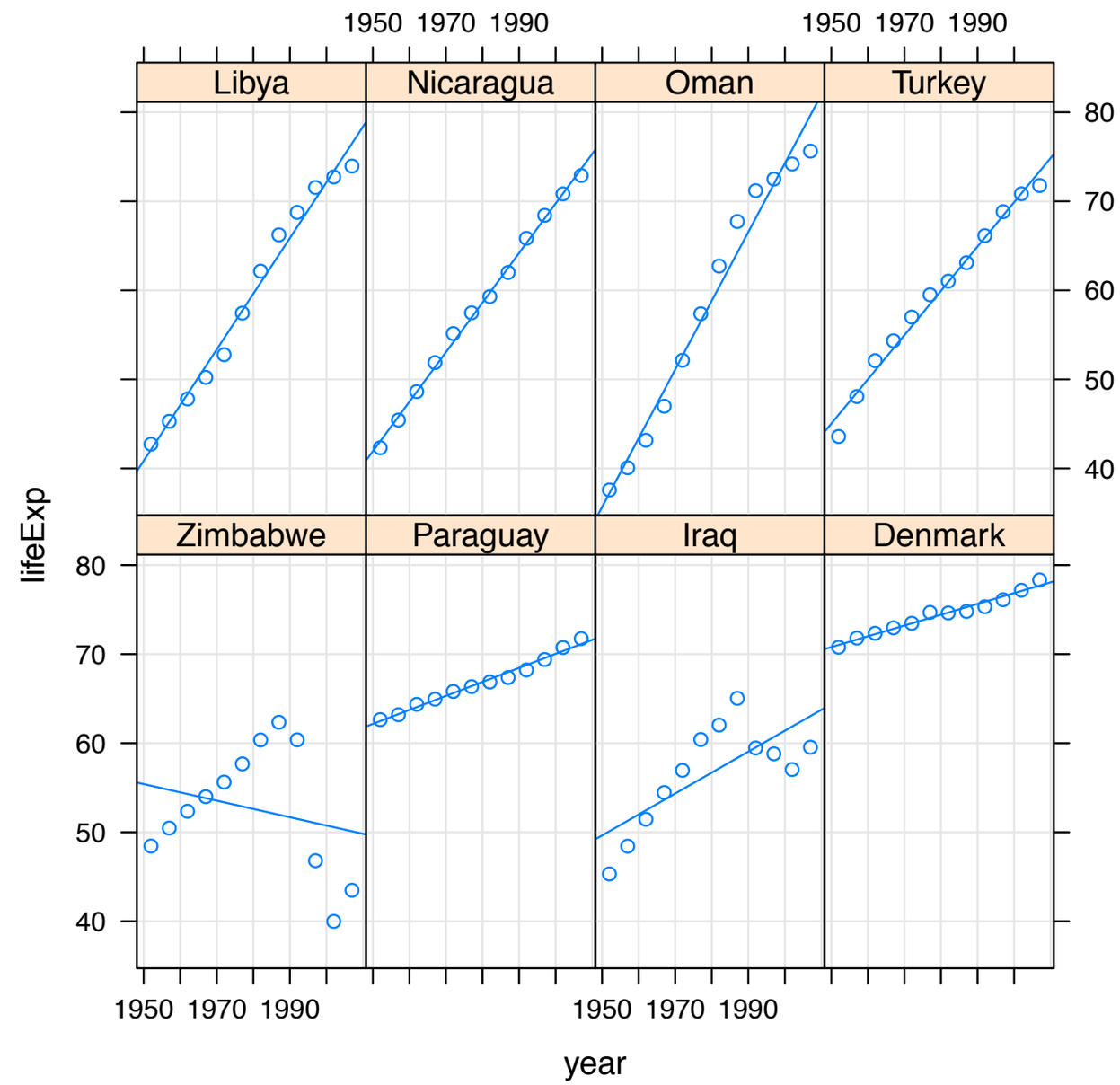

```
> jFit <- lm(lifeExp ~ I(year - 1950), gDat,
+            subset = continent == 'Americas')
> summary(jFit)

<snip, snip>

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    59.03624    1.20834  48.857  < 2e-16 ***
I(year - 1950)  0.30944    0.03535   8.753 7.52e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.178 on 70 degrees of freedom
Multiple R-squared: 0.5225, Adjusted R-squared: 0.5157
F-statistic: 76.61 on 1 and 70 DF,  p-value: 7.524e-13
```

y ~ x

"y twiddle x"

In many plotting functions, esp. lattice, this says to plot y against x.

```
xyplot(lifeExp ~ year | country, zDat,
       layout = c(4,2), type = c('p','g','r'))
```



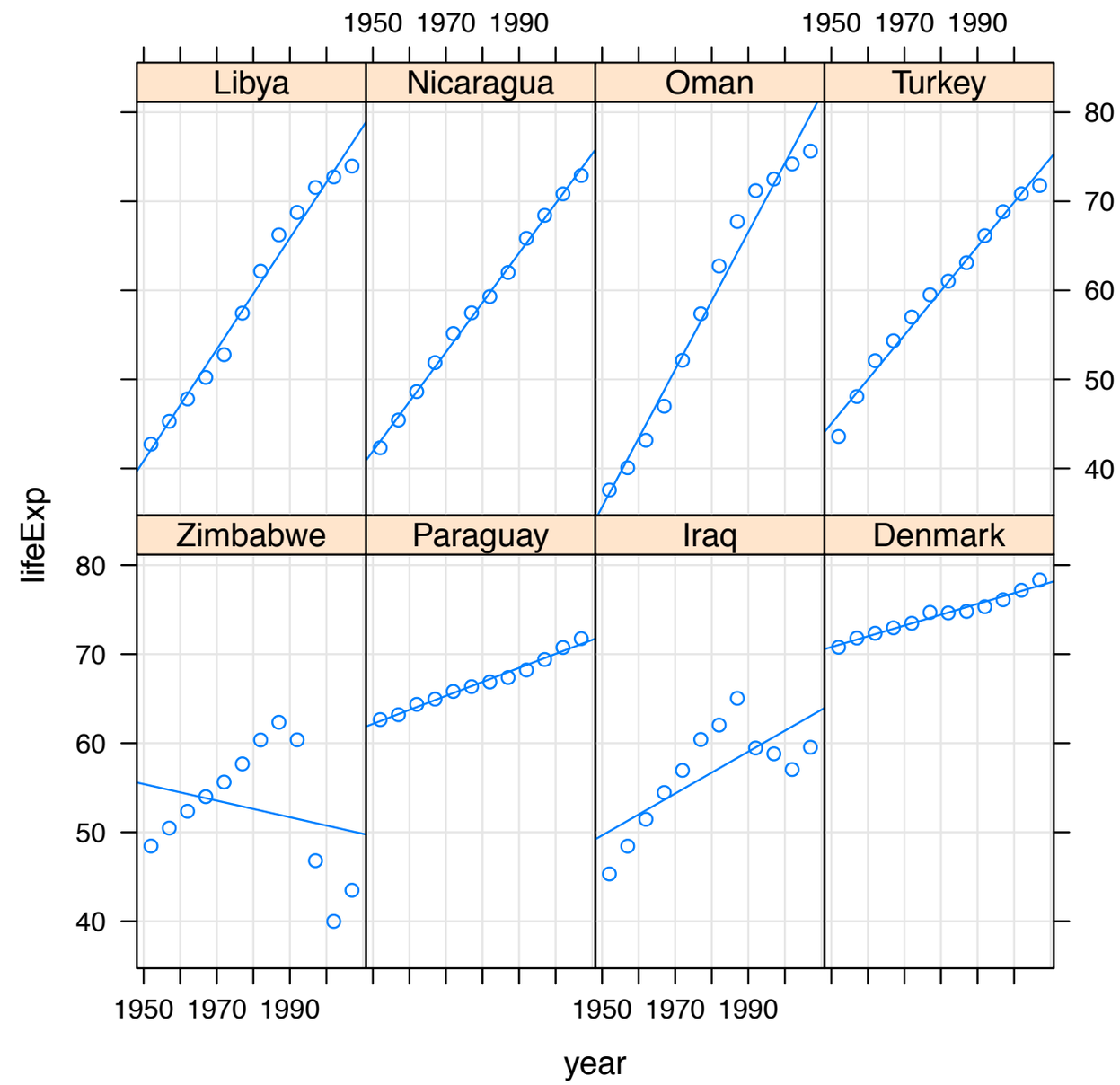
scatterplot example you've seen before

x and y are quantitative

y ~ x | z

In many plotting functions, esp. lattice, this says to plot y against x for every level of z (assumed to be categorical). Evokes conditional probability, "given z", etc.

```
xyplot(lifeExp ~ year | country, zDat,
       layout = c(4,2), type = c('p','g','r'))
```

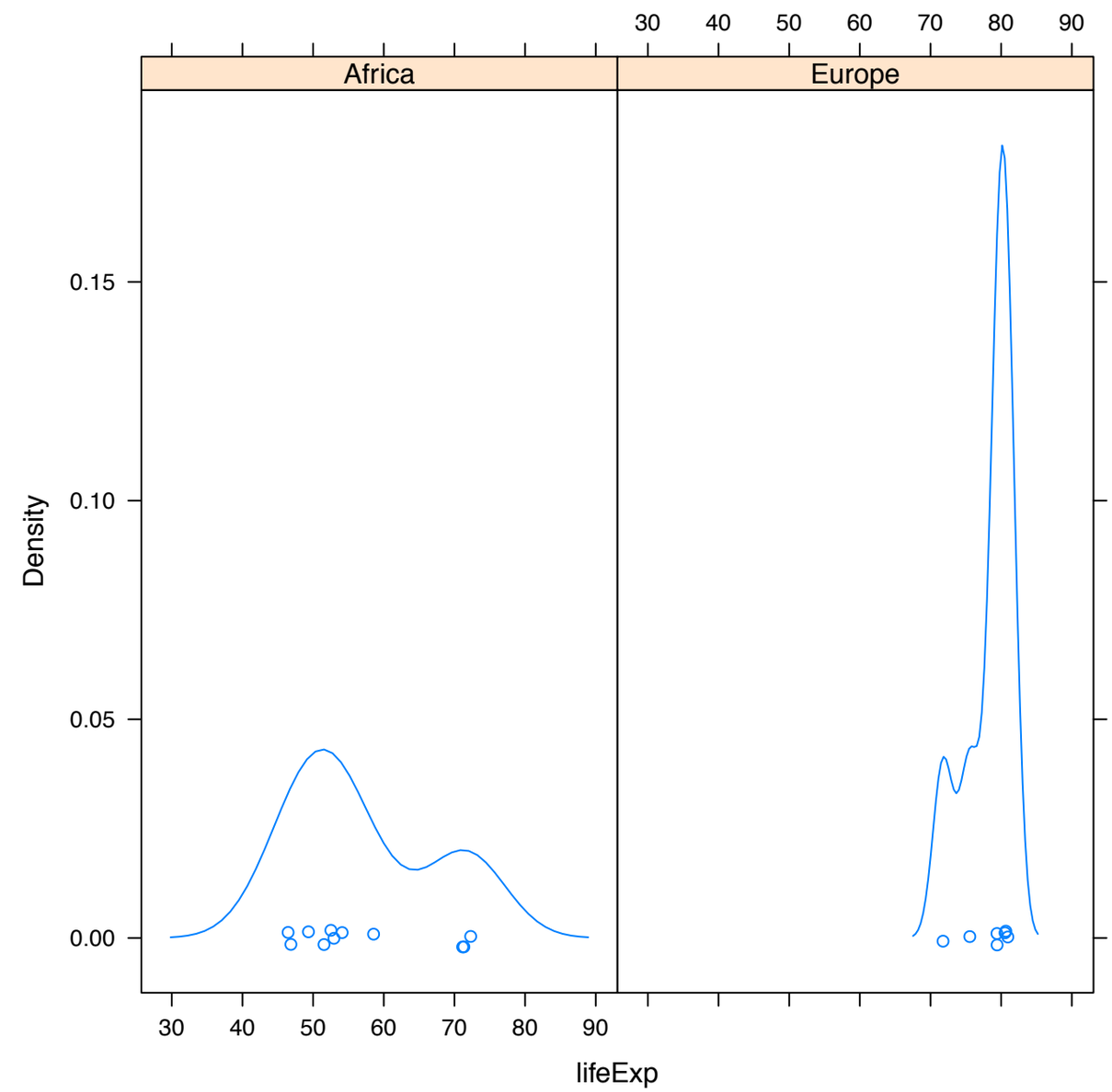


scatterplot example you've seen before
x and y are quantitative
z is categorical

y ~ x

two-groups testing example you've seen before

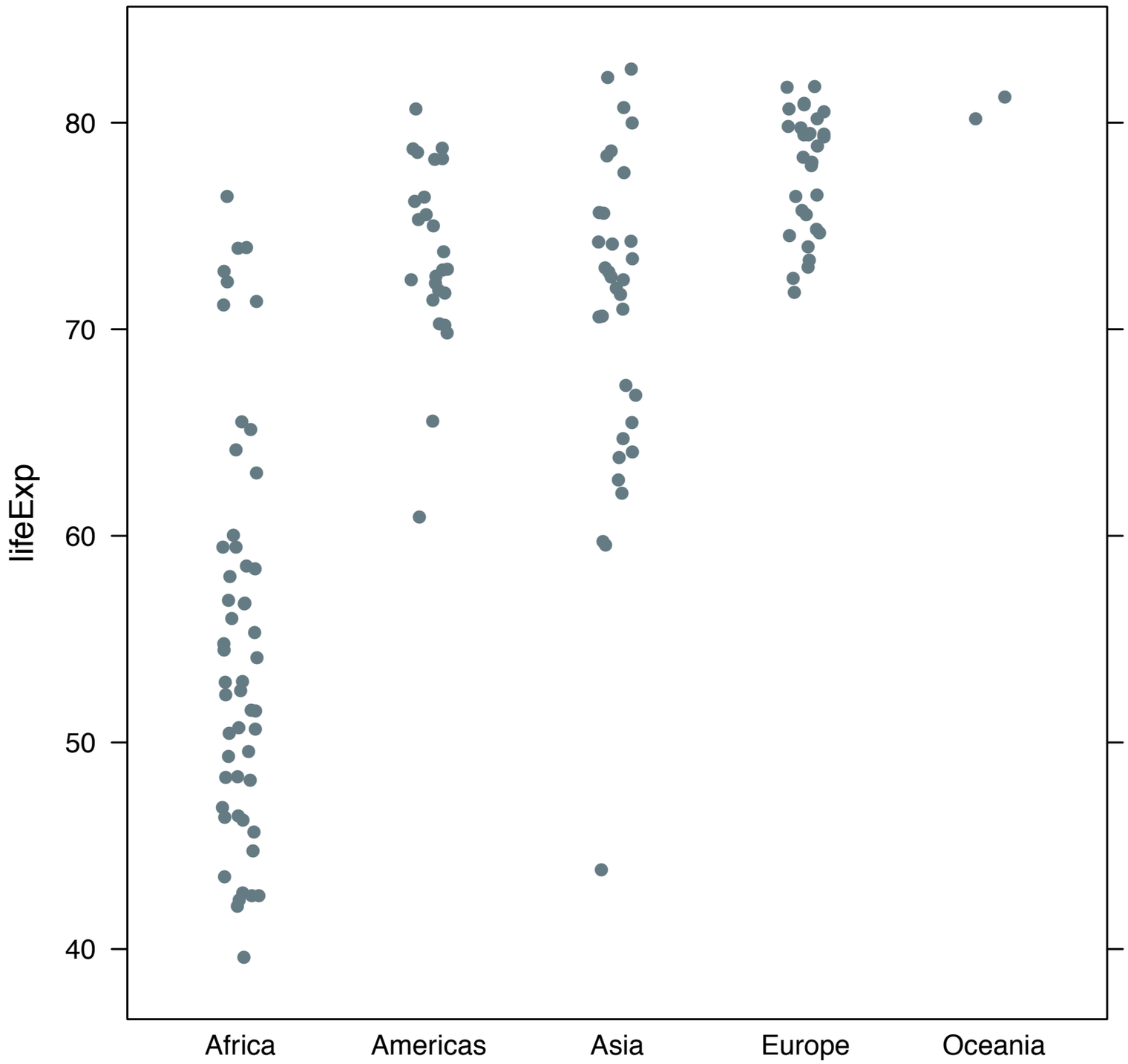
y is quantitative and x is the binary variable that specifies the two groups




```
> t.test(lifeExp ~ continent, tinyDat)

    Welch Two Sample t-test

data:  lifeExp by continent
t = -6.5267, df = 13.291, p-value = 1.727e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -28.35922 -14.27766
sample estimates:
mean in group Africa mean in group Europe
            57.01227             78.33071
```

watch my formulas in the following graphing examples to see more ways to use the formula interface

end digression

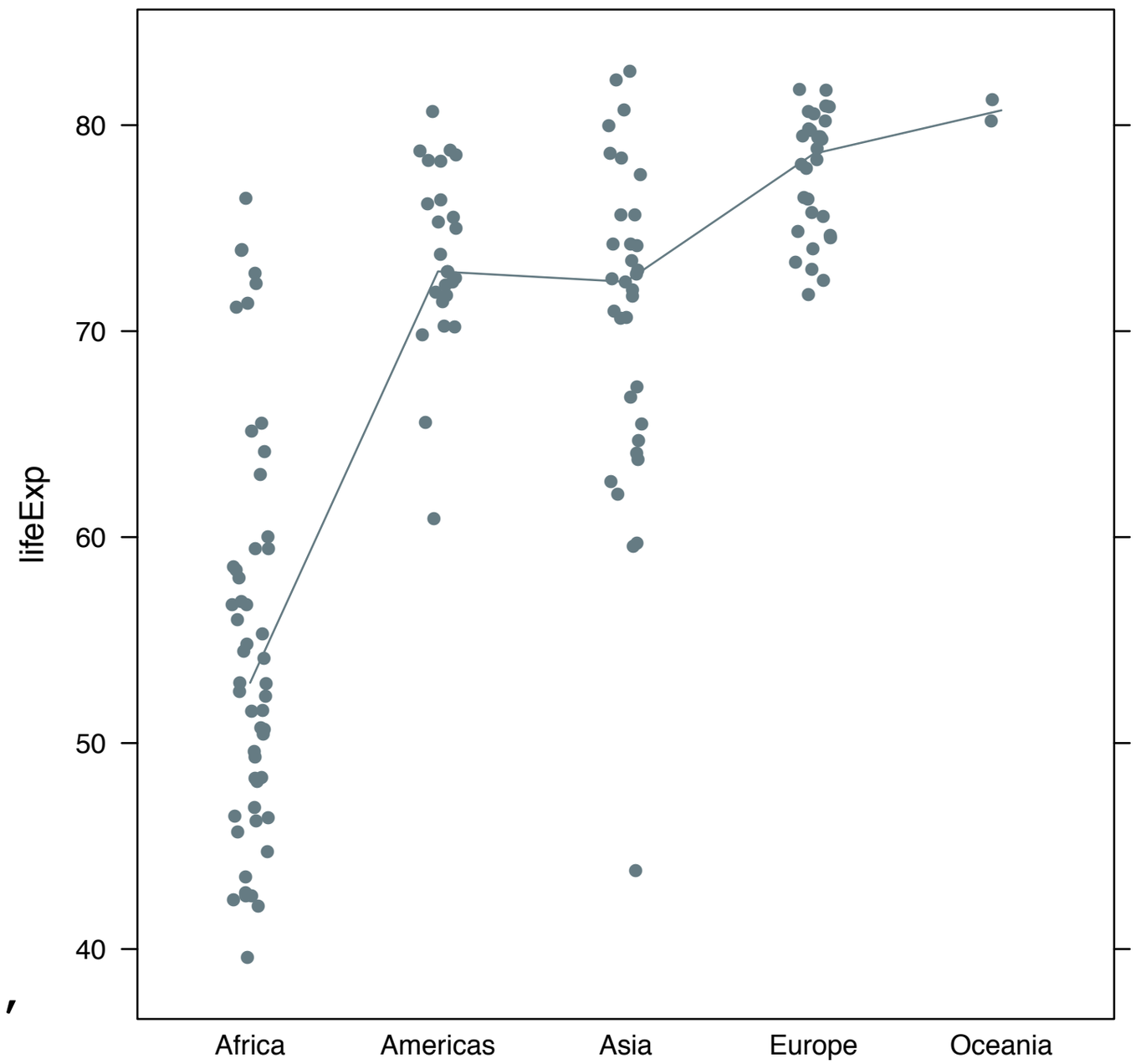



```
stripplot(lifeExp ~ continent, gDat,
          subset = year == 2007,
          jitter.data = TRUE)
```

