Statistics with R for Biologists

James H. Bullard
Kasper Daniel Hansen
Margaret Taub

Berkeley, California
July 7-11, 2008

Exploratory data analysis with R

EDA: Goals

Broadly, exploratory data analysis sets out to accomplish the following tasks: [1]

1. gain insight into a data set
2. find hidden structure
3. find the important variables
4. detect outliers
5. test assumptions
6. begin to develop modeling insights

[1] adapted from: here

EDA: History

In 1801 William Playfair, one of the founders of statistical graphics, had this to say:

> `For no study is less alluring or more dry and tedious than statistics, unless the mind and imagination are set to work or that the person studying is particualarly interested in the subject; which is seldom the case with young men in any rank of life.`

Playfair helped introduce graphics as a means of communicating information and did much to make statistics more alluring.

EDA

- Exploratory data analysis is at the heart of R programming!
- We need to look at our data both graphically and numerically before we can fit models and make tests.
- What common themes in data analysis can we abstract into functions?
- What type of summary statistics should we compute?

An example: viral load data

	viral.load	age	meds	infected
1	31.97	51.87	AZT	2.40
2	29.70	55.45	ABC	3.10
3	26.71	42.29	AZT	1.00
4	39.07	60.60	ABC	3.00
5	25.93	43.39	AZT	5.40
6	26.60	31.74	AZT	1.10

Table 1: Some viral load data

This lecture will focus on an exploratory analysis of a fabricated data set, which consists of a set of 1000 "observations" of four measured quantities:

- `viral.load` gives the measured viral load of an individual.
- age gives the age of the individual, `meds` gives their prescribed medication, and `infected` gives the number of days ago they were infected.

First steps: numerical summaries

- stem, summary, quantile
- mean, median, sd, cor, var, mad
- table : can use tables to perform grouping
- cut : cut vectors into buckets based on cutpoints
- split : partition data sets into groups based on group membership. Always returns a list.

```
> stem(rexp(10))

  The decimal point is at the |

  0 | 111689
  1 | 3
  2 | 058
```

Group means: split

Example

We want to compute group level summary statistics for each of the drug categories in our viral-load data set.

- What statistics should we compute?
- How can we assess correlation between columns?
- Test the `R` summary functions. What do we find?
- Make a table of the drug categories. Are certain drugs over-represented?
- Write a function which takes two arguments – a list of `data.frames` and a function – and computes a summary statistic on each `data.frame`. The *i*th element of the list will be the corresponding data from all subjects who took drug *i*. (lapply, and split will be useful.)

The R graphics system(s)

- There are two "graphics systems" in R: traditional and grid.
- First, we will focus on the "traditional" system.
  - The traditional system is contained in the package graphics.
  - The traditional system works by calling a high-level plotting function which sets up the plotting window.
  - To add to the plot you call additional functions, like lines, abline, points.

```
> x <- rnorm(100, 10)
> y <- 2 + 3 * x + rnorm(100)
> plot(x, y, pch = 16, cex = 0.3)
> abline(2, 3, col = "red")
```

Devices: In R

- Both R graphics systems are built on top of the grDevices package. Just like the package stats this package is loaded by default and you almost never encounter it directly.
- When you make a plot in R, it appears on an instance of the default device, which depends on your OS and how you are running R. Try `getOption("device")`.
- When you make a new plot, it will replace the plot currently displayed, unless you explicitly open a new plotting window, by calling the device.
- There are some ways to interact with the graphics device, e.g. through the function identify.

Devices: In R


```
> a <- 1:10
> b <- rnorm(10)
> names(a) <- LETTERS[1:10]
> plot(a, b)
> identify(a, b, labels = names(a))
> quartz()
> plot(b, a)
```

Devices: Saving R plots

- Besides printing to the default device, there are a variety of devices that you can plot to if you want to save your plots.
- We have the following: ps, pdf, jpeg, png, svg, bmp, wmf.
- When we call the dev.off below we turn the active device off.
- There are a variety of arguments to the device functions which can help control the formatting of your plots in the files you create.

```
> pdf("example%03d.pdf")
> hist(rnorm(1000), breaks = 200)
> dev.off()
```

Count data

We often encounter count or categorical data. Often these data need to be handled very differently from continuous data.