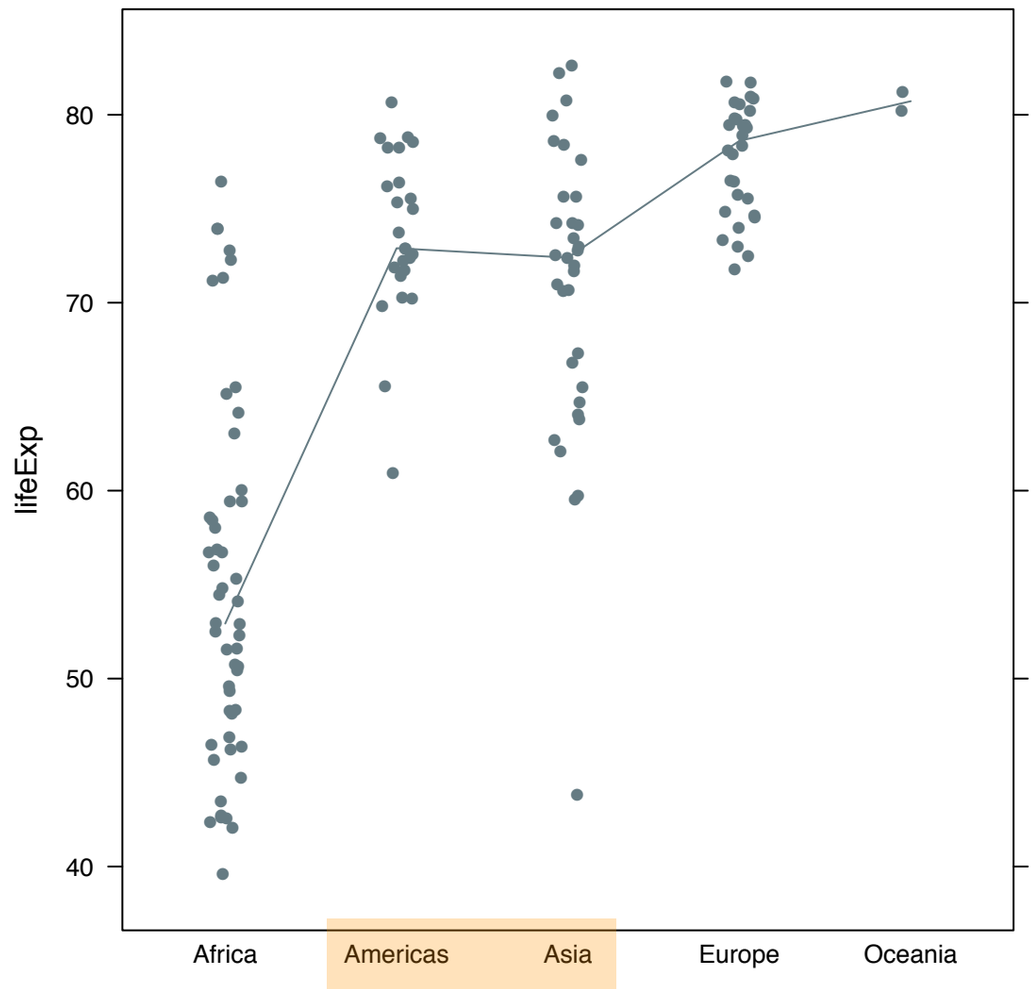
jitter -- adding a bit of Gaussian noise -- is helpful for preventing overplotting in small datasets

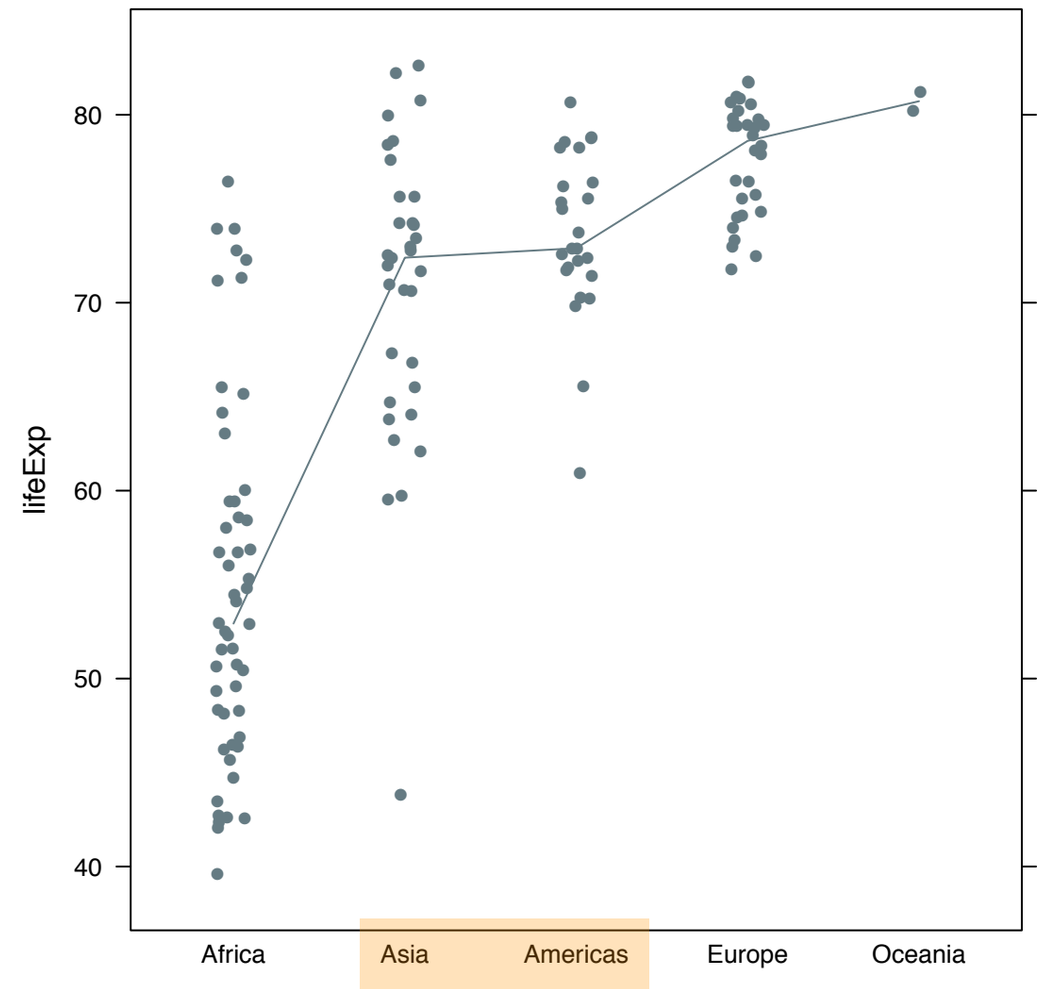
```
> with(gDat,
+      tapply(lifeExp, list(year, continent), median))
      Africa Americas   Asia  Europe Oceania
1952 38.8330   54.745 44.869 65.9000 69.2550
<snip, snip>
2007 52.9265   72.899 72.396 78.6085 80.7195
```

many lattice functions let you request many embellishments via the "type" argument; more info later;
here I add a line connecting the continent-specific medians




```
stripplot(lifeExp ~ continent, gDat,
          subset = year == 2007,
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```





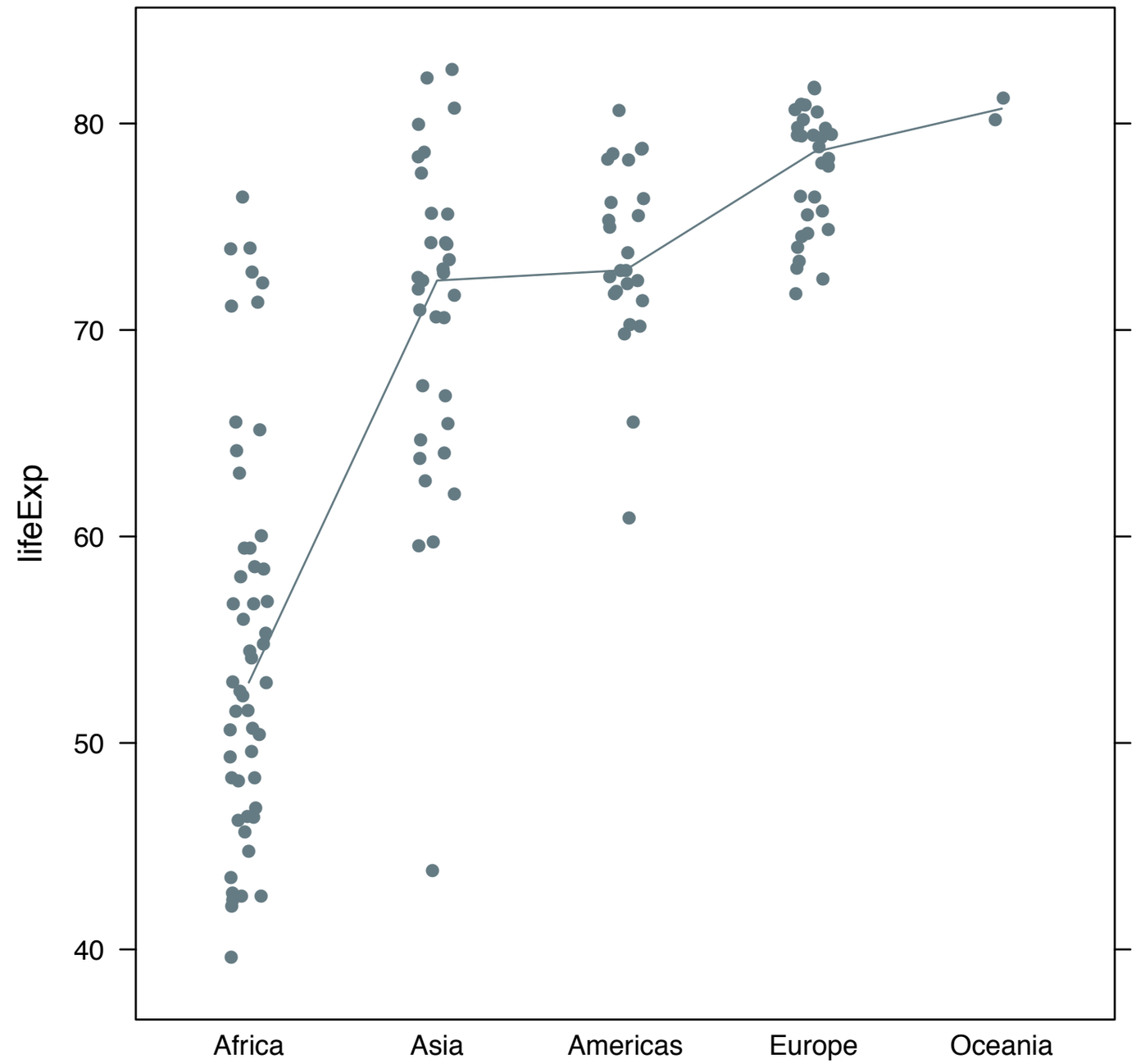

```
stripplot(lifeExp ~ continent, gDat,
          subset = year == 2007,
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```

```
stripplot(lifeExp ~ reorder(continent, lifeExp), gDat,
          subset = year == 2007,
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```

reorder() helps reorder factor levels in terms of a summary measure on a quantitative variable; see, e.g. Sarkar 10.6 "Ordering levels of categorical variables" or Case study 2 in this talk Sarkar gave at UseR! 2007

here we reorder on the fly with reorder()

sometimes we actually change the factor levels order in the underlying data.frame, e.g. gDat

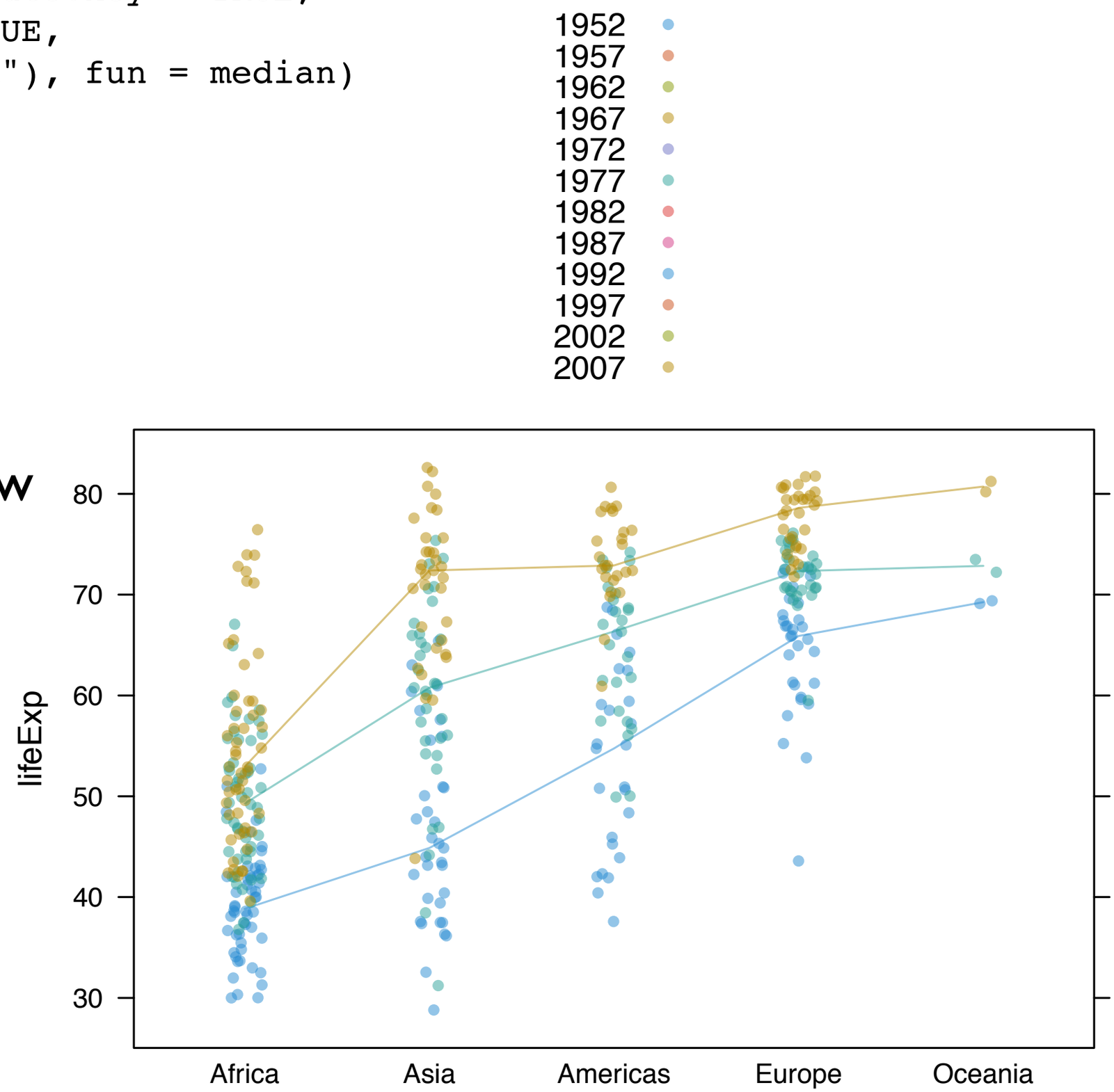



```
stripplot(lifeExp ~ reorder(continent, lifeExp), gDat,
          subset = year == 2007,
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```

part of the proper care and feeding of factors!

```
stripplot(lifeExp ~ reorder(continent, lifeExp),
          gDat, subset = year %in% c(1952, 1977, 2007),
          groups = year, auto.key = TRUE,
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```

like many modelling functions, most lattice functions accept a subset argument; here we narrow to just 3 years (early, middle, and late)

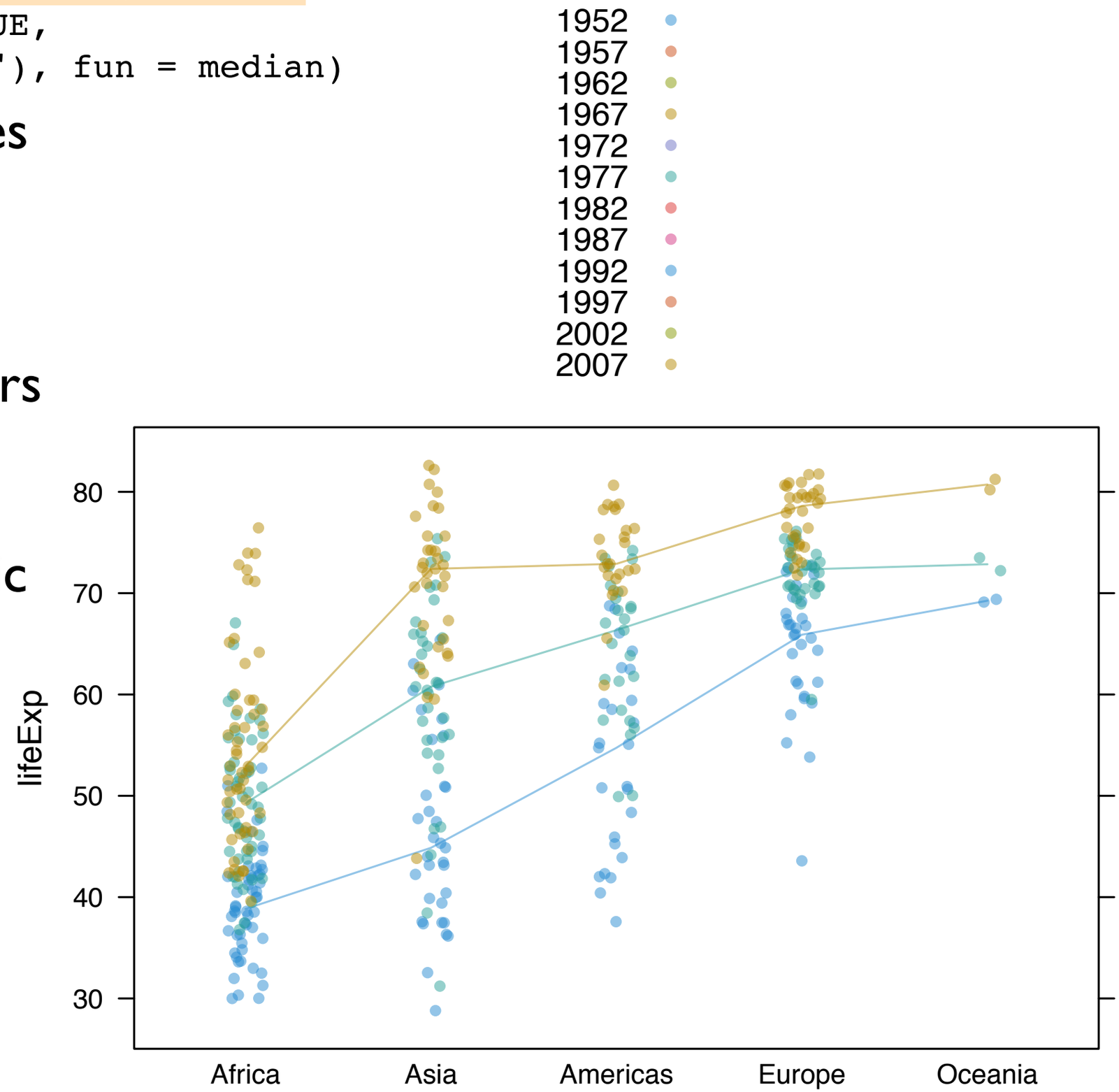


```
stripplot(lifeExp ~ reorder(continent, lifeExp),
          gDat, subset = year %in% c(1952, 1977, 2007),
          groups = year, auto.key = TRUE,
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```

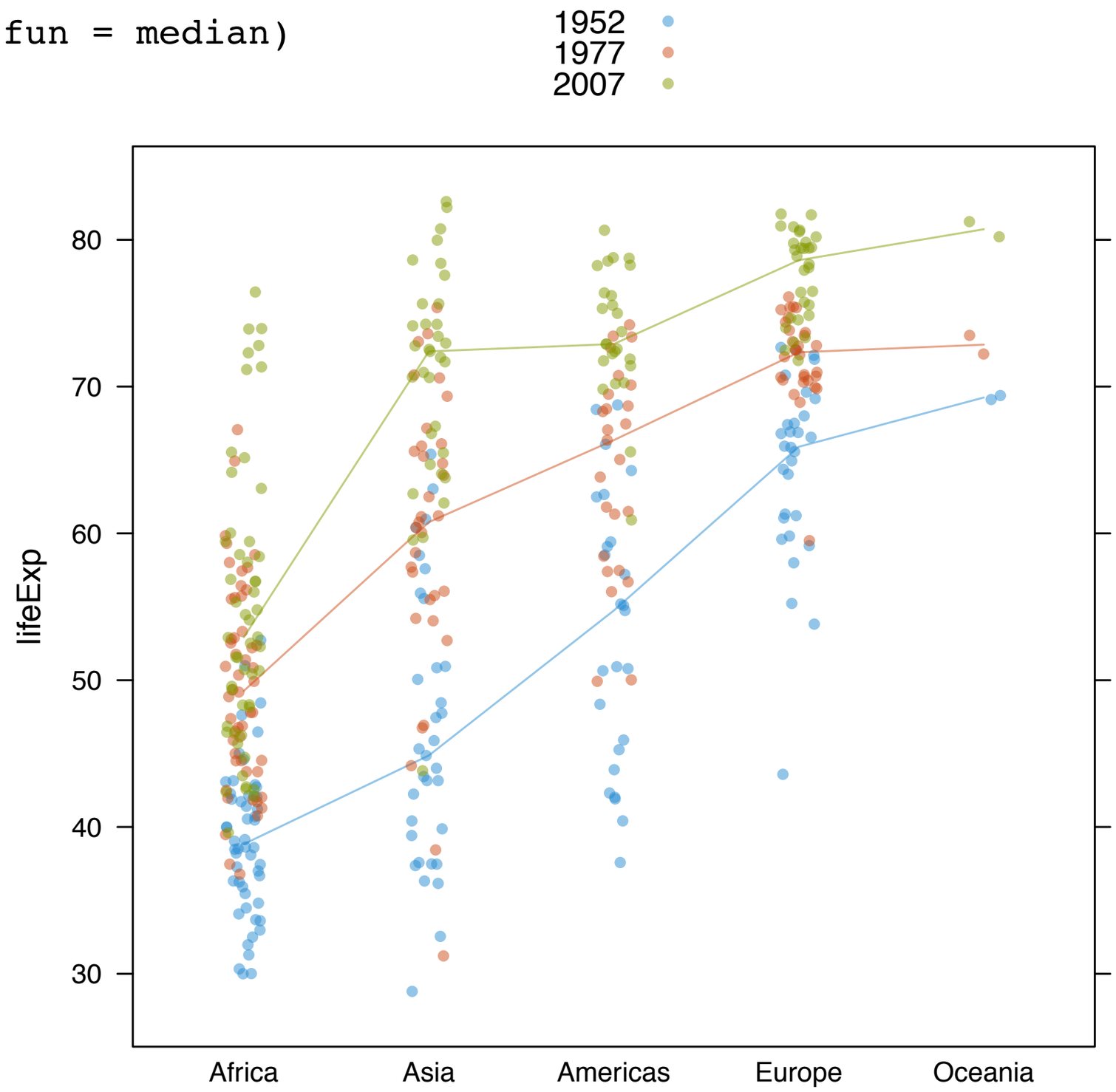
‘groups’ argument specifies a factor variable to distinguish in the plot via superposition, i.e. to highlight via different colors or symbols or line types

auto.key adds an automatic key

but you can see some problems, no?



```
stripplot(lifeExp ~ reorder(continent, lifeExp),
          subset(gDat, subset = year %in% c(1952, 1977, 2007)),
          groups = year, auto.key = TRUE,
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```



much better!

unused factor levels in the factor specified via groups can cause problems

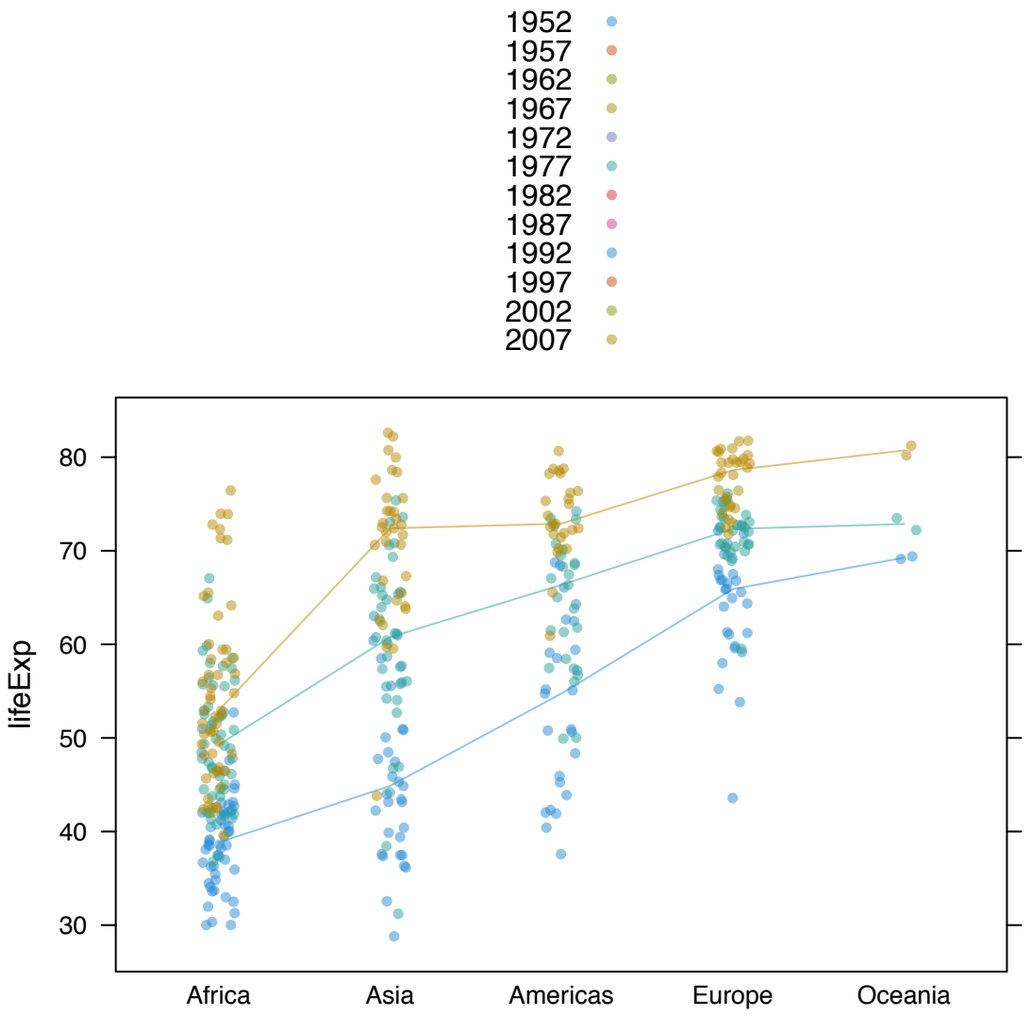
better to subset the data prior to enacting the graphing command

here I've been clever and used subset() in gDat

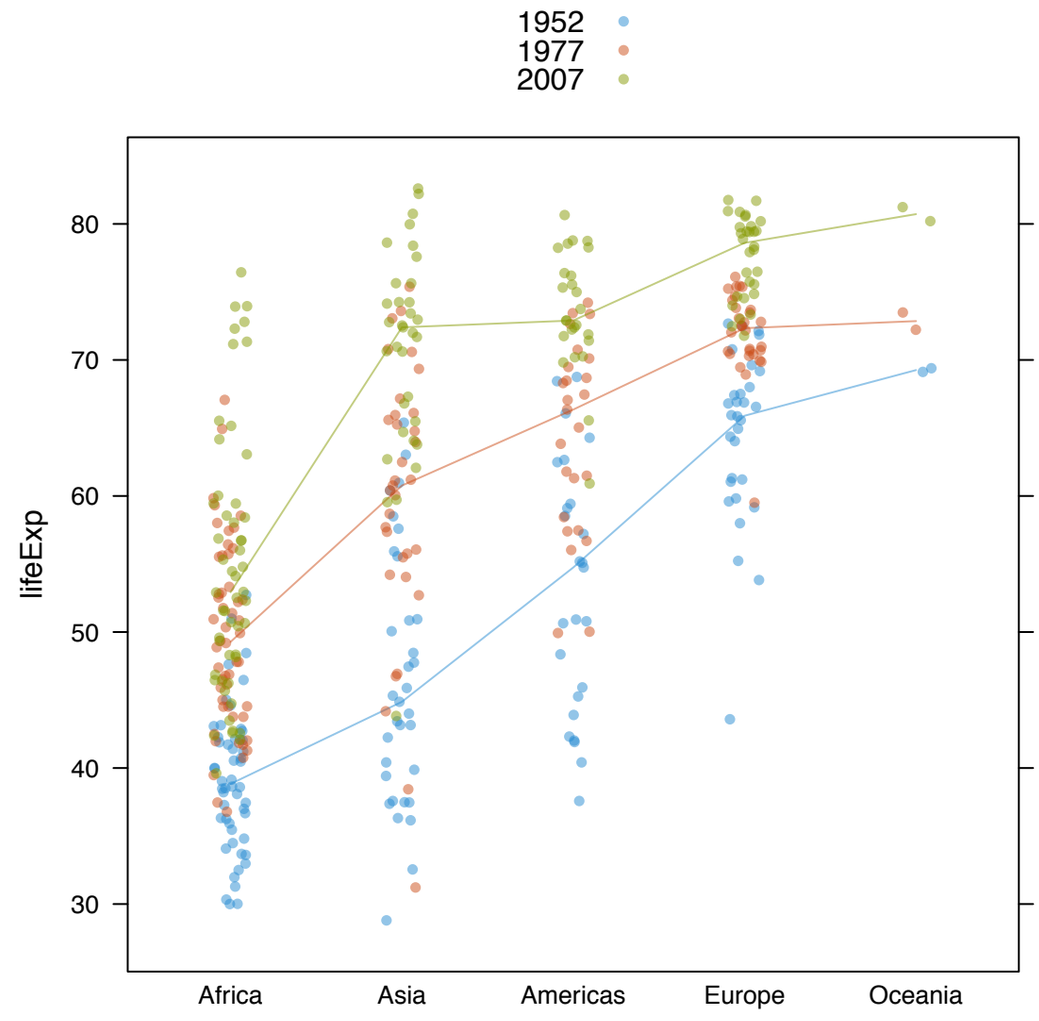
See Sarkar 9.2.5 "Dropping unused levels from groups", Sarkar 10.4.1 "Dropping of factor levels"

subtle difference in stripplot() call; big difference in result

```
stripplot(lifeExp ~ reorder(continent, lifeExp),
          gDat, subset = year %in% c(1952, 1977, 2007),
          groups = year, auto.key = TRUE,
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```

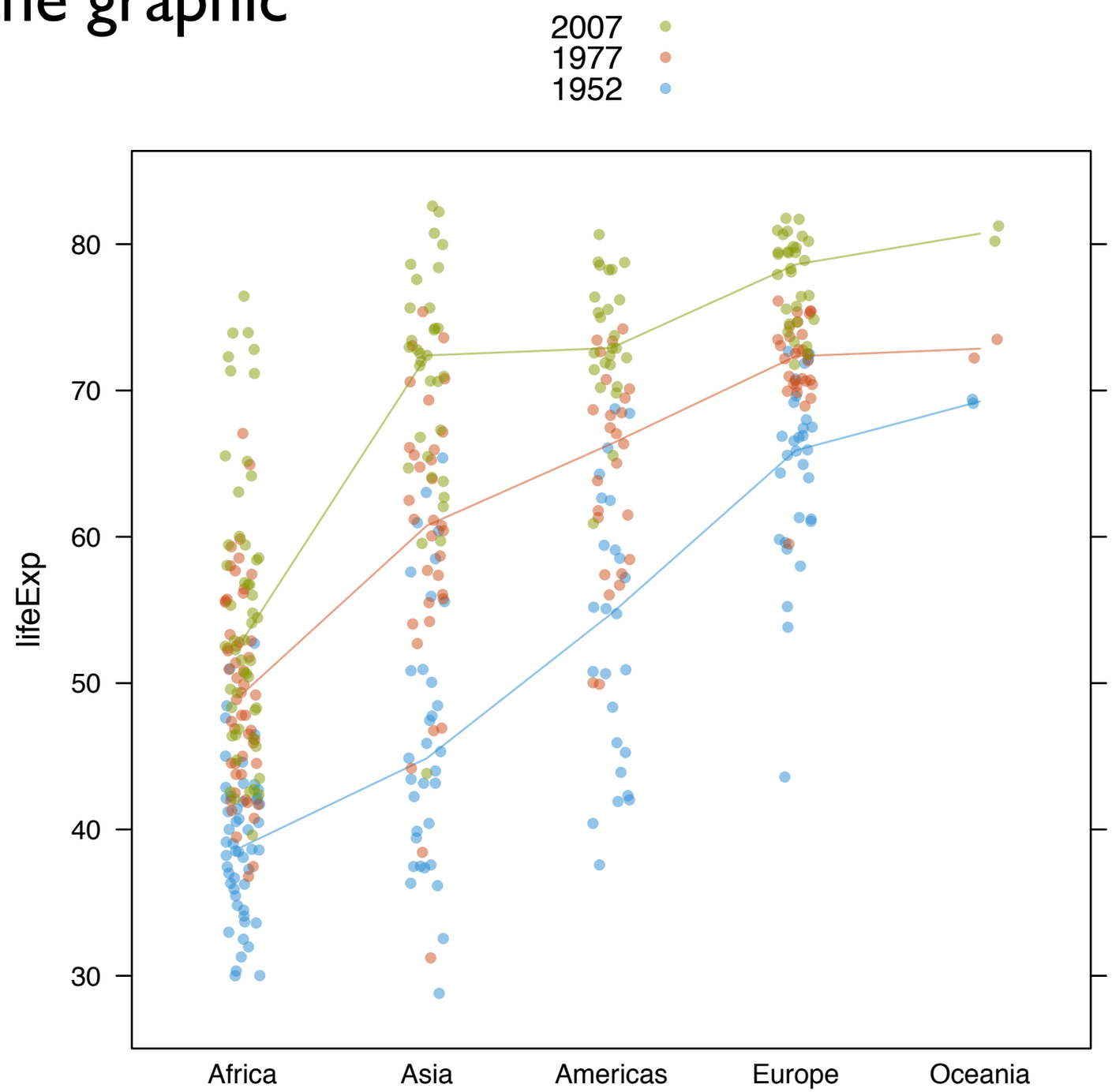



```
stripplot(lifeExp ~ reorder(continent, lifeExp),
          subset(gDat, subset = year %in% c(1952, 1977, 2007)),
          groups = year, auto.key = TRUE,
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```



gilding the lily:
make the order of your key correspond to what the viewer confronts in the graphic

```
stripplot(lifeExp ~ reorder(continent, lifeExp),
          subset(gDat, subset = year %in% c(1952, 1977, 2007)),
          ## reversing rows in key makes it easier to read
          groups = year, auto.key = list(reverse.rows = TRUE),
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```



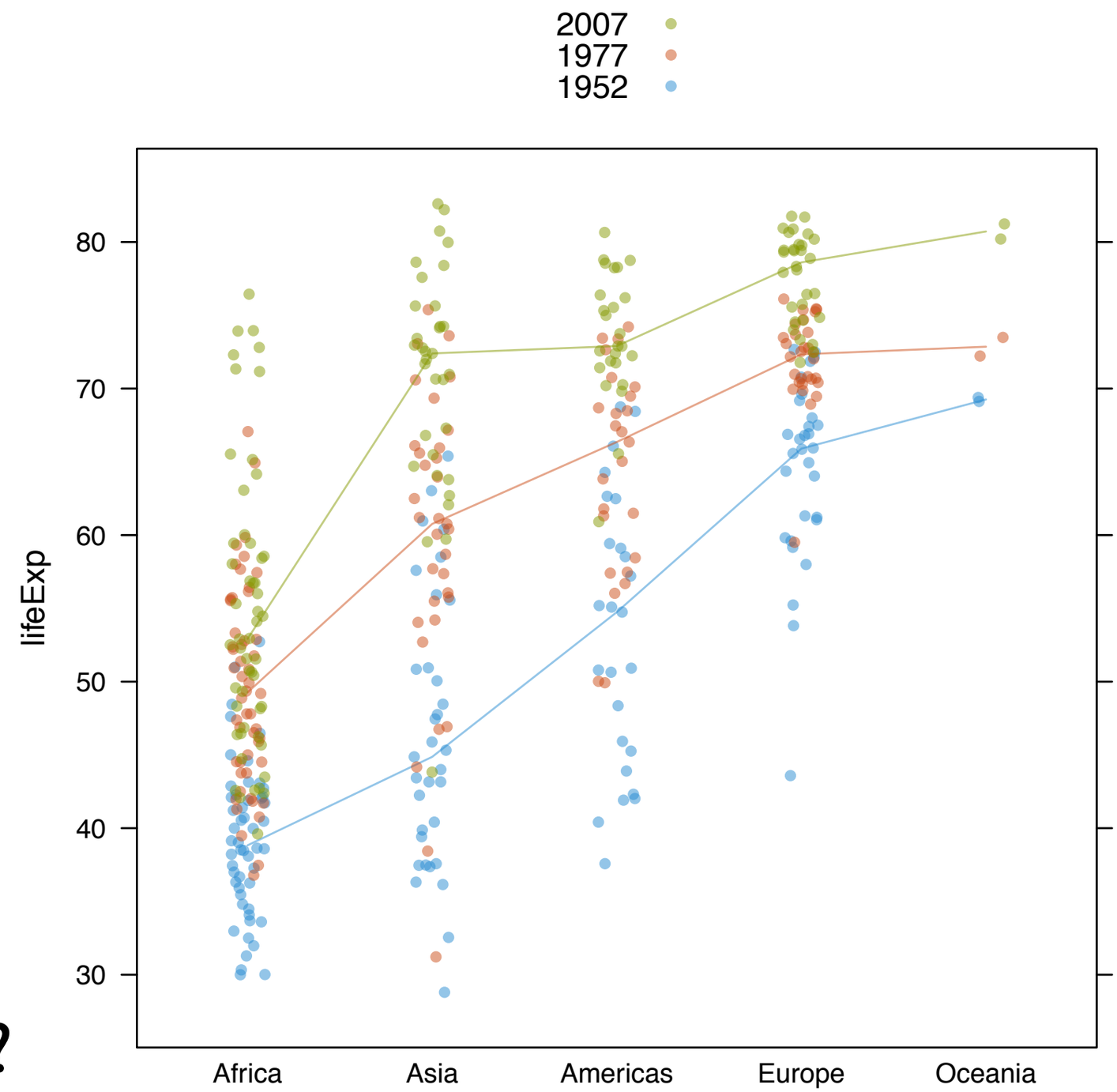
Affords opportunities to .....

confirm the expected

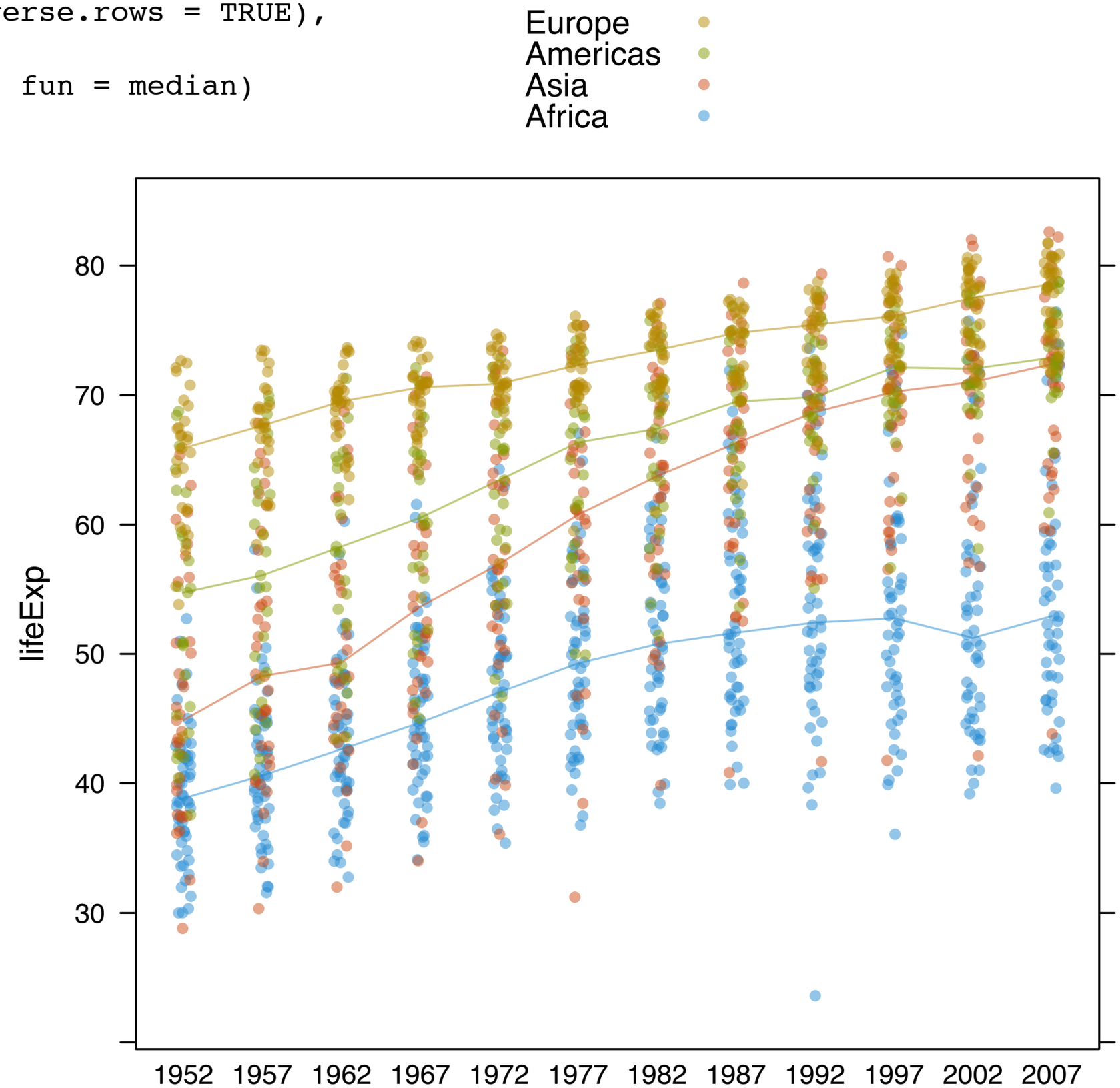
make comparisons:
across continents
across time within continent