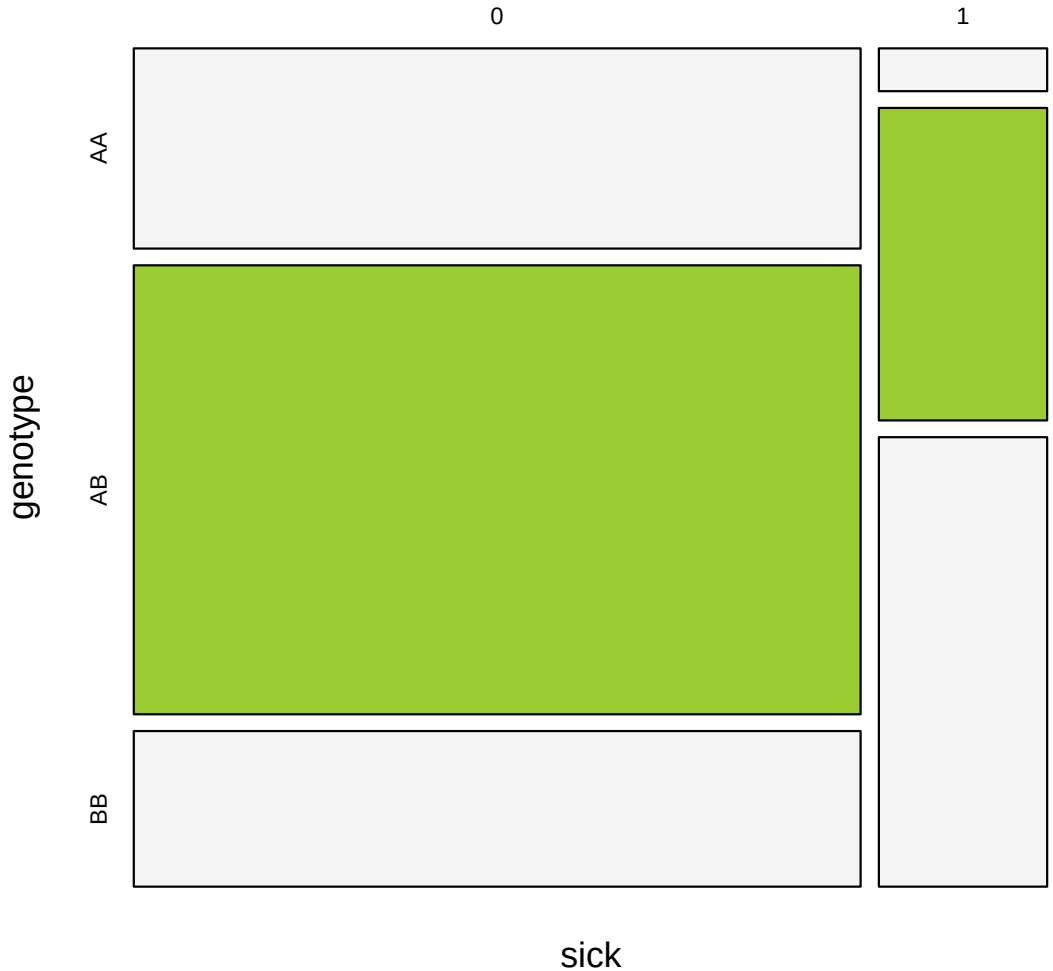
	AA	AB	BB
0	202	453	157
1	10	73	105

Table 2: Disease-status and genotype

- What are things we might want to do to count data?

```
> mosaicplot(table(df), main = NULL,
+      col = c("whitesmoke", "yellowgreen"))
```