identify trends:
change over time

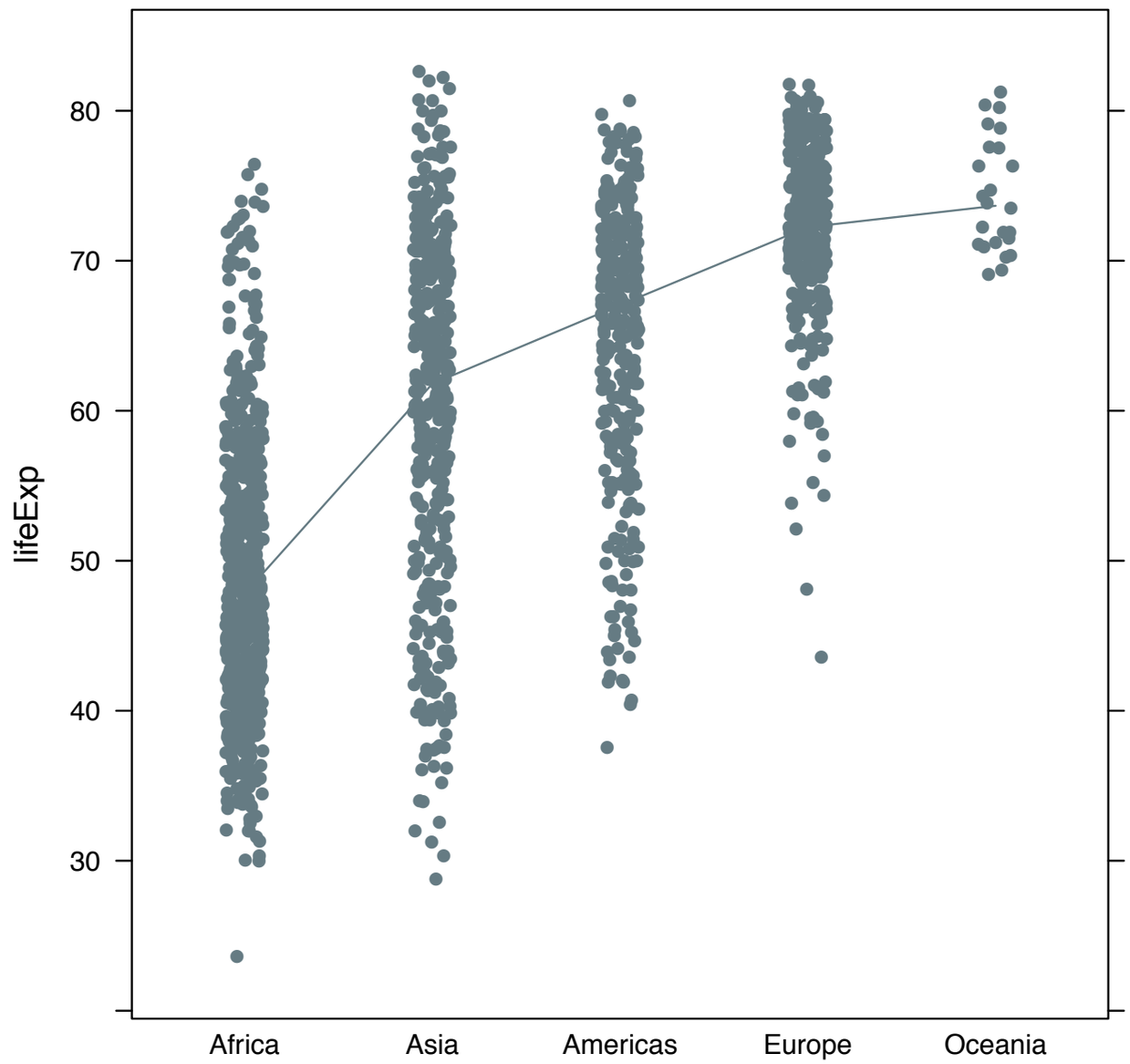
make comparisons of trends:
is change over time similar
or different across continent?



```
stripplot(lifeExp ~ factor(year),
          droplevels(subset(gDat, continent != "Oceania")),
          groups = reorder(continent, lifeExp),
          auto.key = list(reverse.rows = TRUE),
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```

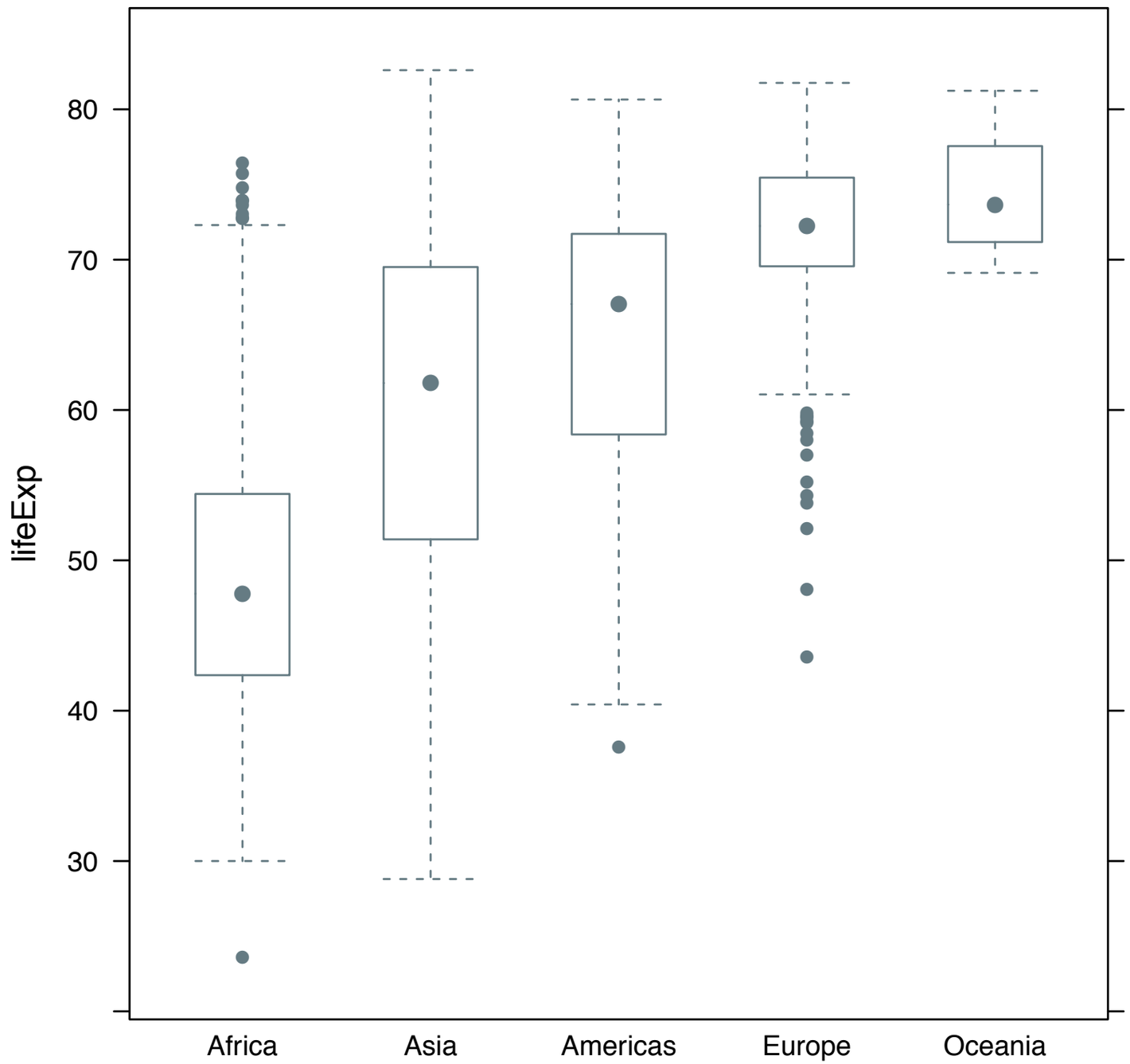


But what about larger datasets? Jittering is not enough. Overplotting remains a problem.



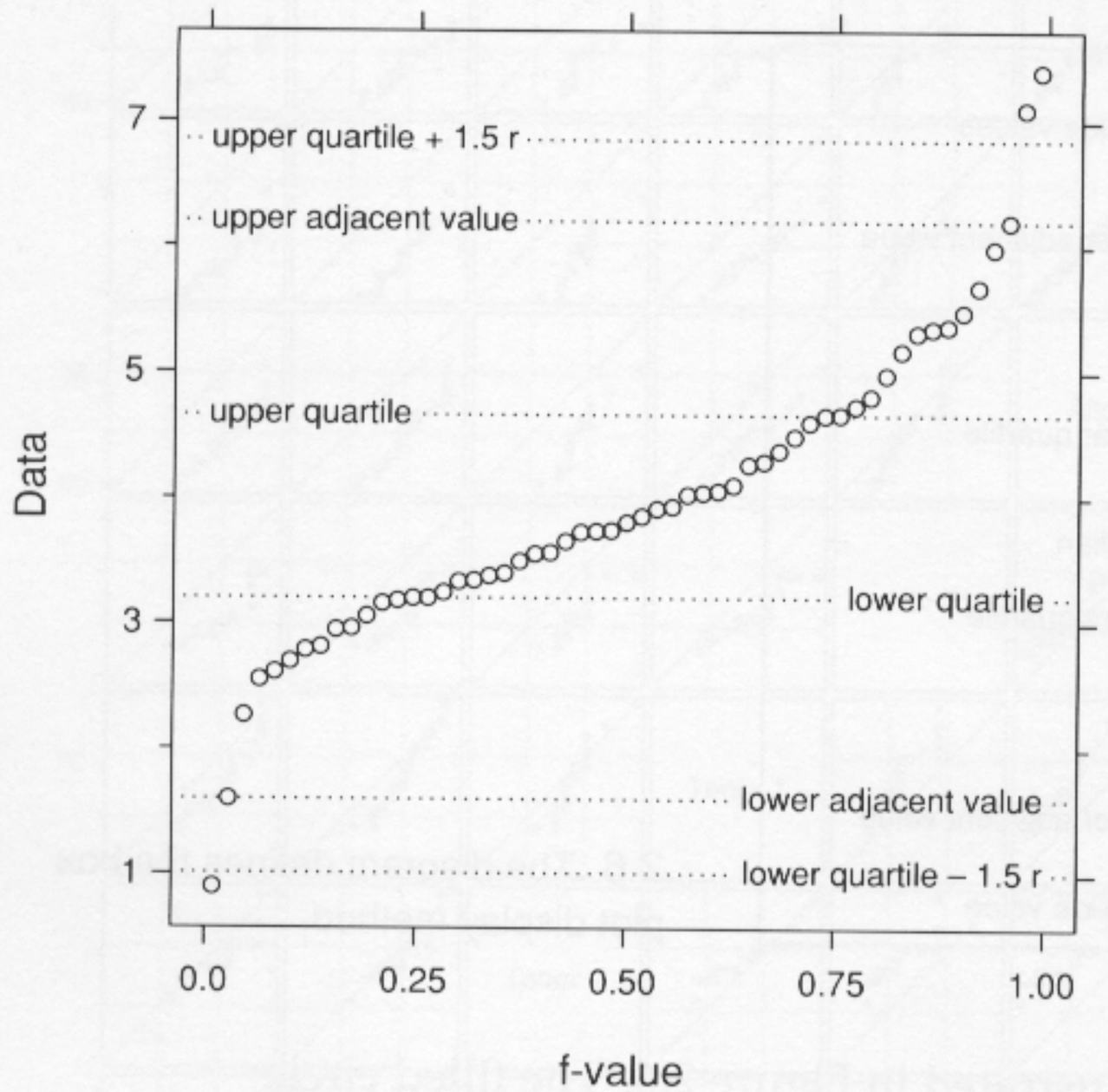

```
stripplot(lifeExp ~ reorder(continent, lifeExp), gDat,
          jitter.data = TRUE,
          type = c("p", "a"), fun = median)
```

boxplot or `box and whiskers' plot (hence 'bwplot()')




```
bwplot(lifeExp ~ reorder(continent, lifeExp), gDat)
```

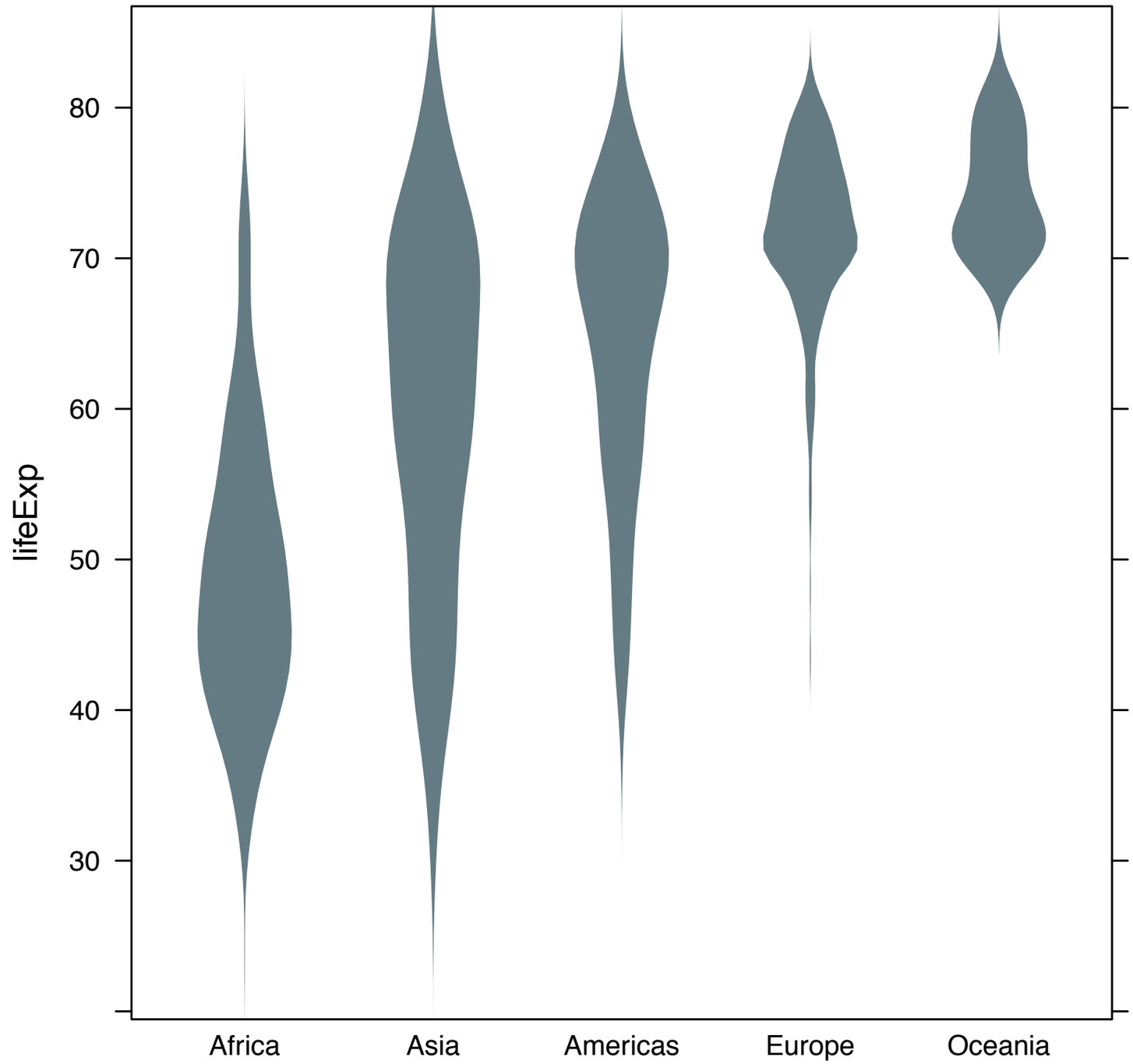
Where boxplots come from



2.7 The diagram illustrates the computation of the adjacent values, which are used in the box plot display method.

Figure from Visualizing Data by Cleveland.

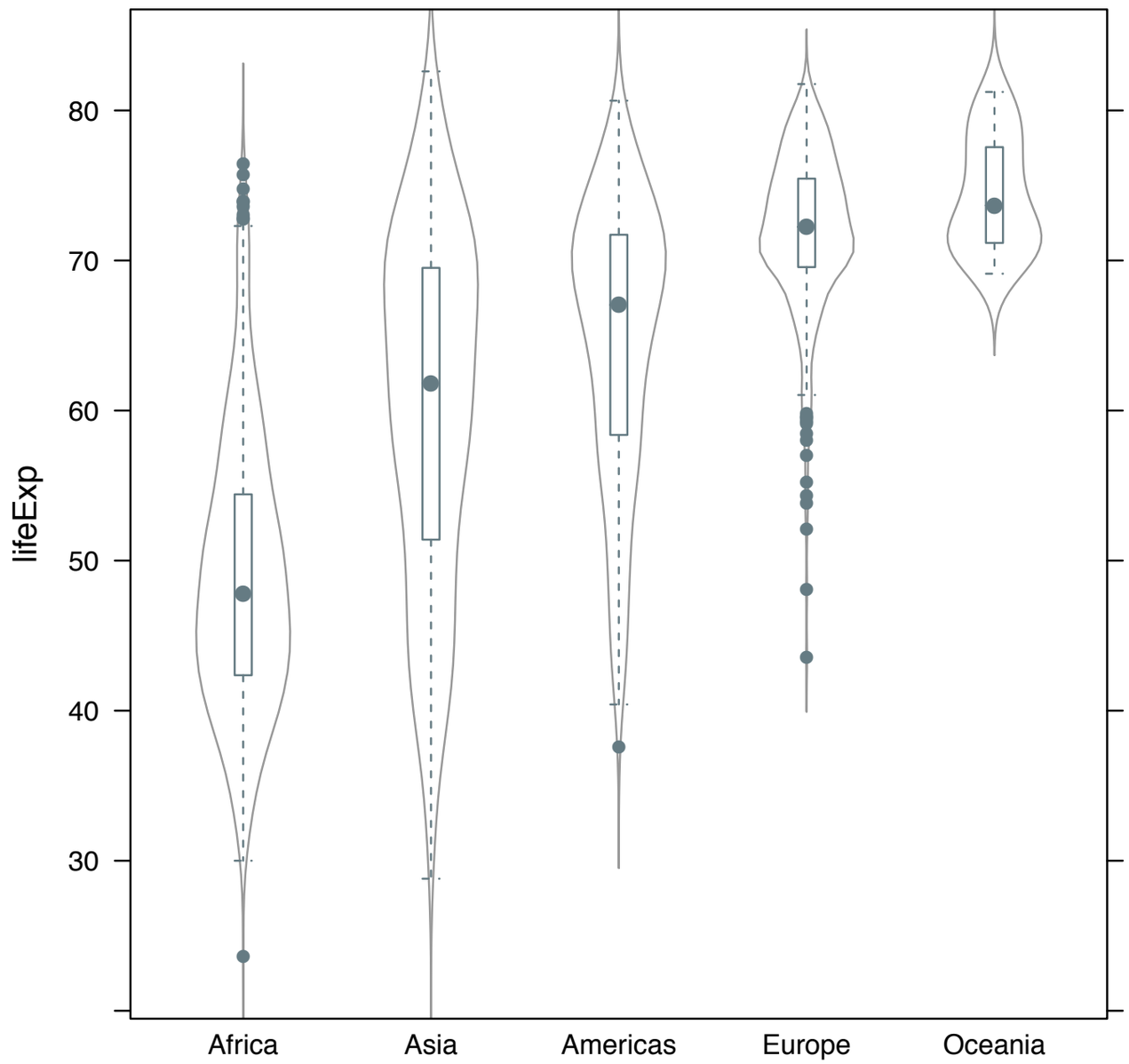
violin plot



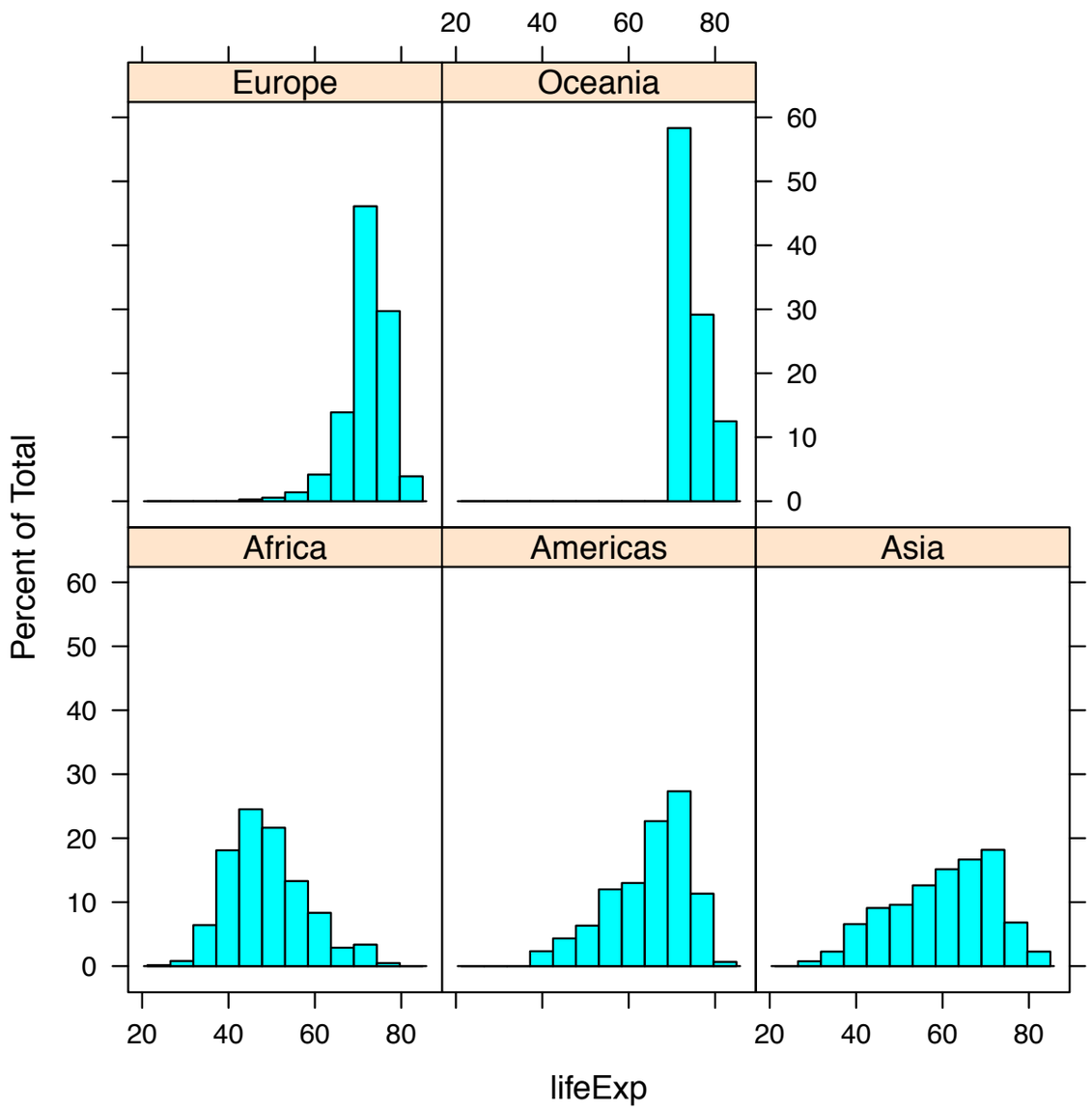