Count data

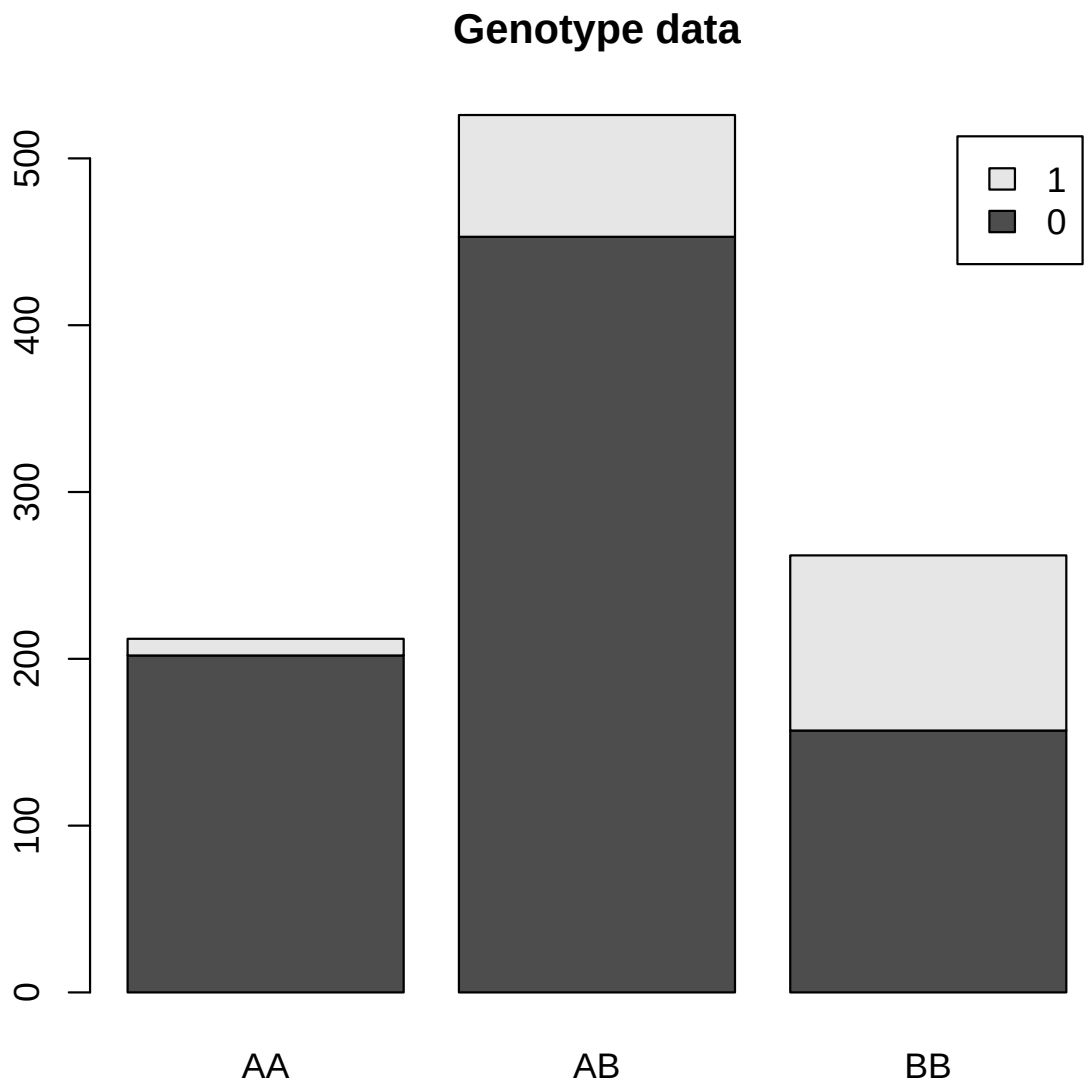


Plotting classics: Barplot

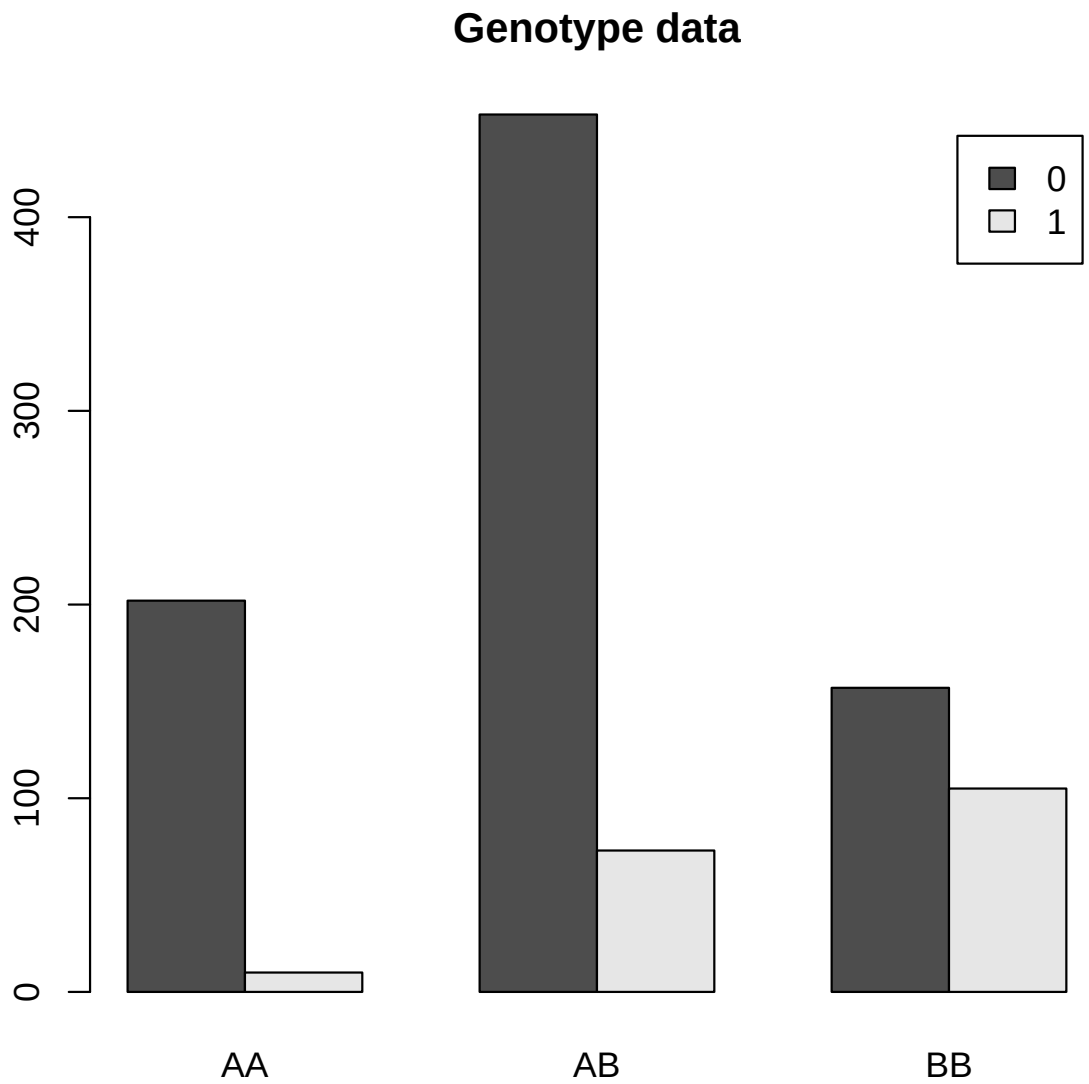
There are a couple of plots like the barplot and pie chart that we see all the time, and although there are often significantly "better" plots than these, it is important to see them in action. They are both good for plotting count data, although they can be used for other things as well.

```
> barplot(table(df), legend.text = rownames(table(df)
+     main = "Genotype data")
> barplot(table(df), legend.text = rownames(table(df)
+     main = "Genotype data", beside = TRUE)
```

Plotting classics: Barplot



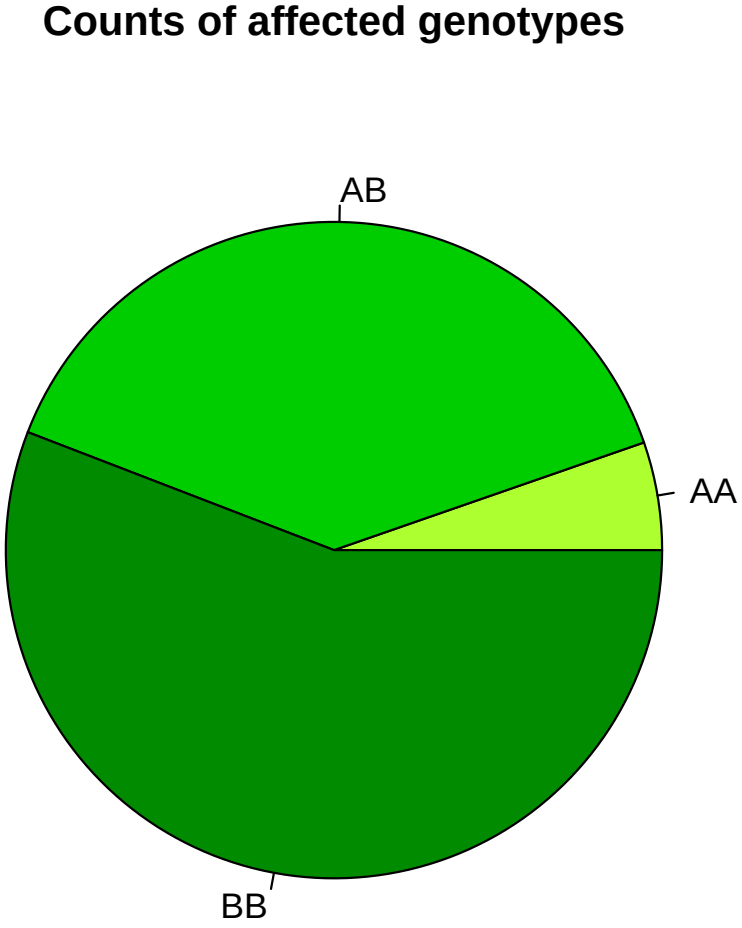
Plotting classics: Barplot



Plotting classics: Pie chart


```
> pie(table(df)[2, ], main = "Counts of affected geno
+     col = colors()[c(259, 257,
+         258)])
```