```
bwplot(lifeExp ~ reorder(continent, lifeExp), gDat,
       panel = panel.violin)
```

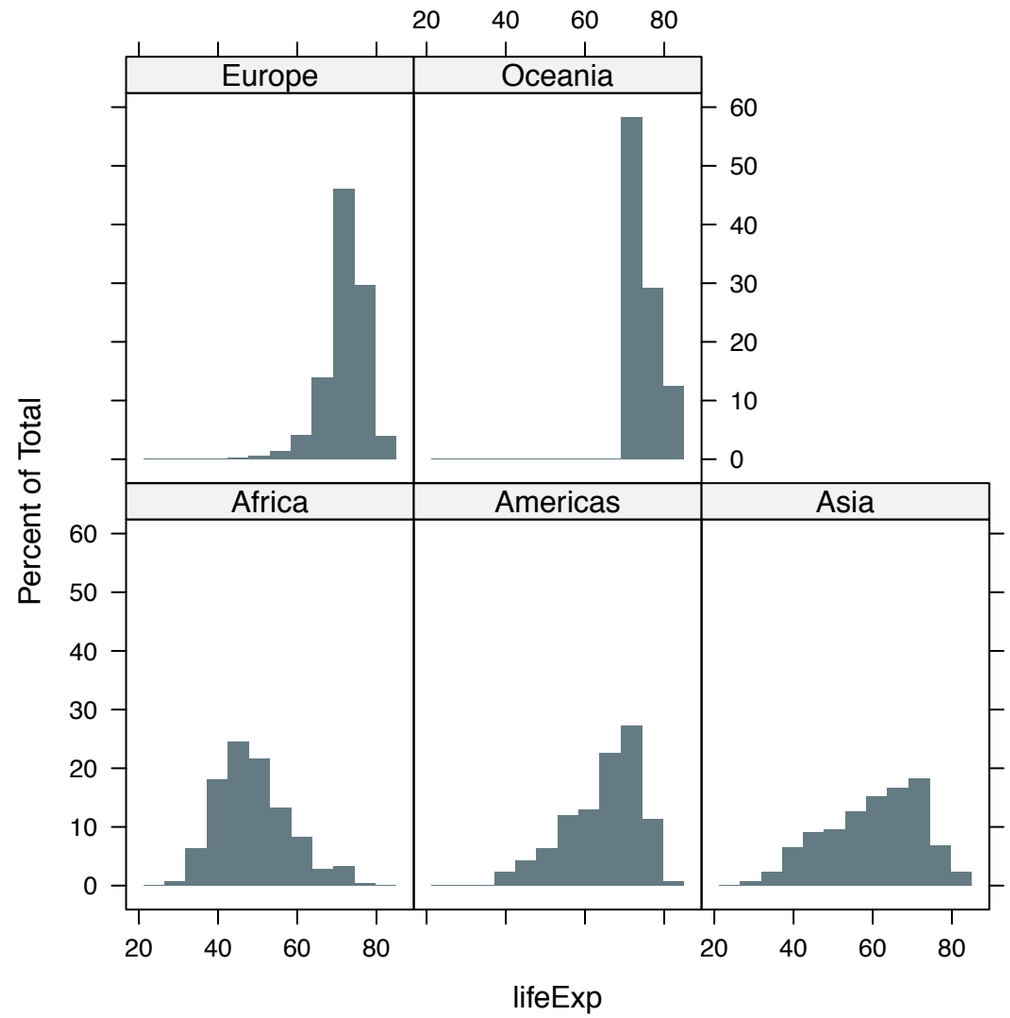
Note: I will talk explicitly about panel functions when we properly introduce lattice.

violin plot + boxplot



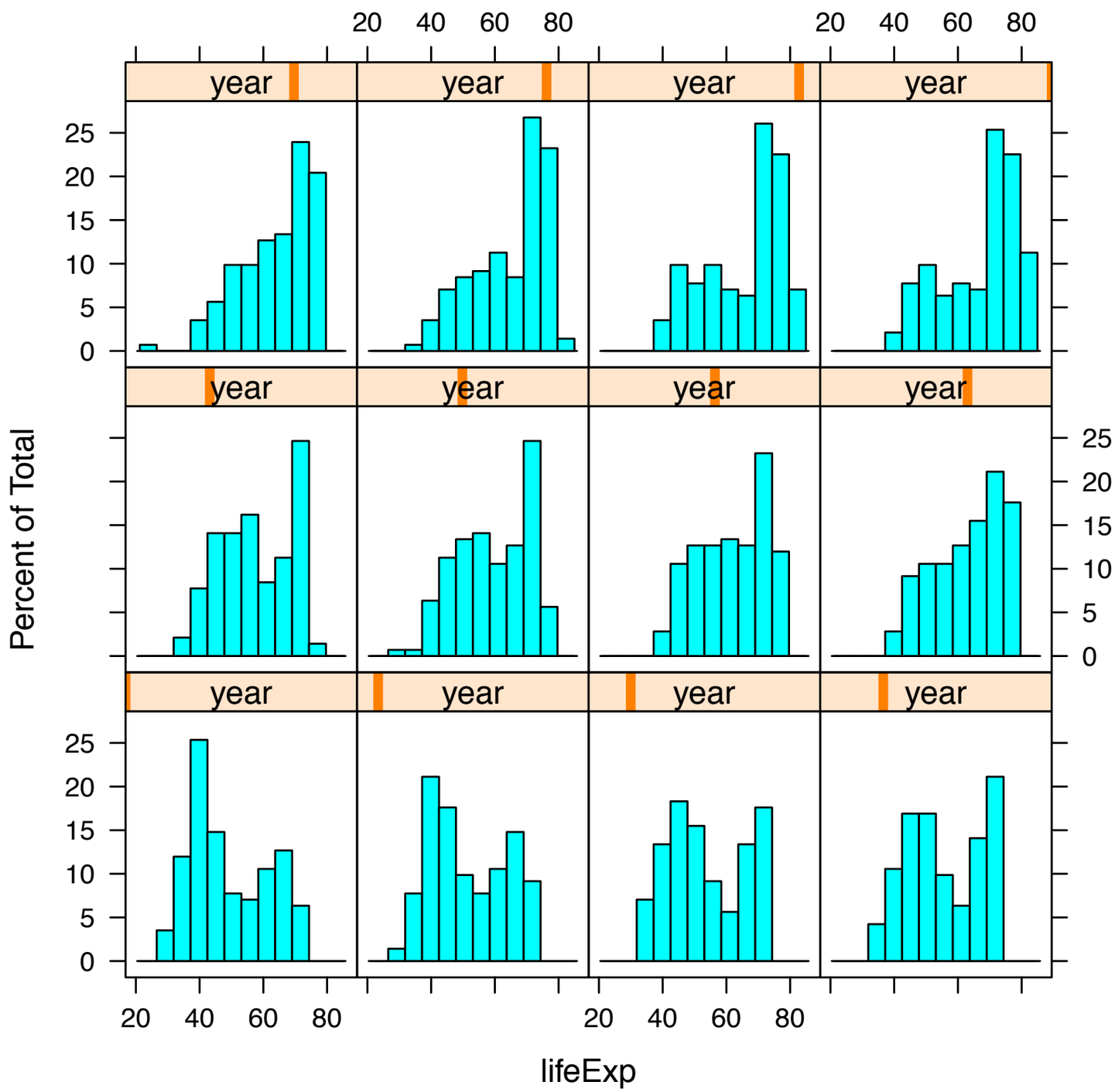

```
bwplot(lifeExp ~ reorder(continent, lifeExp), gDat,
       panel = function(..., box.ratio) {
         panel.violin(..., col = "transparent", border = "grey60",
                      varwidth = FALSE, box.ratio = box.ratio)
         panel.bwplot(..., fill = NULL, box.ratio = .1)
       })
```

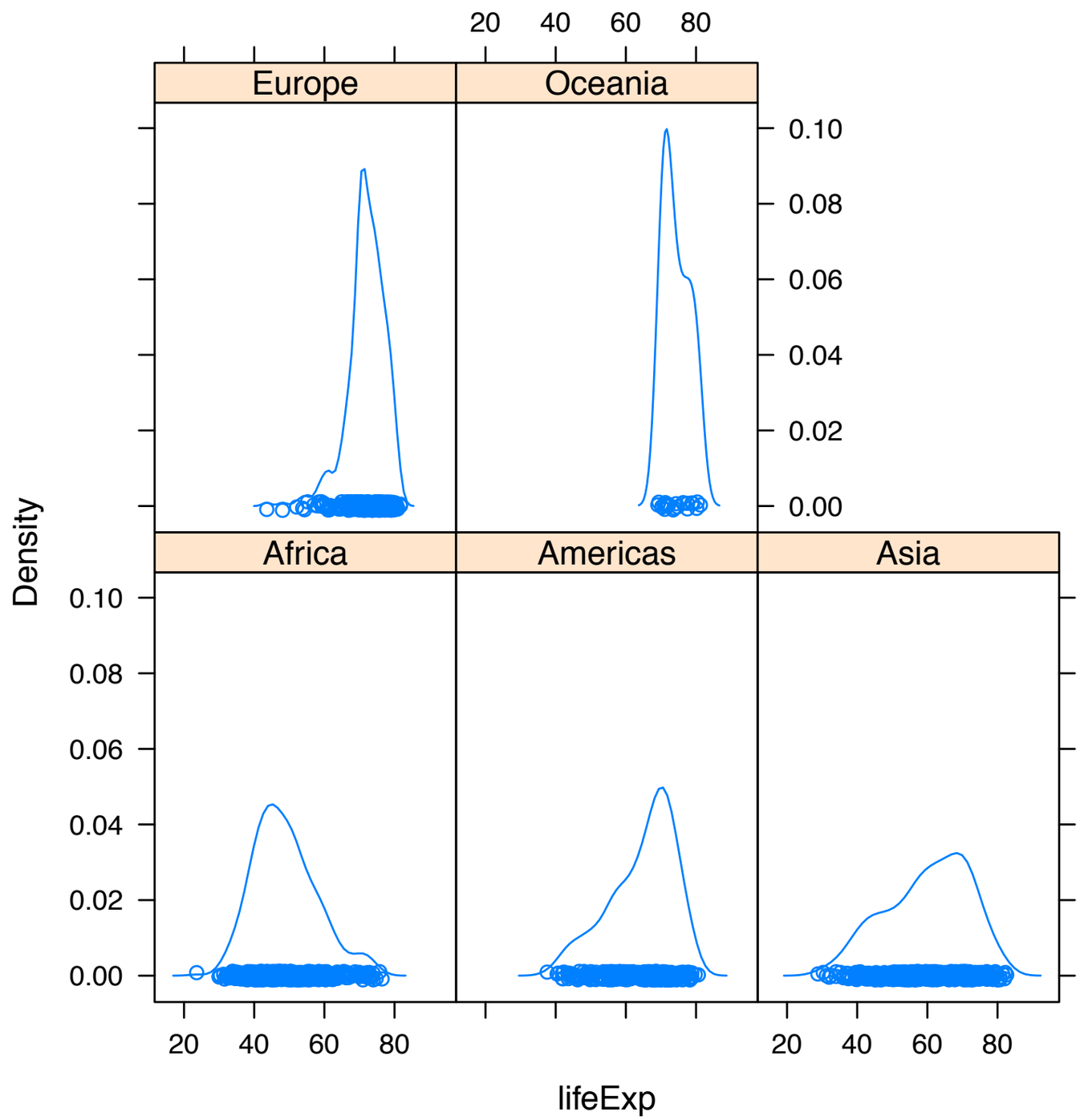





```
histogram(~ lifeExp | continent, gDat)
```

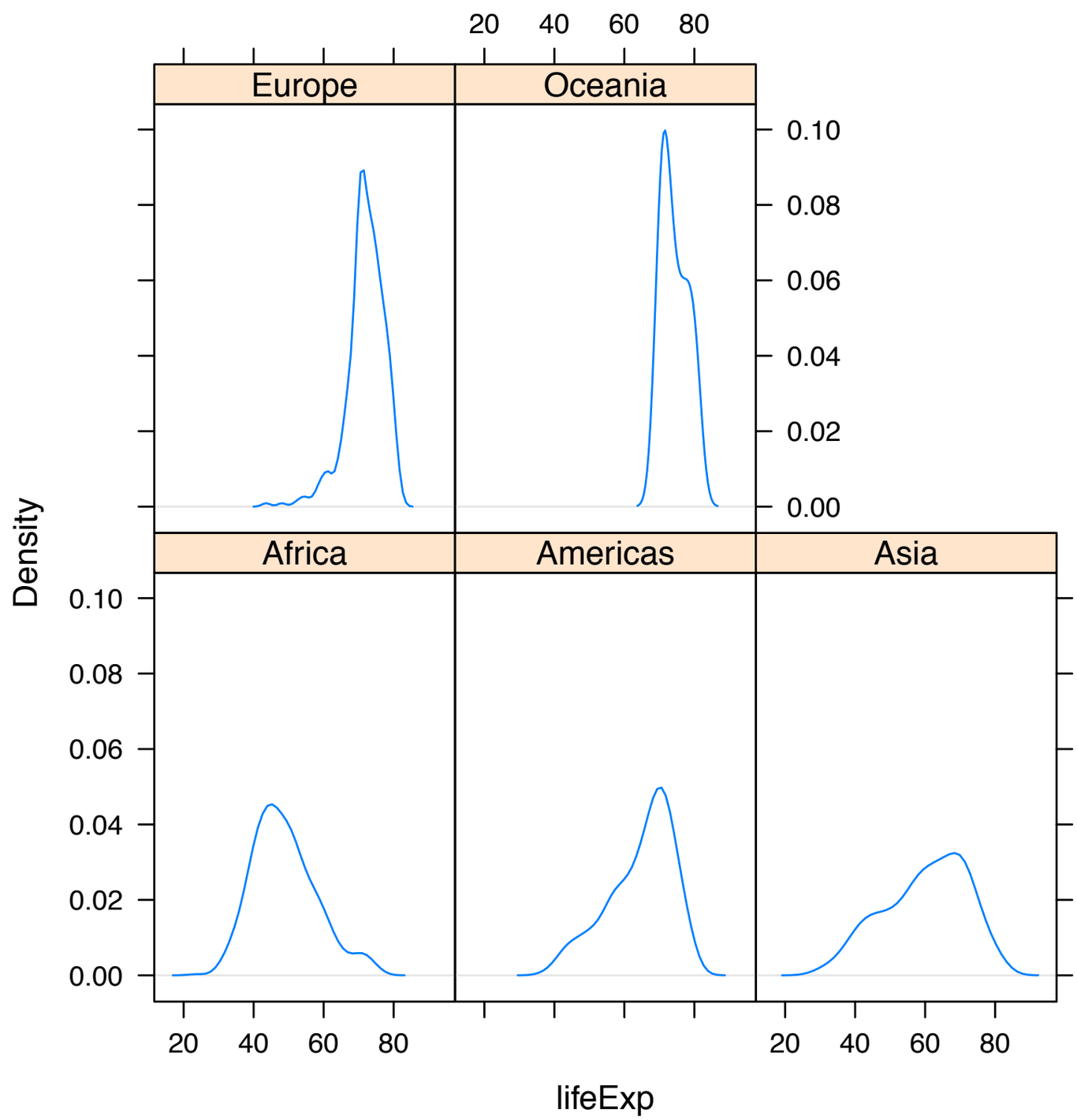
```
histogram(~ lifeExp | year, gDat)
```



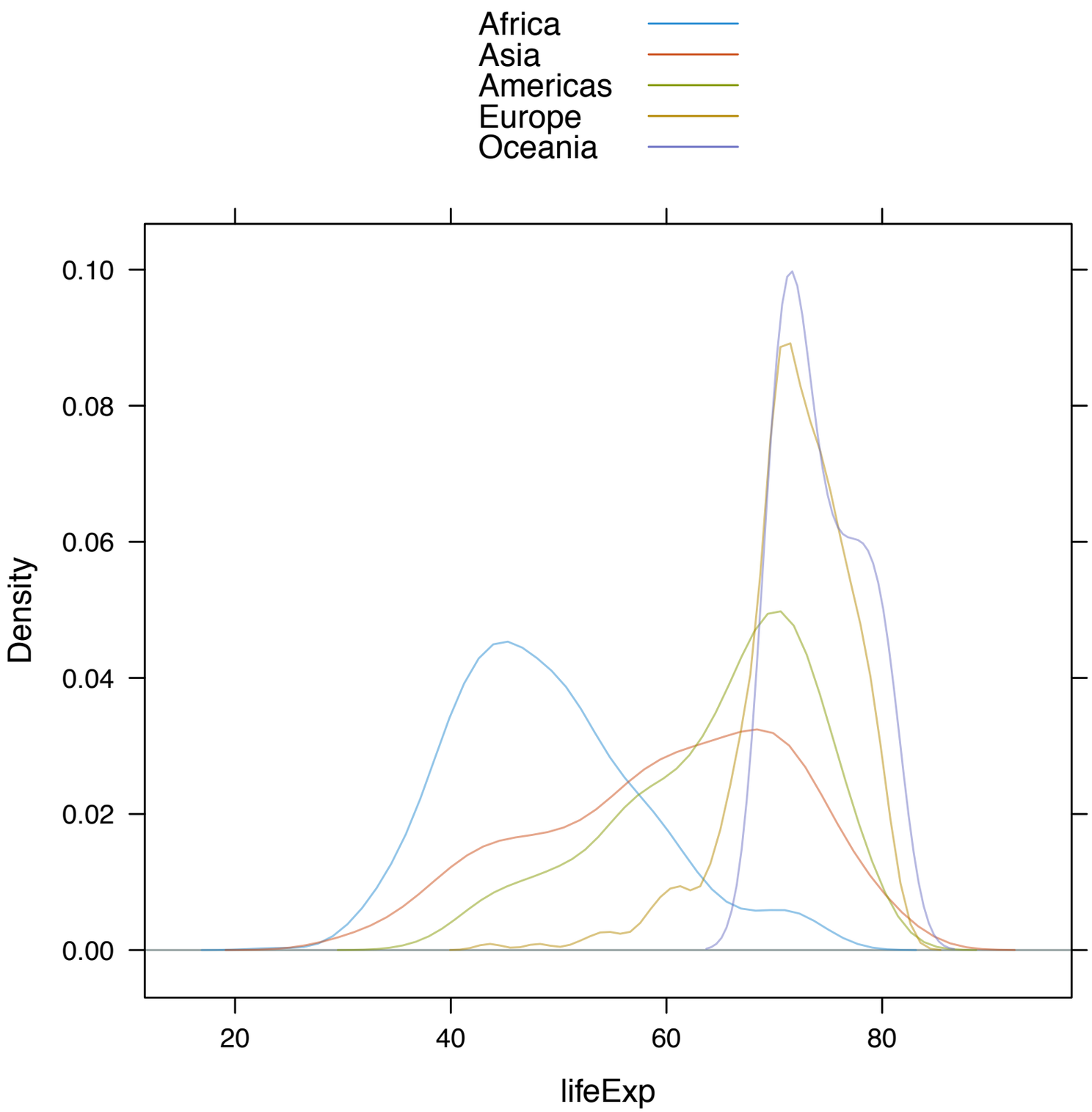



```
densityplot(~ lifeExp | continent, gDat)
```

```
densityplot(~ lifeExp | continent, gDat,
            plot.points = FALSE, ref = TRUE)
```