Plotting classics: Pie chart

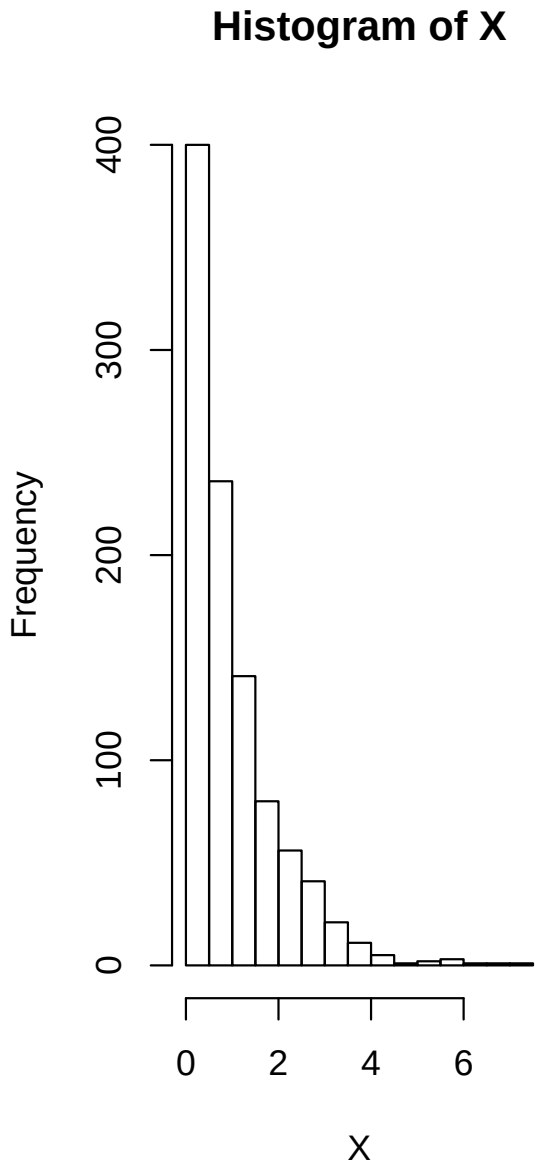


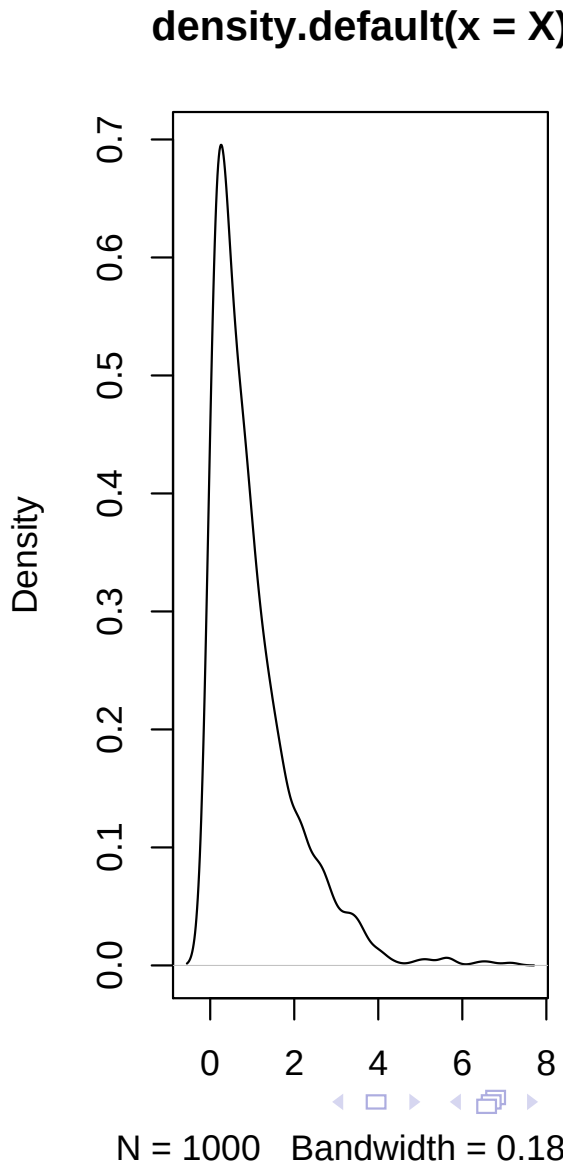
Changing graphics parameters: par

A very useful function when working with plots is the function par, which sets parameters which determine the how things are displayed in the graphics window. Many graphical parameters can be set directly with plotting functions, such as plot, however, sometimes it is only possible to use par.
One situation where we need to use par is when we want to put multiple plots on a single page/window.

```
> par(mfrow = c(1, 2))
> X <- rexp(1000, 1)
> hist(X)
> plot(density(X))
```

Changing graphics parameters: par





piechart and barplot

Example

- Make a plot with two "panels" containing a pie chart and a bar plot of the number of subjects in each medication category from the viral load data set.
- We can also use our drug category summary statistics computed in the previous example to make interesting pie charts and barplots. Try to make a barplot of the mean or median viral.load for each drug category. Is there anything interesting to report here?

Useful graphics functions

- plot : general plotting function in `R`
- hist : function for making histograms (important argument: `breaks`)
- qqplot, qqnorm : plots for comparing quantiles of two distributions
- matplot : function for plotting columns of matrices
- lines, points : draw lines and points on top of the active window
- curve : try: curve($x\hat{2}$), polygon : a little advanced - see the help
- par : set graphics parameters

The function plot

- Learning how to use the plot function in `R` could be a whole session of the course. Instead we'll learn just enough to accomplish basic tasks.
- The basic function takes an x and optionally a y as we have already seen: `plot(x = seq(-1,1,length = 100), y = dnorm(seq(-1,1,length=100)))`
- Plot annotations can be made using the arguments `xlab`, `ylab` and `main`, as well as many others.
- Lots of customization is possible: axes can be changed/removed, labels can be formatted in different ways, ranges for x and y values can be set, etc., either through direct arguments to plot or by using par.

plot methods

plot is an S3 generic function. We can see that plot is defined differently for different objects (try `methods(plot)`).

```
> plot(density(rnorm(100)))
> plot(ecdf(rnorm(100)))
```

These plotting functions have been specialized for both the `density`, and `ecdf` classes. What other classes have a specialized plot function? Some can be very fancy.

```
> x <- rnorm(100)
> y <- 2 * x^2 + rnorm(100)
> par(mfrow = c(2, 2))
> plot(lm(y ~ x))
> plot(viralLoad)
```

Boxplots

The function boxplot is a plotting method for generating Tukey's boxplots. Boxplots are good for:

- Comparing location shifts of distributions of data sets of varying size.
- Assessing skewness and spread of either of one or more distributions.
- Summarizing a distribution without the pitfalls of histograms: no bandwidth choice and less need to have large data sets.

Example

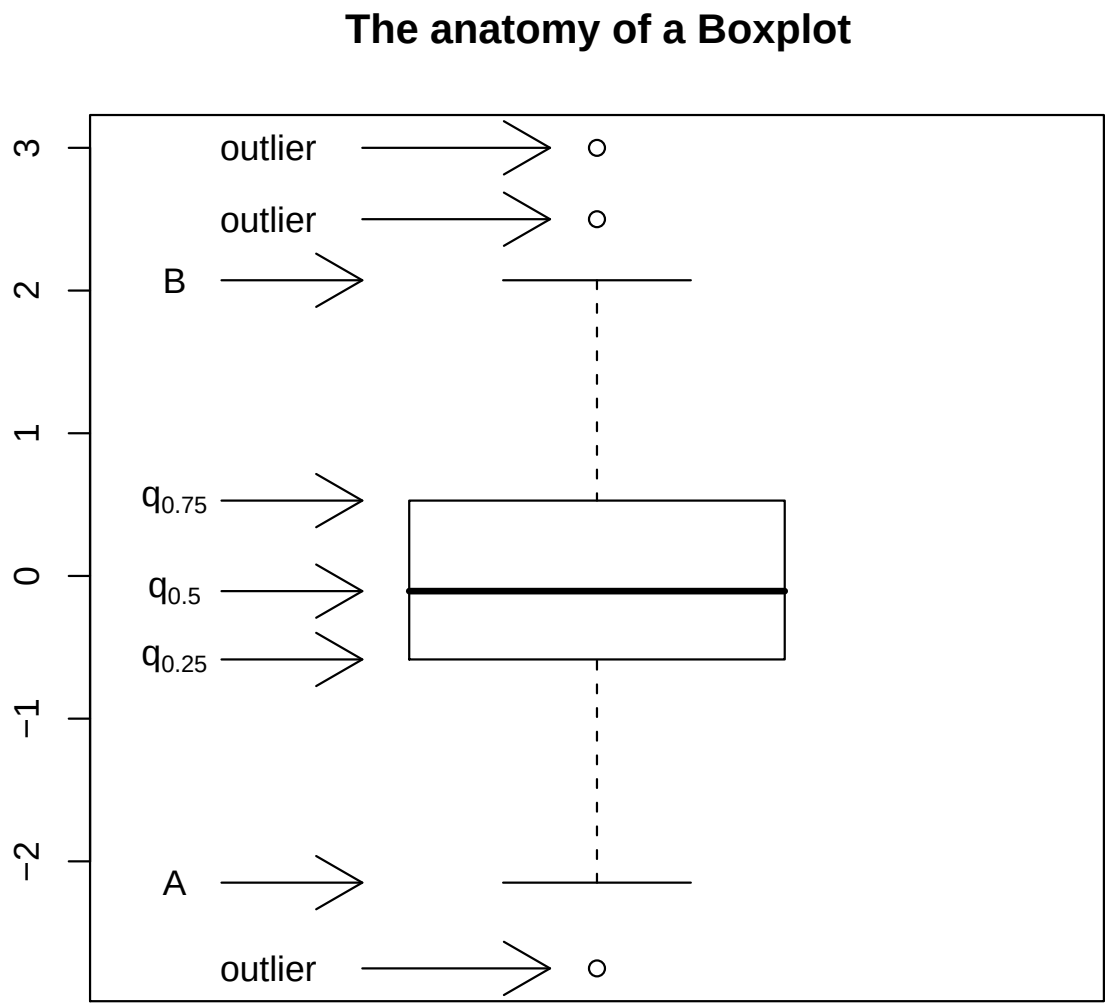
What does skewness look like on a boxplot? How about spread? Can we generate some data to exemplify these things? (Hint: Remember all of the random number generators which we talked about yesterday.)

Anatomy of a boxplot

- A, B : lower/upper adjacent values:

$$
\begin{aligned}
r &\triangleq |q_{75} - q_{25}| && (1)\\
A &= \inf\{x_i : x_i > q_{25} - 1.5r\} && (2)\\
B &= \sup\{x_i : x_i < q_{75} + 1.5r\} && (3)
\end{aligned}
$$