```
densityplot(~ lifeExp, gDat,
            groups = reorder(continent, lifeExp), auto.key = TRUE,
            plot.points = FALSE, ref = TRUE)
```



ability to superpose to facilitate direct visual comparison is big advantage of densityplot over histogram

using reorder() again so that order in key better matches order of the distributions

For medium-to-large datasets,
main data visualizations driven by

1. the density f
   a. histogram
   b. kernel density estimate
2. the CDF F
   1. box-and-whisker plot
   2. empirical cumulative distribution function

See Ch. 3 of Sarkar

Main data visualizations driven by

1. the density f
   a. histogram (`histogram`)
   b. kernel density estimate (`densityplot`)
2. the CDF F
   1. box-and-whisker plot (`bwplot`)
   2. empirical cumulative distribution function (`ecdfplot`)

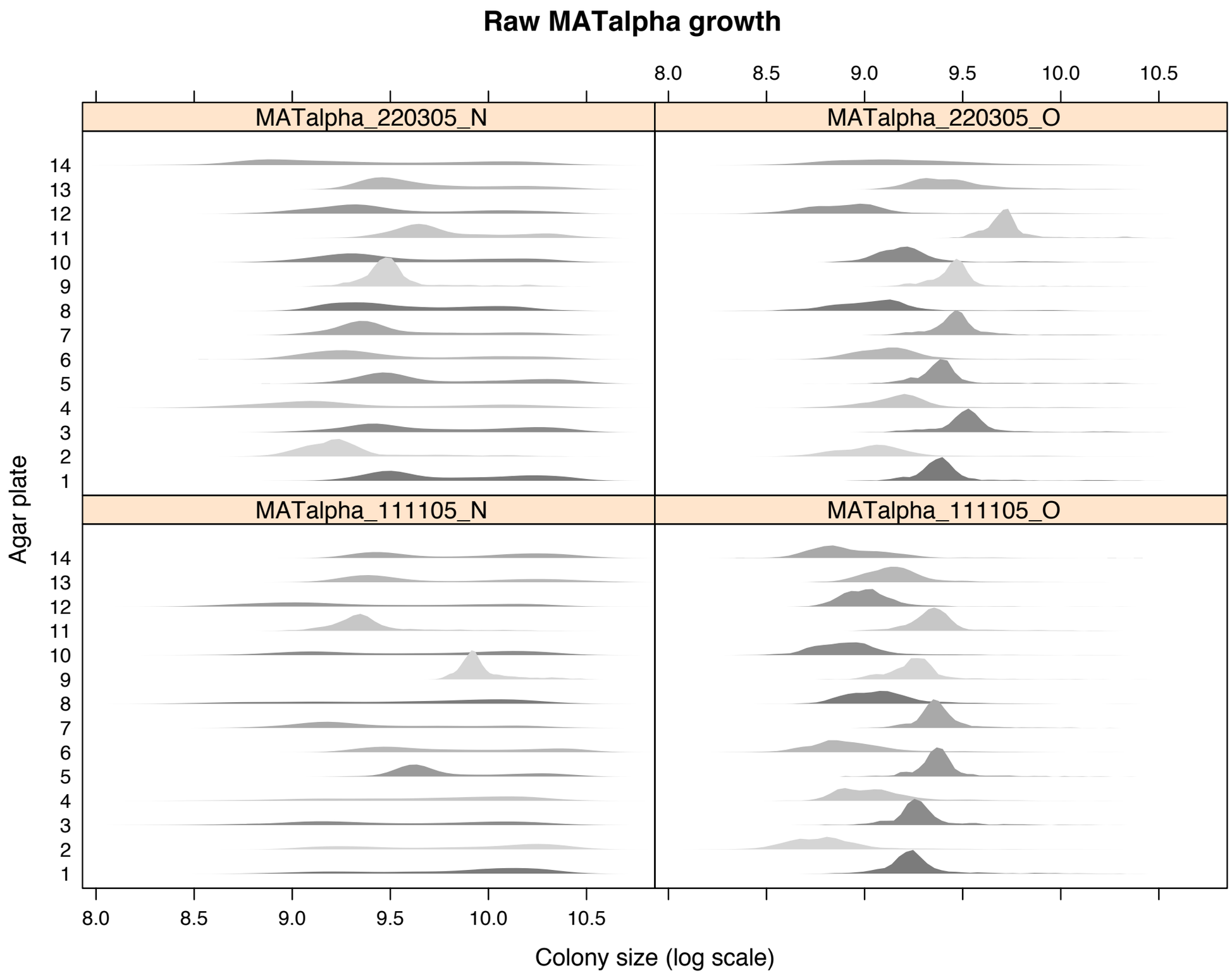
If `densityplot` and `bwplot` had a child ...
you might get a violin plot.

See Ch. 3 of Sarkar

functions from `lattice` or `latticeExtra`

✓=possible/ sensible	`~ x`	`y ~ x`	`~ x | y`	`~ x , groups = y`
`stripplot`		✓		
`bwplot`		✓		
`histogram`	✓		✓	
`densityplot`	✓	*	✓	✓
`ecdfplot`	✓		✓	✓

* I've actually extended densityplot to work here, for personal use. See next page.



I was so disappointed that y ~ x and y ~ x | z didn't work for densityplot, that I implemented that.

Why do I like densityplot better than histogram?

less sensitive (at least visually) to arbitrary choice of tuning parameter (bandwidth for densityplot, bin boundaries for histogram)

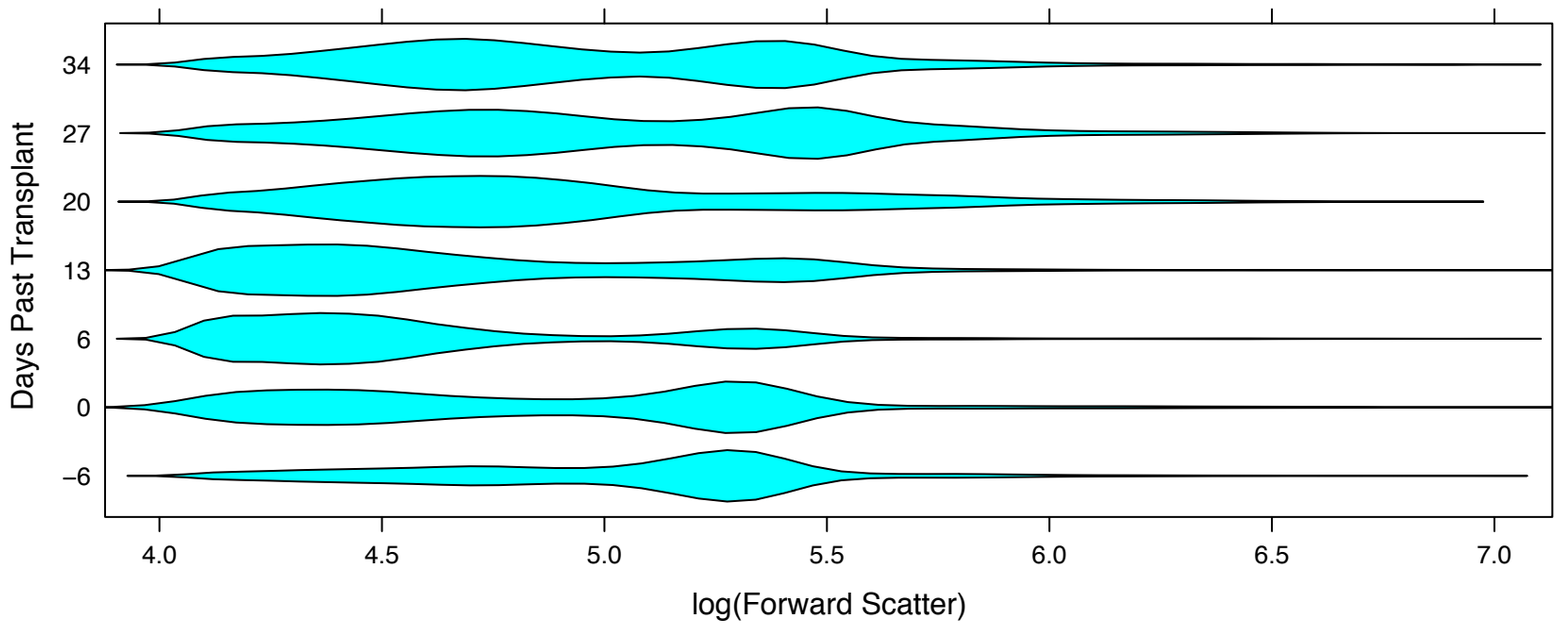
ability to superpose

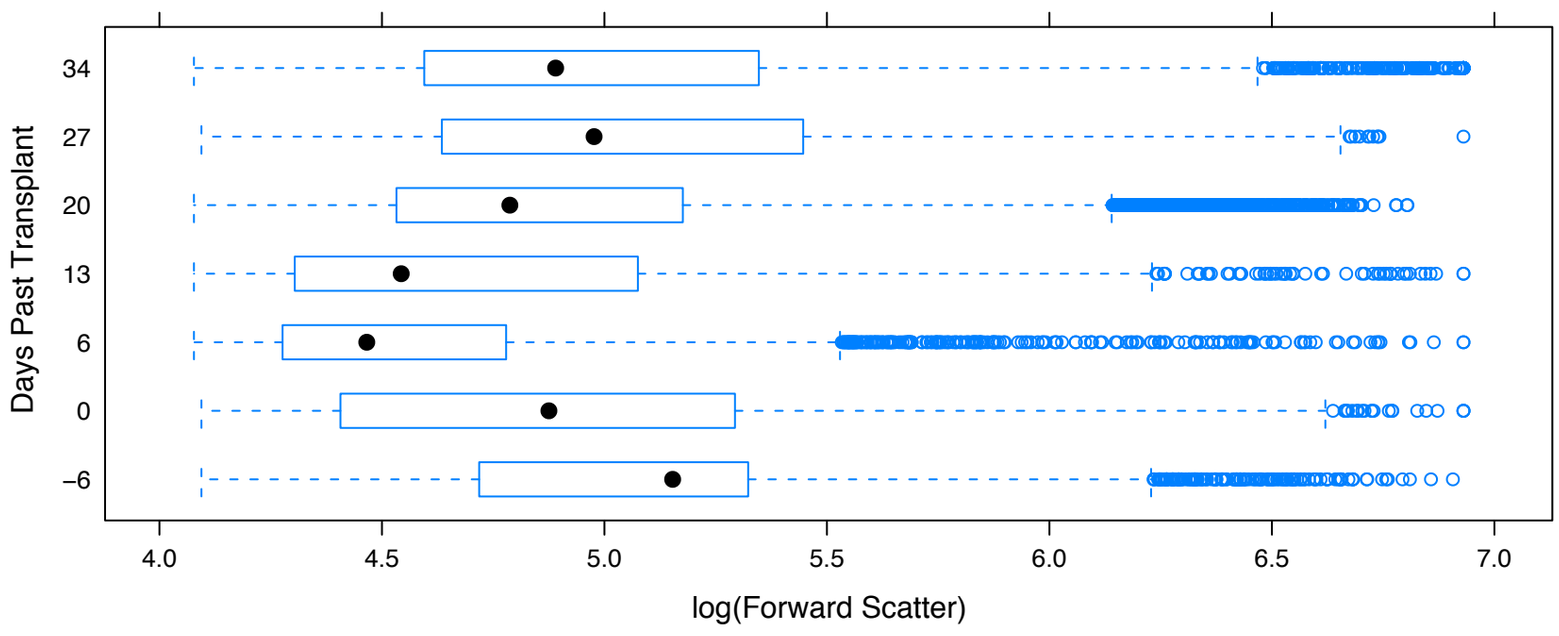
natural to include raw observed data in a rug

Why do I like violinplot and my version of densityplot better than boxplot?

ability to spot bimodality

Where boxplots fail





Flow cytometry data from five samples from a patient

Description:

Flow cytometry data from blood samples taken from a Leukemia patient before and after allogenic bone marrow transplant. The data spans five visits.

Usage:

```
data(gvhd10)
```

Format:

A data frame with 113896 observations on the following 8 variables.

'FSC.H' forward scatter height values

'SSC.H' side scatter height values

'FL1.H' intensity (height) in the FL1 channel

'FL2.H' intensity (height) in the FL2 channel

'FL3.H' intensity (height) in the FL3 channel

'FL2.A' intensity (area) in the FL2 channel

'FL4.H' intensity (height) in the FL4 channel

'Days' a factor with levels '-6' '0' '6' '13' '20' '27' '34'

Violin plot > boxplot?

Figure 3.13

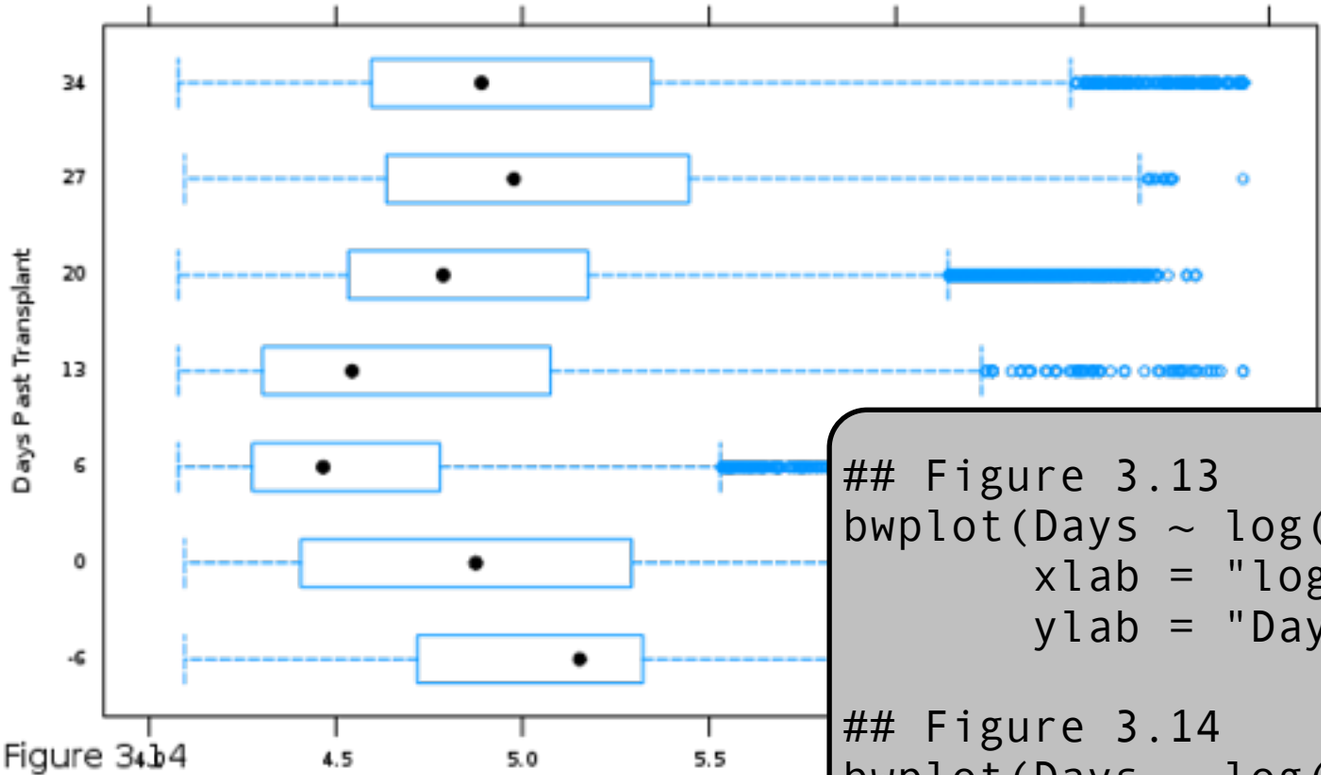
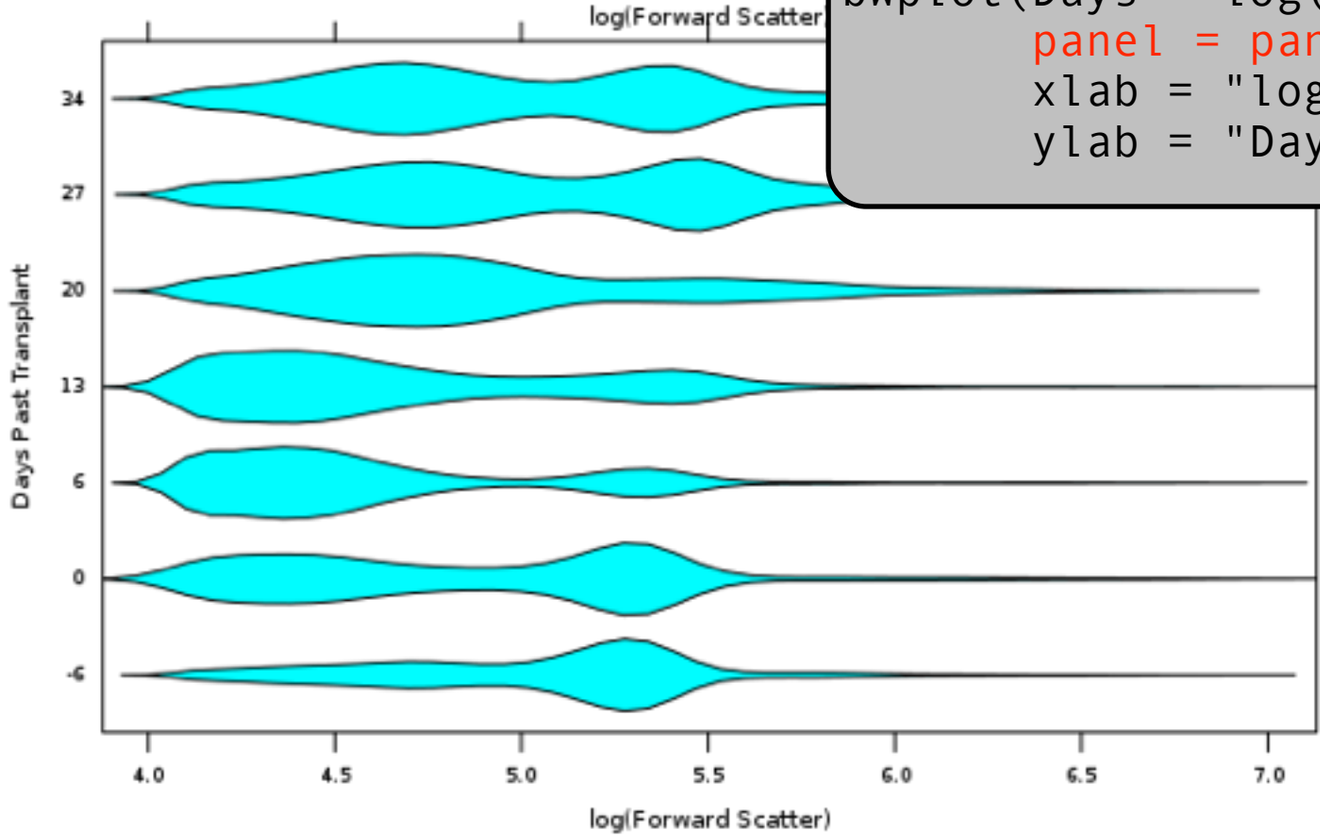


Figure 3.14




```
## Figure 3.13
bwplot(Days ~ log(FSC.H), data = gvhd10,
       xlab = "log(Forward Scatter)",
       ylab = "Days Past Transplant")