Anatomy of a boxplot

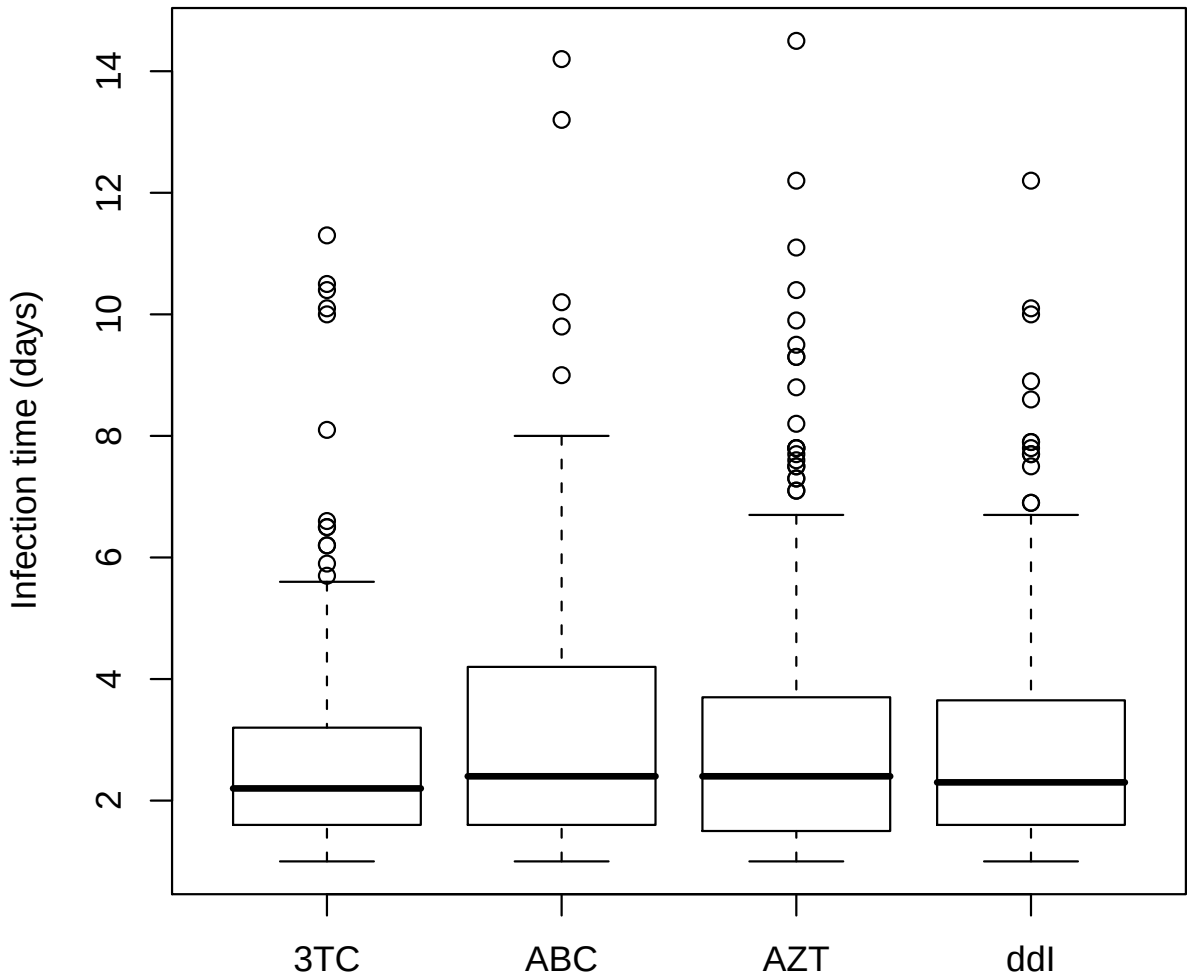


Parallel boxplots

To compare different data sets, we may want to plot more than one boxplot on a single plot. We can do this by creating parallel boxplots.

```
> boxplot(infected ~ meds, data = viralLoad,
+     ylab = "Infection time (days)")
```

Parallel boxplots



Formulas

We saw in the last example the use of the following construct:

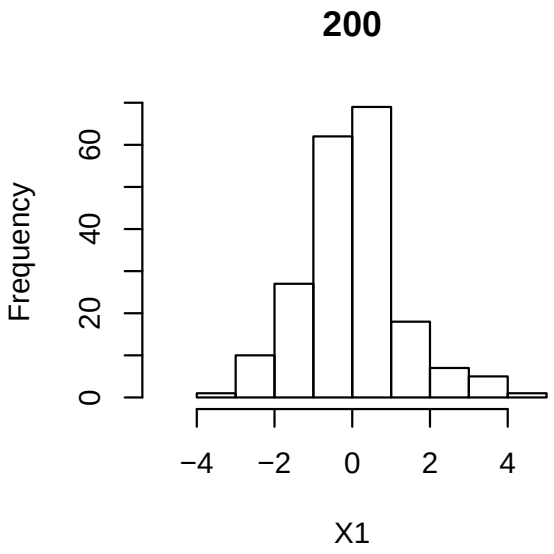
```
> infected ~ meds
> class(infected ~ meds)
```

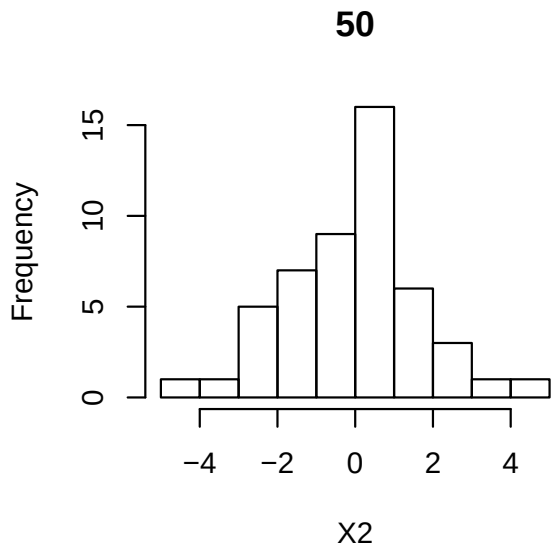
- This is a formula object in R. Formulas are ways of describing relationships between variables in R. A formula describes a model.matrix, which is essentially a design matrix in addition to the response variable; plot is specialized for the formula object.
- We will see lots more of these in the next section when we talk about statistical models in R.

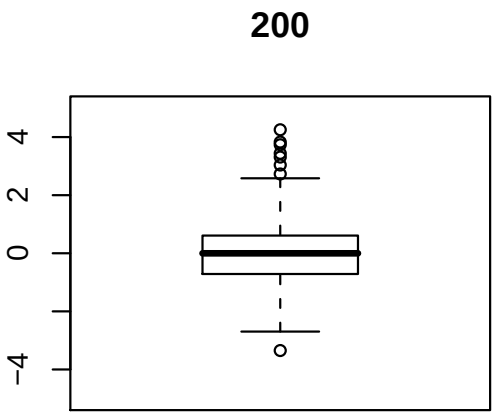
Boxplots vs histograms

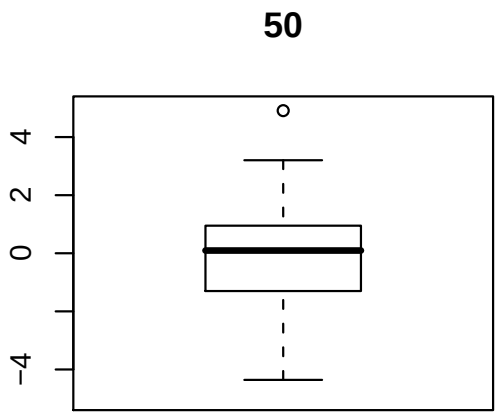
Below we have plotted two distributions using both boxplots and histograms. One distribution has 200 points and the other has 50 points. Which plotting method makes it easier to determine that they are the same? What kind of plot might even be better than either to compare two distributions?