## Figure 3.14
bwplot(Days ~ log(FSC.H), gvhd10,
       panel = panel.violin, box.ratio = 3,
       xlab = "log(Forward Scatter)",
       ylab = "Days Past Transplant")
```

what about “empirical cumulative distribution plots” or ECDF plots?

Personally, I don’t have much use for them.

What is the empirical cumulative distribution (ecdf)?

$$\hat{F}_n(x) = \frac{\# \, x_i\text{'s} \;\leq\; x}{n}$$

$$\hat{F}_n(x) = \frac{1}{n}\sum_i I(x_i \leq x)$$

A step function that increases by 1/n at every observed value of X. The NPMLE of F.

histogram vs. densityplot

not a huge difference in what you can see/learn

Figure 3.4

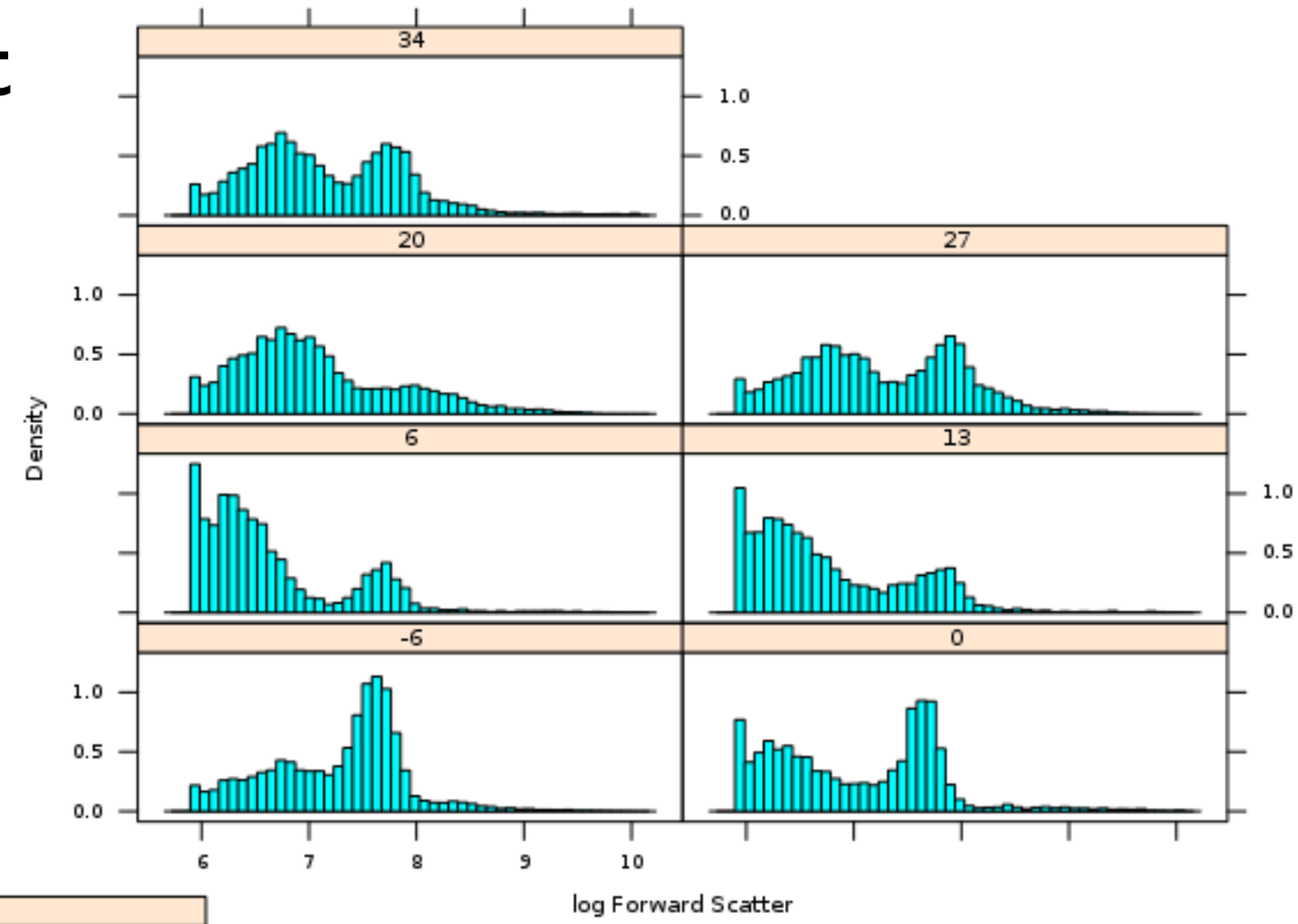
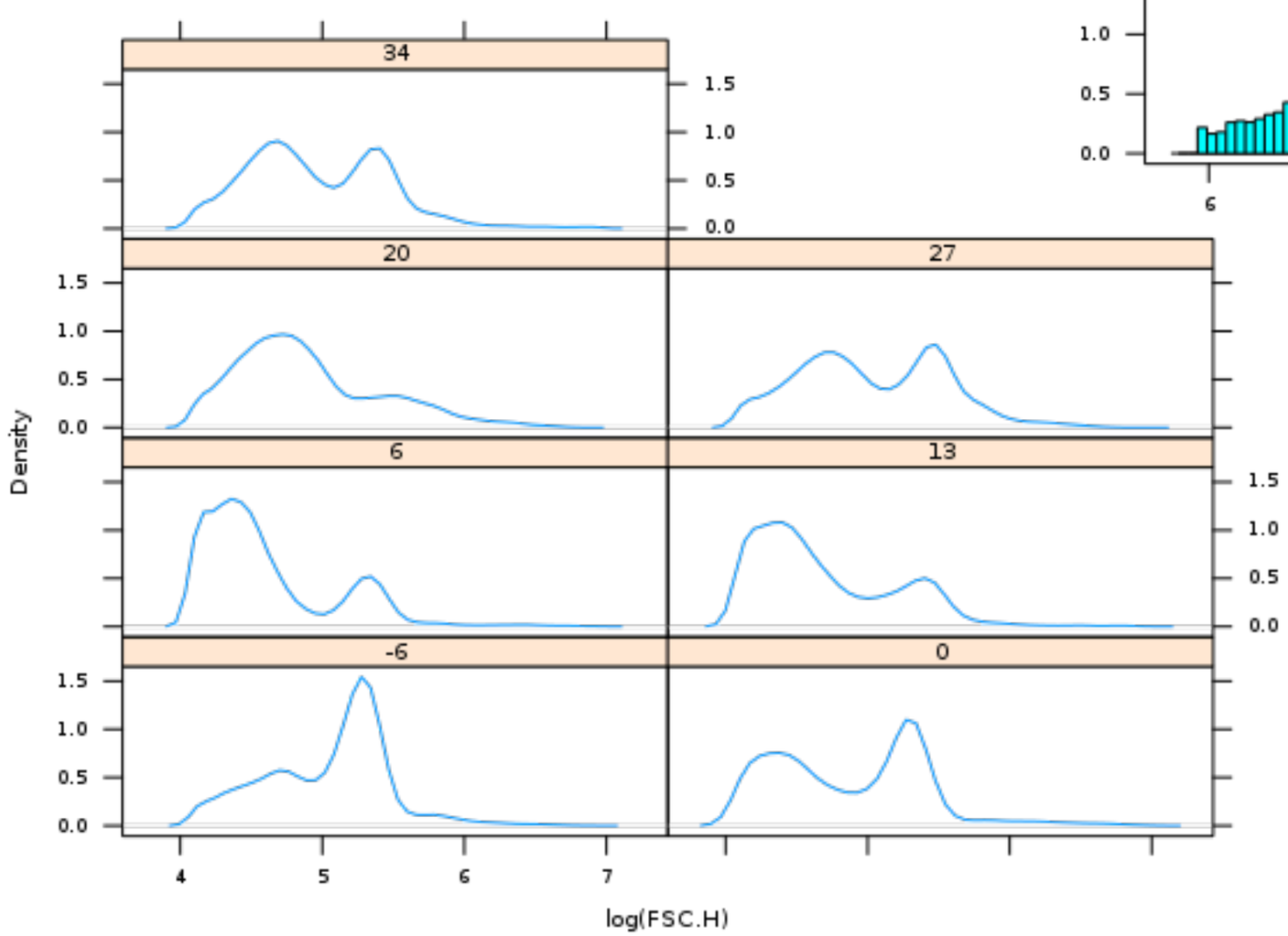


Figure 3.3



ecdfplot vs. densityplot

very different view of the data!

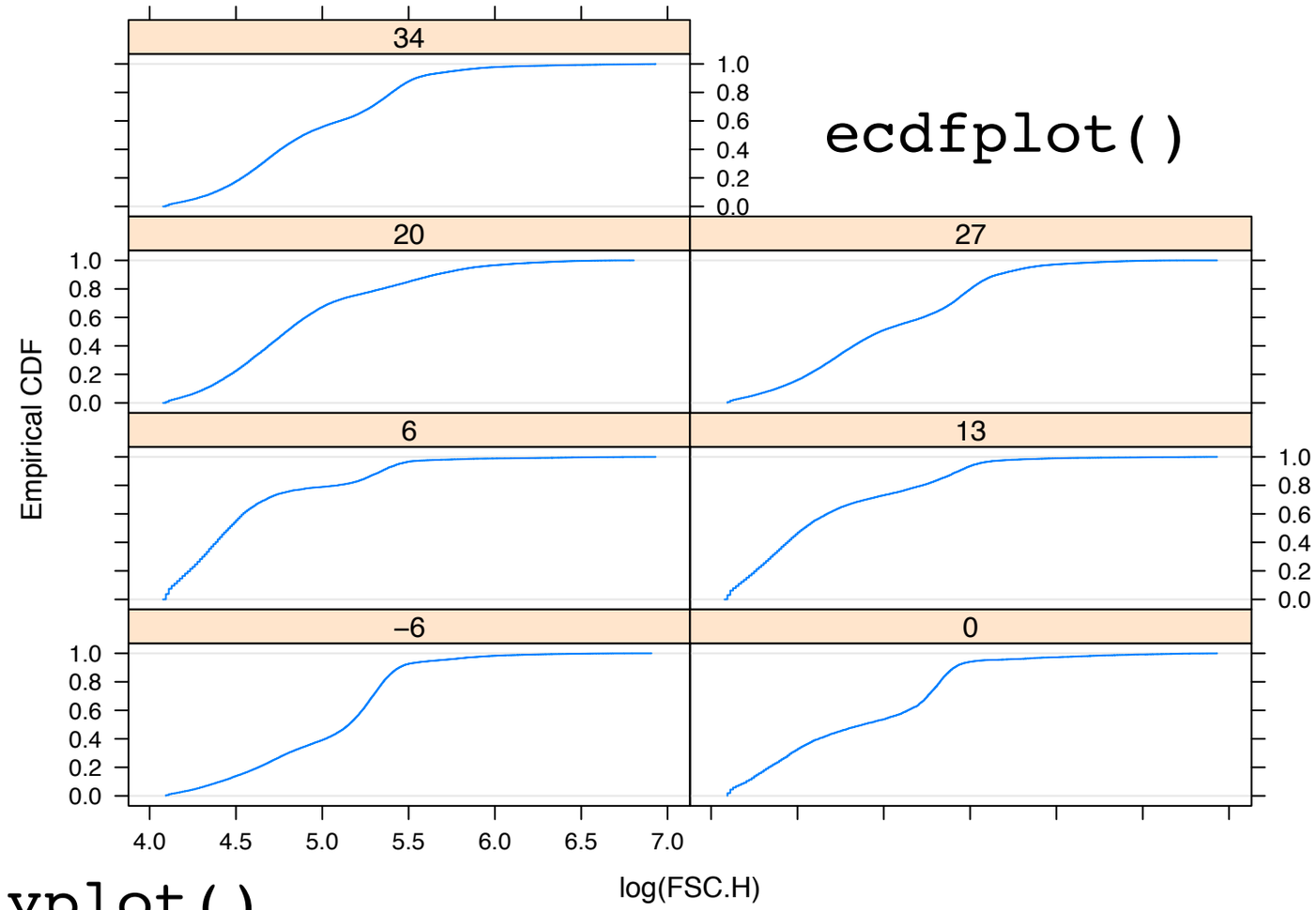
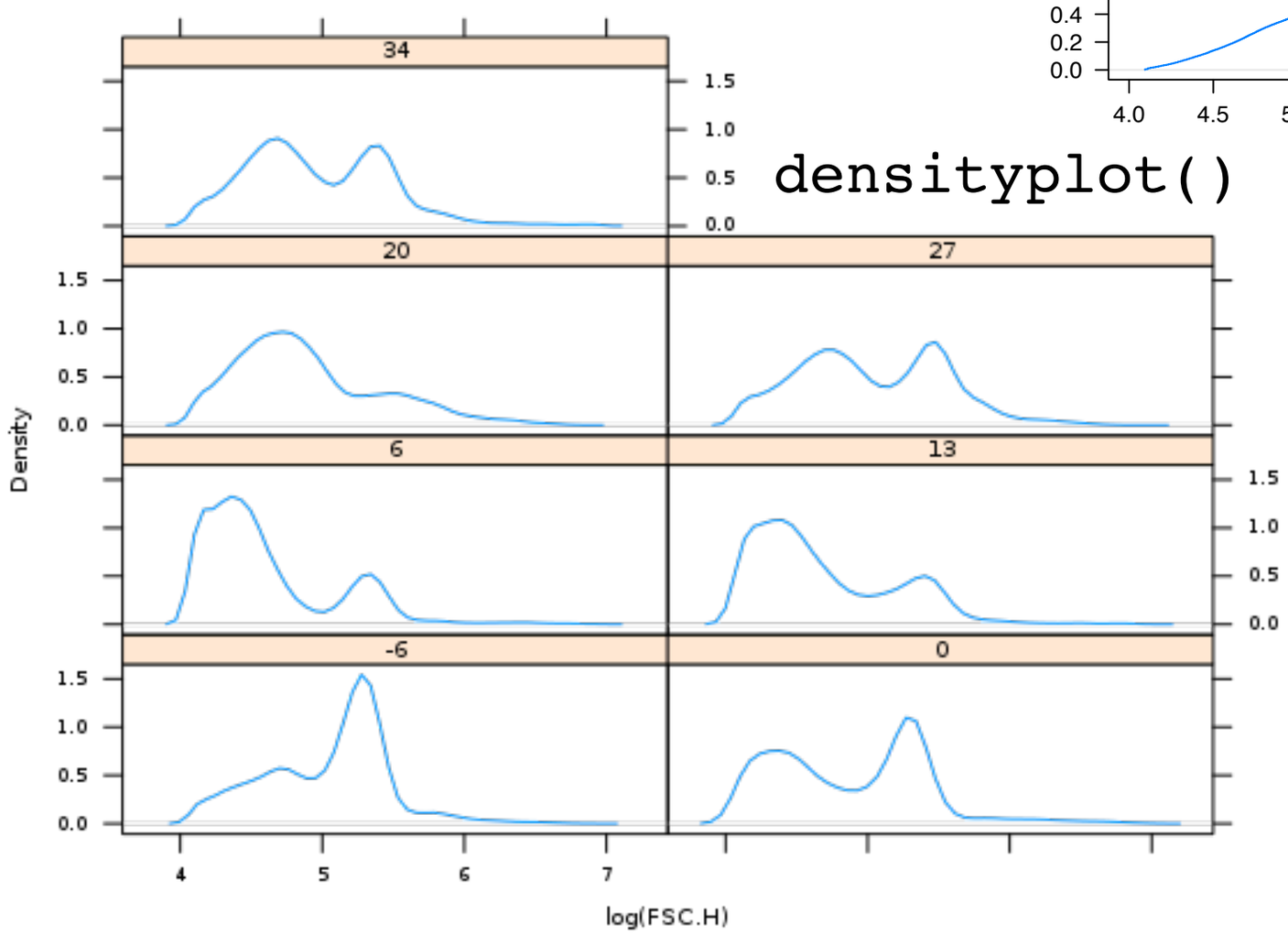


Figure 3.3

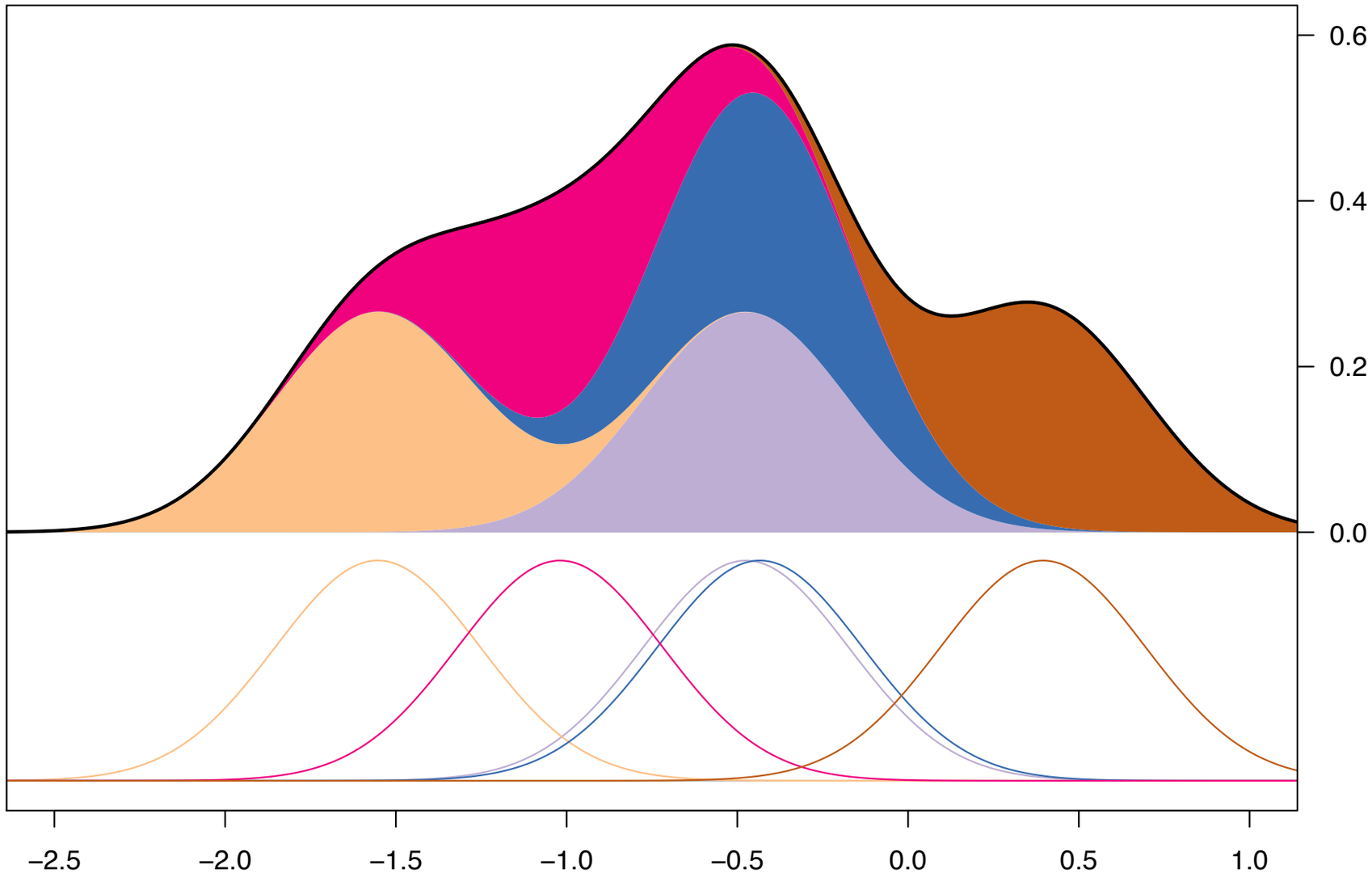


I cannot 'read' ecdfplots ... can you spot bimodality? What's the mean? Which distribution has greater spread?

Visualizing dist'n of X (given Y = y)

- I favor smooth histograms = density estimates. Path of least resistance is `densityplot`.
- Observed data, if sample small enough, can be overlaid via points or rug.
- In small datasets, strip plot is good, especially with summary statistic, such as median, overlaid.
- Boxplots and, in some very special cases, ecdf plots, seem useful. I like violin plots.
- Honestly, hard to find advantage of histograms, given all the other options.

Illustration of kernel density estimation



Produced from code at the R graph gallery

brief introduction to kernel density estimation

based on Camila Souza's presentation in STAT 545A (2008)

Histogram

Well-established, widely-practiced method of density estimation.

Basic principle: count the number of observations in an interval of size h, called a *bin*. Formally bin B_j is:

$$B_j = [x_0 + (j-1)h, x_0 + jh], j = 1,2,\ldots k$$

The histogram density estimate is:

$$\hat{f}_h(x) = \frac{1}{nh} \sum_i \sum_j I(x_i \in B_j) I(x \in B_j)$$

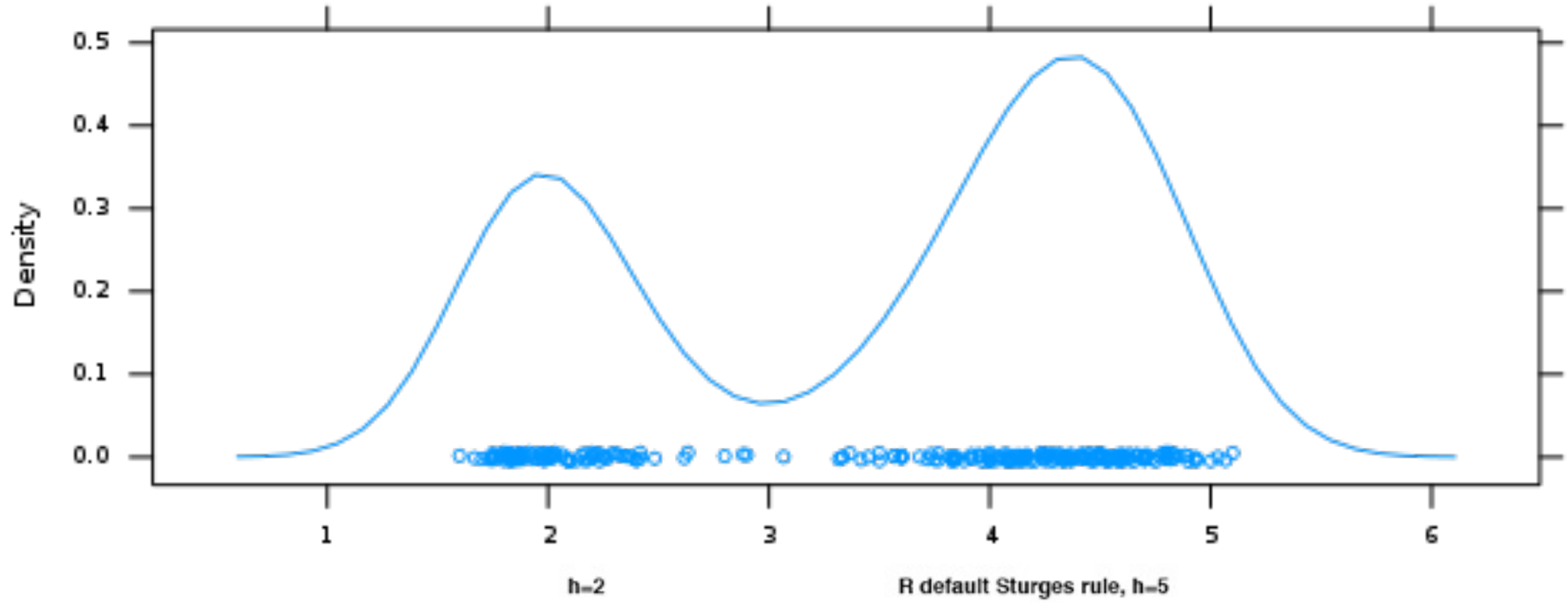
Histogram

Crucial 'tuning' parameter for histogram density estimation: the bins (or bin widths or number of bins)

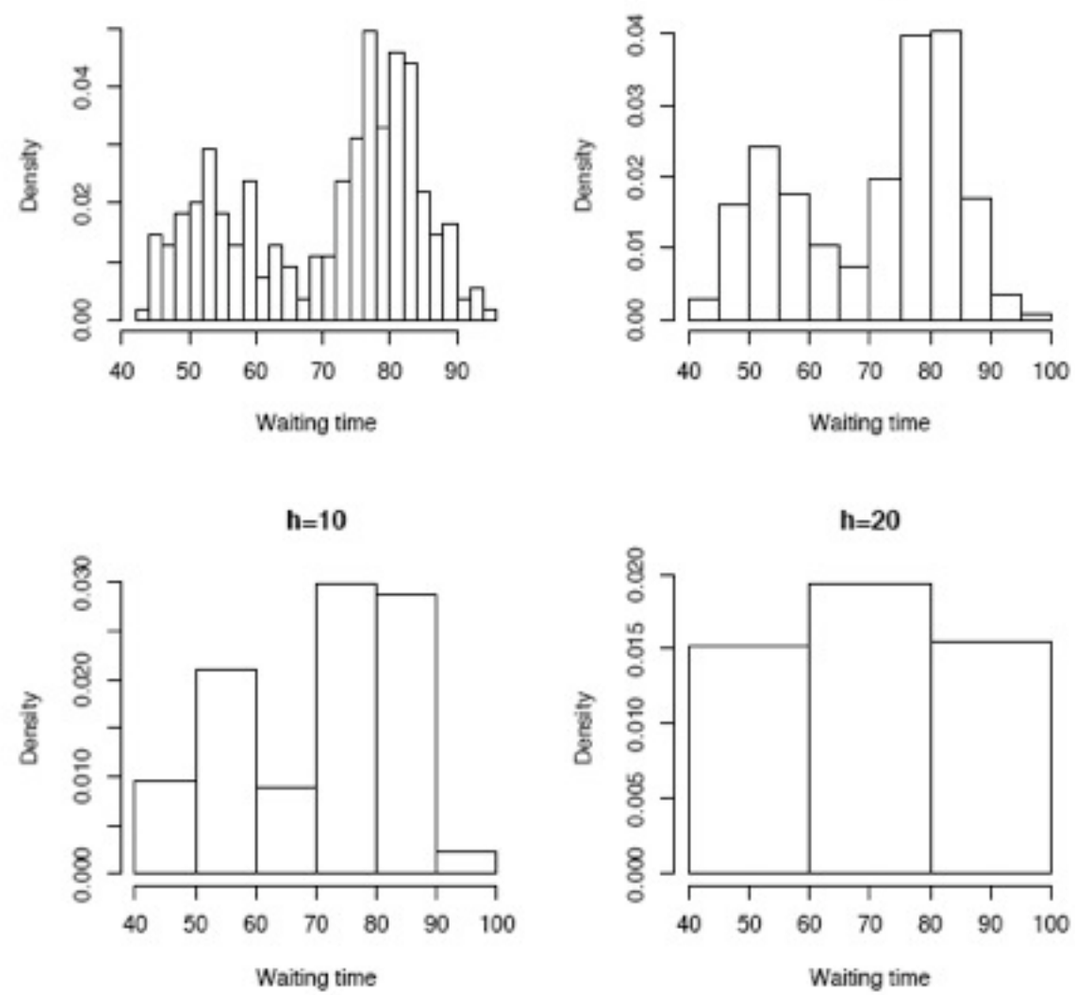
hist() base R	$k = 1 + \log_2 n$
truehist() MASS	$h = 3.5\hat{\sigma}n^{-1/3}$
histogram() lattice	$k = \text{round}(1 + \log_2 n)$

Figure 3.1

`faithful` data set



bin selection has a huge impact on the result!



Naive estimator, uniform kernel estimator

Remember definition of the density f:

$$f(x) = \lim_{h \to 0} \frac{1}{2h} P(x - h < X < x + h)$$

Therefore, for small h, a crude estimator of f is:

$$\hat{f}_h(x) = \frac{1}{2nh} [\# \, x_i \in (x - h, x + h)]$$

Naive estimator, uniform kernel estimator

Therefore, for small h, a crude estimator of f is:

$$\hat{f}_h(x) = \frac{1}{2nh}[\# \, x_i \in (x-h, x+h)]$$

Define a weight function:

$$w(x) = \frac{1}{2} \text{ if } |x| < 1 \text{ and } 0 \text{ otherwise}$$

And re-write the crude / naive estimator as:

$$\hat{f}_h(x) = \frac{1}{n}\sum_i \frac{1}{h} w\left(\frac{x - x_i}{h}\right)$$

Naive estimator, uniform kernel estimator

And re-write the crude / naive estimator as:

$$\hat{f}_h(x) = \frac{1}{n}\sum_i \frac{1}{h} w\left(\frac{x - x_i}{h}\right)$$

In plain English, place a box of width 2h and height $(2nh)^{-1}$ on each observation. Density estimate at any point x is the sum of these boxes.

Moving beyond a uniform (or rectangular) kernel

Let's replace the weight function with another function K that satisfies the following:

$$K(x) \geq 0$$

$$\int K(x)dx = 1$$

So K is a probability density function and, usually, is symmetric.

Tabela 1: *Kernels*

Kernel	$Kern(u)$
Uniform	$\frac{1}{2}I(\lvert u\rvert \leq 1)$
Triangle	$(1-\lvert u\rvert)I(\lvert u\rvert \leq 1)$
Epanechnikov	$0.75(1-u^2)I(\lvert u\rvert \leq 1)$
Gaussian	$\frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}u^2)$

In general, the kernel estimator is given by:

$$\hat{f}_h(x) = \frac{1}{nh}\sum_i K\left(\frac{x - x_i}{h}\right)$$

Tuning parameter h is called the *bandwidth*

In general, the kernel estimator is given by:

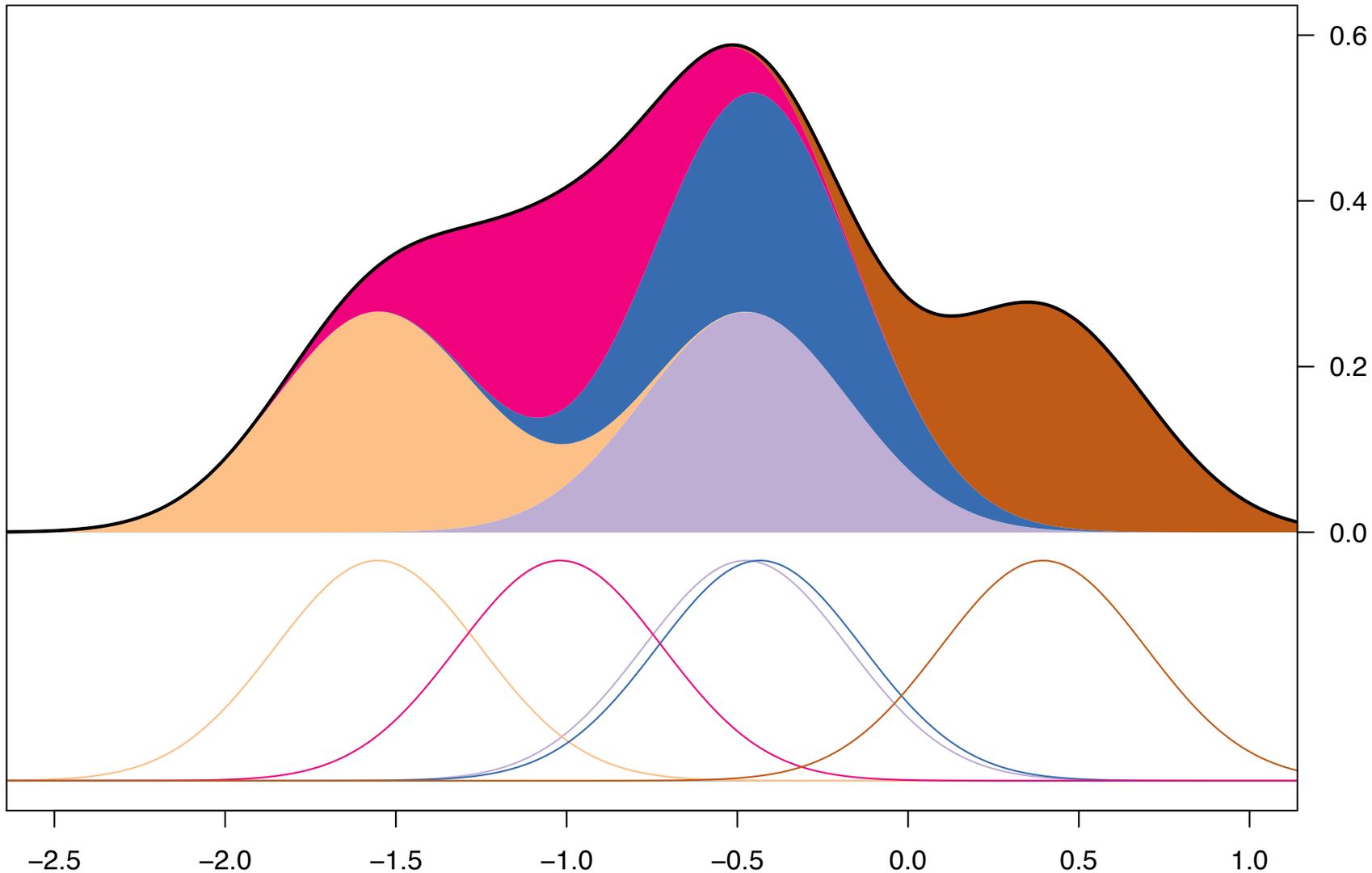
$$\hat{f}_h(x) = \frac{1}{nh} \sum_i K\left(\frac{x - x_i}{h}\right)$$

Tuning parameter h is called the *bandwidth*

Instead of a sum of boxes, the kernel estimator is a sum of 'bumps'. K determines the shape of the bumps and h determines their width.

Later, we will talk about how to choose h with cross-validation.

Illustration of kernel density estimation



Produced from code at the R graph gallery

```
density                 package:stats                 R Documentation

Kernel Density Estimation

Description:

     The (S3) generic function 'density' computes kernel density
     estimates.  Its default method does so with the given kernel and
     bandwidth for univariate observations.

Usage:

     density(x, ...)
     ## Default S3 method:
     density(x, bw = "nrd0", adjust = 1,
             kernel = c("gaussian", "epanechnikov", "rectangular",
                        "triangular", "biweight",
                        "cosine", "optcosine"),
             weights = NULL, window = kernel, width,
             give.Rkern = FALSE,
             n = 512, from, to, cut = 3, na.rm = FALSE, ...)

Arguments:

       x: the data from which the estimate is to be computed.

      bw: the smoothing bandwidth to be used.  The kernels are scaled
          such that this is the standard deviation of the smoothing
          kernel.  (Note this differs from the reference books cited
          below, and from S-PLUS.)

          'bw' can also be a character string giving a rule to choose
          the bandwidth.  See 'bw.nrd'.
          The default, '"nrd0"', has remained the default for
          historical and compatibility reasons, rather than as a
          general recommendation, where e.g., '"SJ"' would rather fit,
          see also V&R (2002).

          The specified (or computed) value of 'bw' is multiplied by
          'adjust'.

  adjust: the bandwidth used is actually 'adjust*bw'.  This makes it
          easy to specify values like 'half the default' bandwidth.

kernel, window: a character string giving the smoothing kernel to be
          used. This must be one of '"gaussian"', '"rectangular"',
          '"triangular"', '"epanechnikov"', '"biweight"', '"cosine"' or
          '"optcosine"', with default '"gaussian"', and may be
          abbreviated to a unique prefix (single letter).

          '"cosine"' is smoother than '"optcosine"', which is the usual
          'cosine' kernel in the literature and almost MSE-efficient.
          However, '"cosine"' is the version used by S.
```

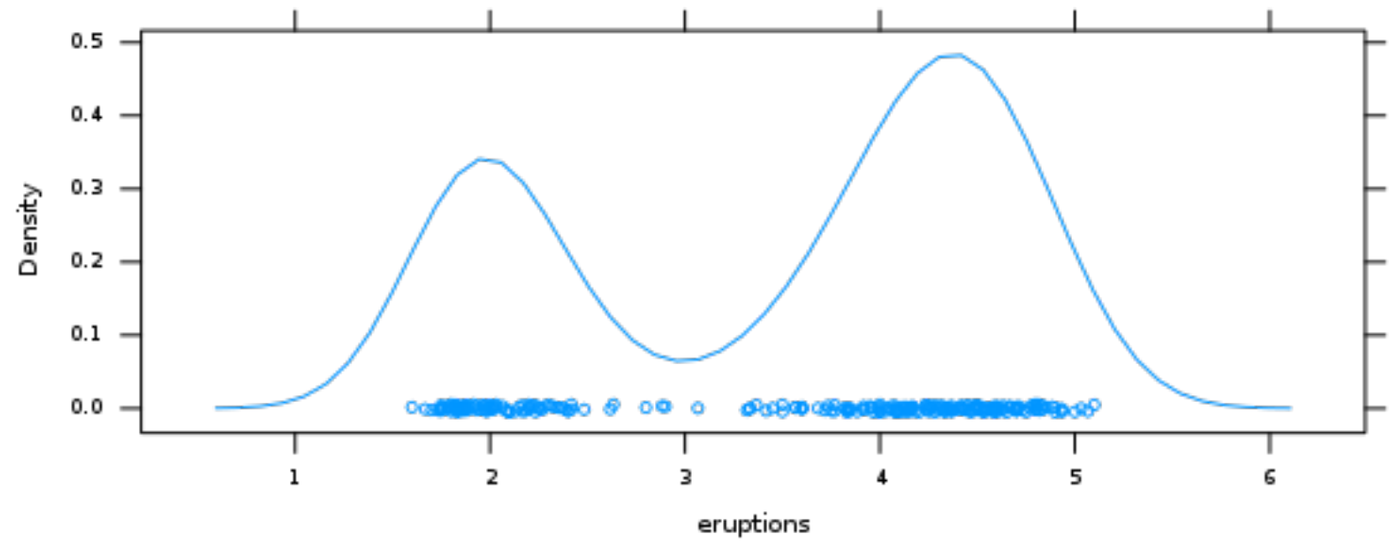
```
     n: the number of equally spaced points at which the density is
        to be estimated.  When 'n > 512', it is rounded up to a power
        of 2 during the calculations (as 'fft' is used) and the final
        result is interpolated by 'approx'.  So it almost always
        makes sense to specify 'n' as a power of two.
```

density() is the workhorse function that powers densityplot()

important arguments highlighted

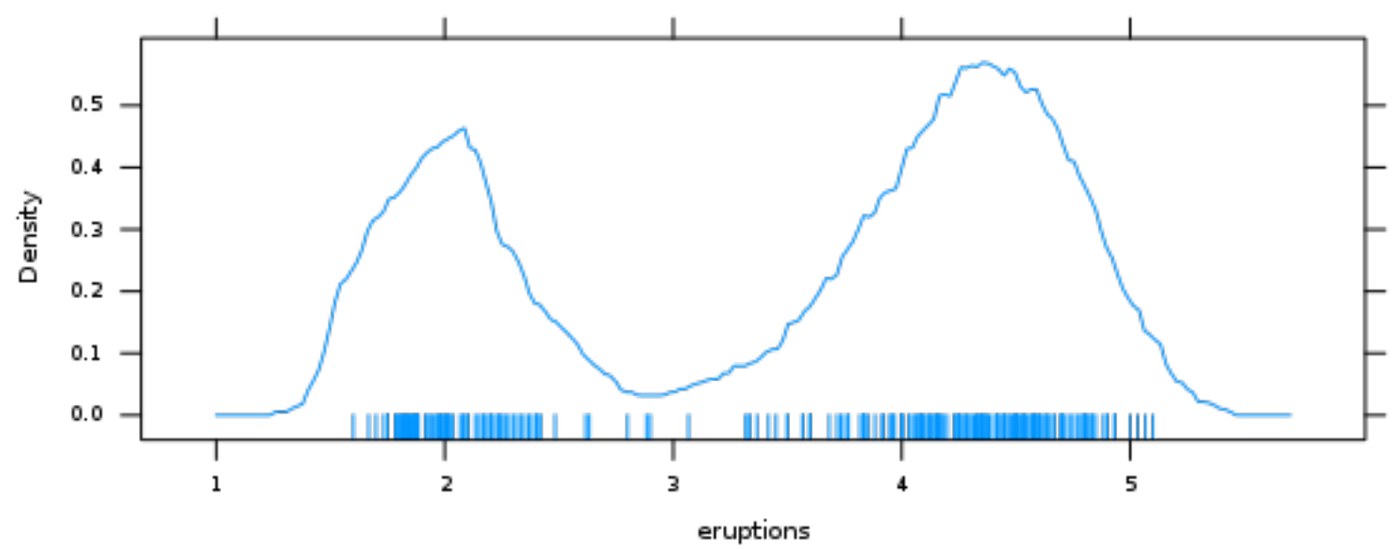
don't confuse yourself: you probably think 'n' means the size of your sample but density() has different ideas!

Figure 3.1

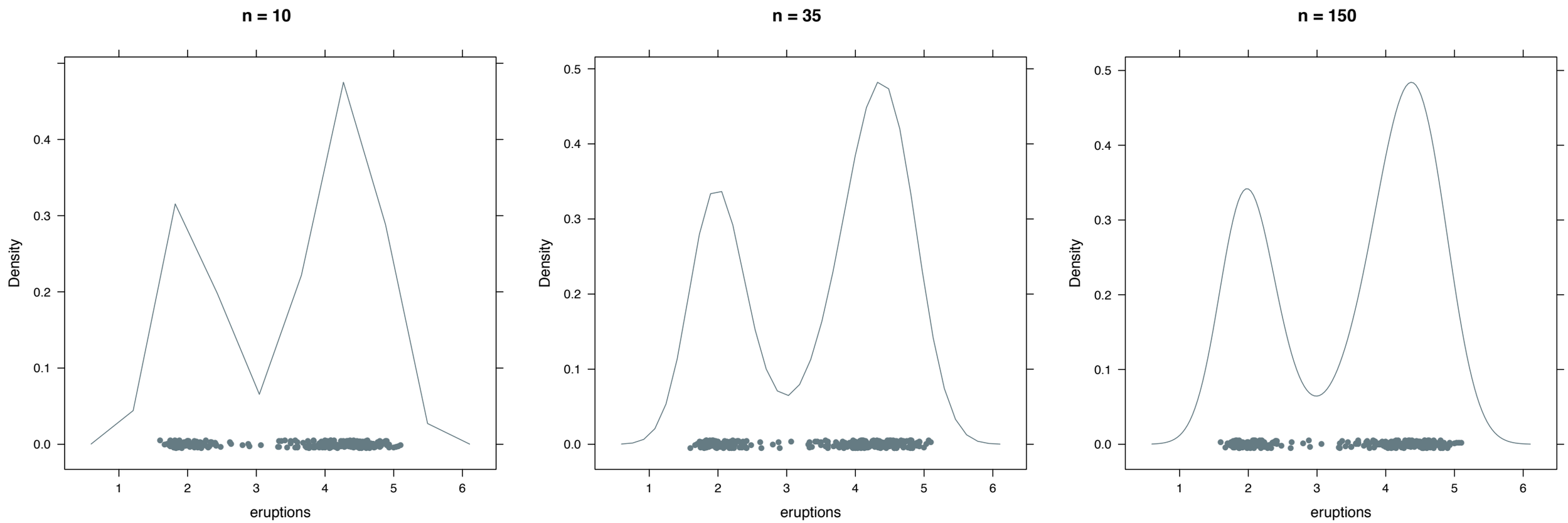



```
densityplot(~ eruptions, data = faithful)
densityplot(~ eruptions, data = faithful,
            kernel = "rect", bw = 0.2,
            plot.points = "rug", n = 200)
```

Figure 3.2



`densityplot` allows the user to specify the arguments to `density`, e.g. the kernel, bandwidth




```
densityplot(~ eruptions, data = faithful,
            n = 35,
            main = "n = 35")
```

```
density                 package:stats                 R Documentation

Kernel Density Estimation

<snip, snip>

      n: the number of equally spaced points at which the density is
         to be estimated.  When 'n > 512', it is rounded up to a power
         of 2 during the calculations (as 'fft' is used) and the final
         result is interpolated by 'approx'.  So it almost always
         makes sense to specify 'n' as a power of two.
```

Practical usage tip: if the kernel density estimate in your densityplot() isn't as smooth as you'd like, try specifying (a high) value of n.

Recommended sources:

Härdle, W. (1990) Smoothing Techniques With Implementation in S, Springer-Verlag, 1990. Sadly, not available via SpringerLink.

Silverman, B.W. (1986) Density Estimation for Statistics and Data Analysis, Chapman & Hall, 1986. Sadly, not available via STATSnetBASE.

Here's something that IS available via STATSnetBase (and seems to have been a source for this material in the first place!):

Chapter 8, Density Estimation: Erupting Geysers and Star Clusters from
A Handbook of Statistical Analyses Using R, Second Edition
Torsten Hothorn and Brian S . Everitt
Chapman and Hall/CRC 2009
Pages 139–159
Print ISBN: 978-1-4200-7933-3
eBook ISBN: 978-1-4200-7934-0
DOI: 10.1201/9781420079340.ch8

JB succeeded in getting as PDF!
Maybe this link will work for you (?):
http://www.crcnetbase.com/doi/pdfplus/10.1201/9781420079340.ch8
otherwise get on STATSnetBASE yourself and search and click ....