Boxplots vs histograms

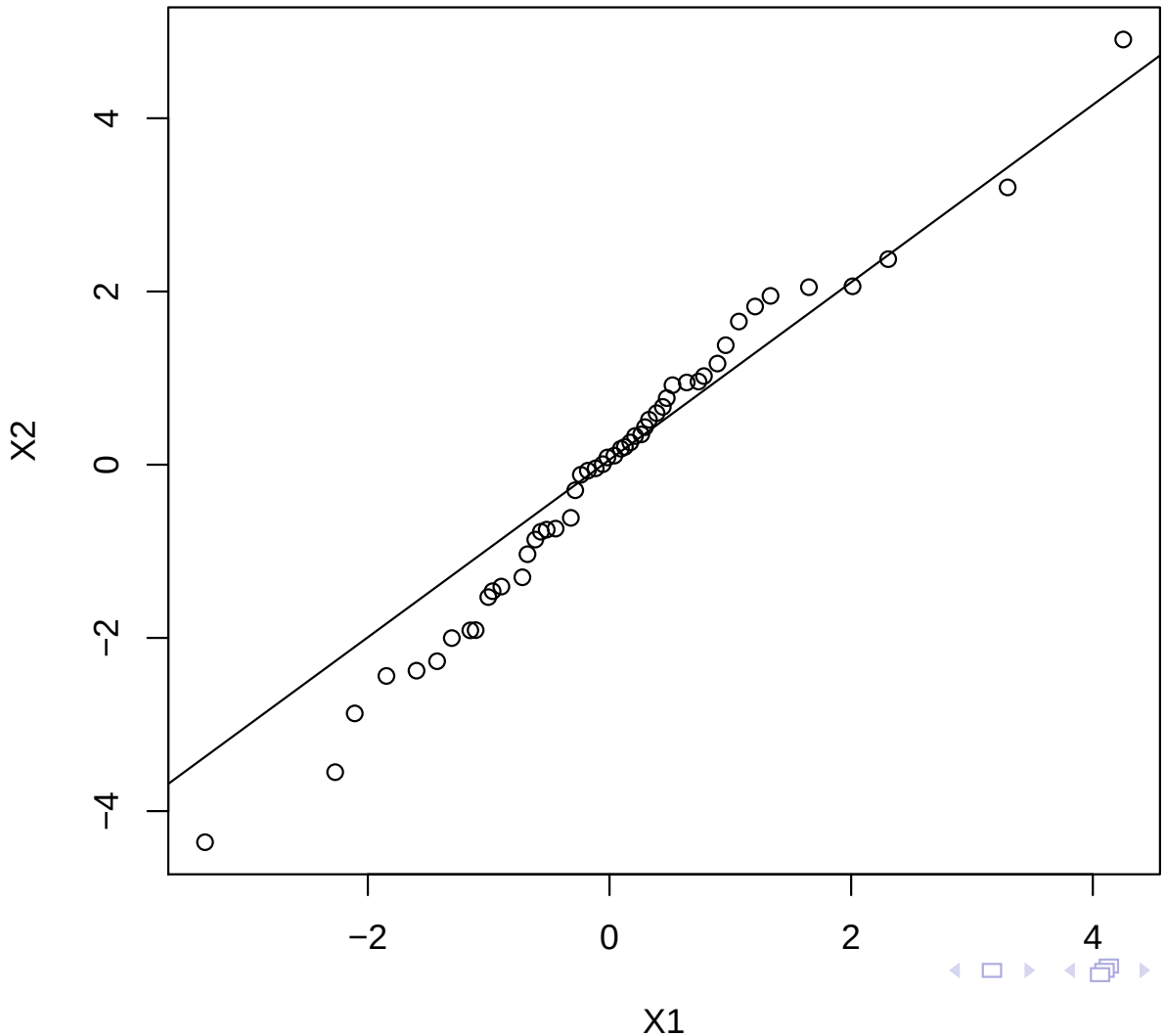








Remember from yesterday: qqplot, qqline



Bimodal distributions

- Imagine we have data with the following density:

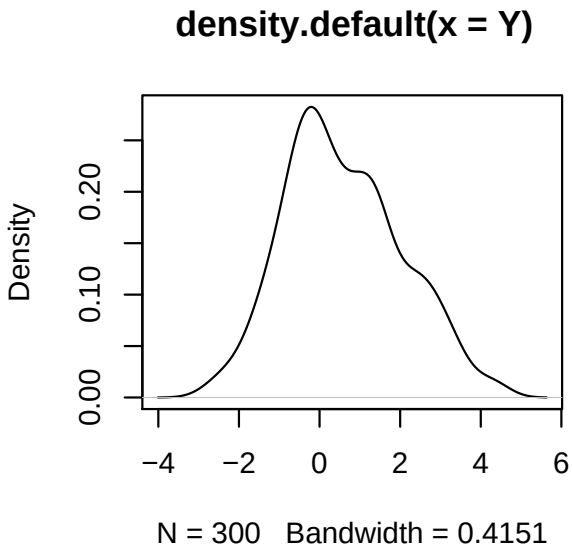
$$y = \pi\phi(\mu_1, \sigma_1) + (1 - \pi)\phi(\mu_2, \sigma_2) \tag{4}$$

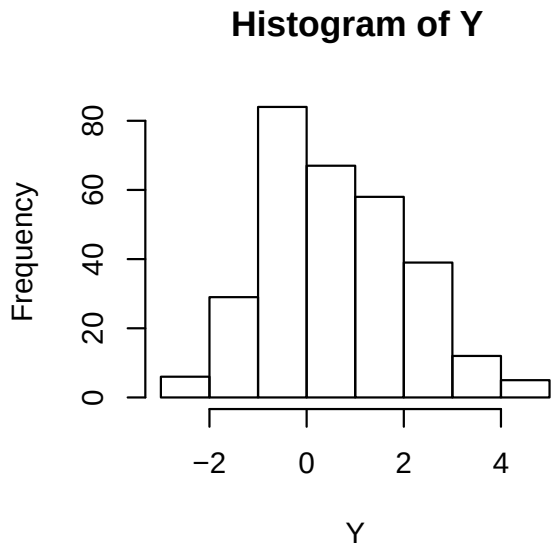
- y comes from a mixture of two normals! How can we see this graphically?
- In this case boxplot doesn't help that much, but density estimation and histogram do a better job. Be careful though - histograms can be misleading based on the number of breakpoints which we choose.

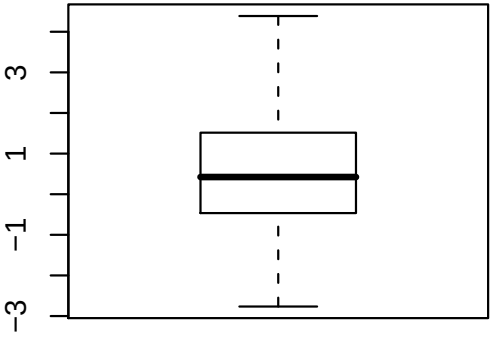
Bimodal distributions

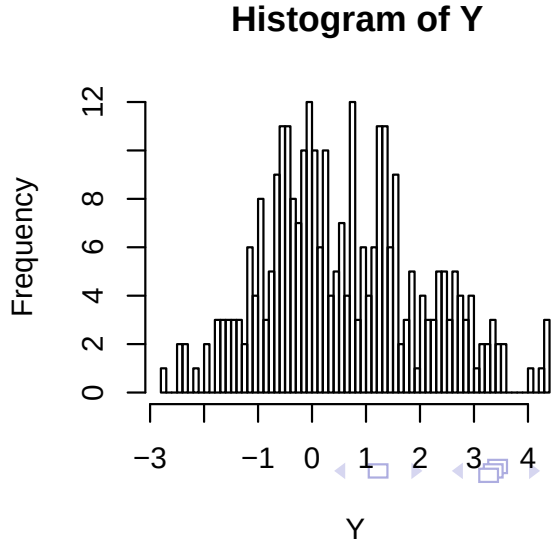

```
> N <- 300
> Y <- ifelse(rbinom(N, prob = 0.8,
+     size = 1), rnorm(N, 0, 1),
+     rnorm(N, 2.5, 1))
> par(mfrow = c(2, 2))
> plot(density(Y))
> hist(Y, breaks = 10)
> boxplot(Y)
> hist(Y, breaks = 100)
```

Bimodal distributions









Density estimation

- We have now seen the density a number of times and we should describe more carefully what the density function does.
- Essentially, kernel density estimation performs a weighted average of points using as a weighting scheme a particular kernel, often the normal kernel.

A kernel density estimator is defined as:

$$f_n(x) = \frac{1}{n} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \tag{5}$$

K is the kernel and must satisfy (**??**).

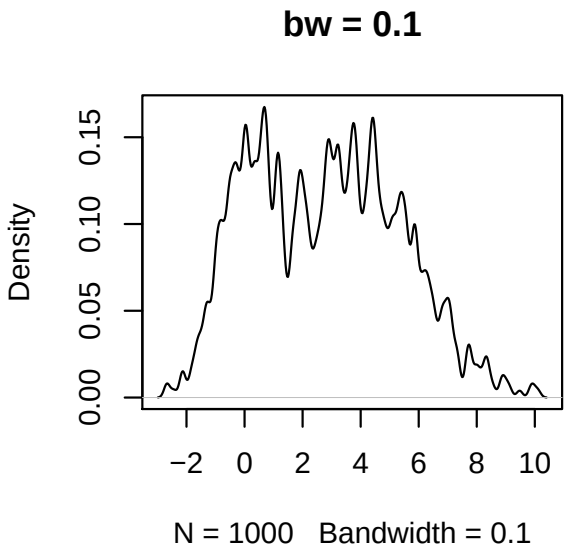
$$\int_{-\infty}^{-\infty} K(x)dx = 1 \tag{6}$$

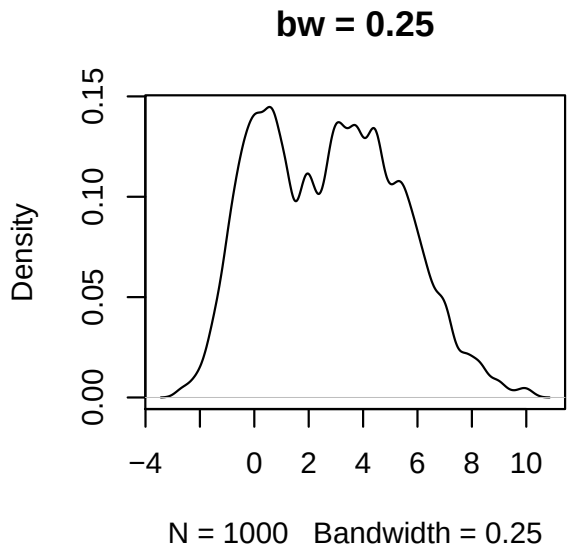
Density estimation

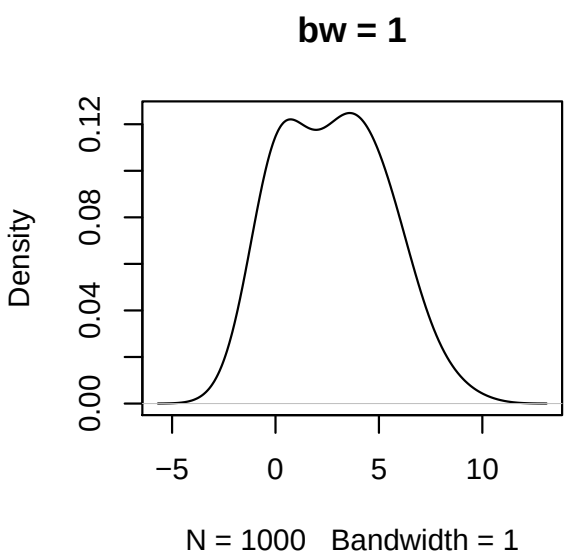
h is the bandwidth and determines the smoothness of the estimated function $f_n(x)$. We can visually show the effects of the choice of h.

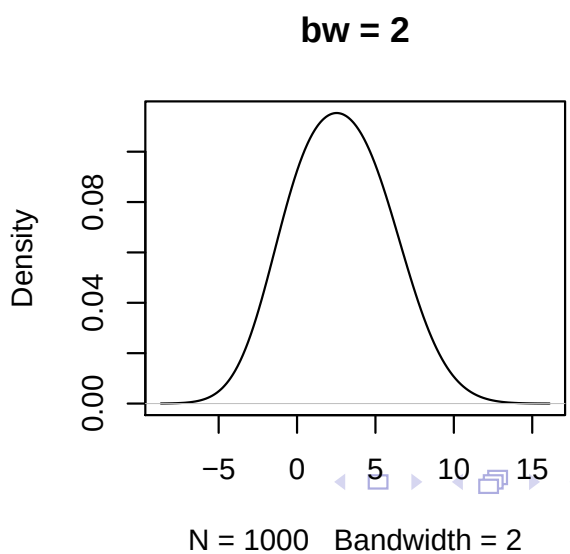
```
> N <- 1000
> X <- ifelse(rbinom(N, prob = 0.3,
+     size = 1), rnorm(N, 0, 1),
+     rnorm(N, 4, 2))
> par(mfrow = c(2, 2))
> g <- sapply(c(0.1, 0.25, 1, 2),
+     function(bw) {
+         plot(density(X, bw = bw),
+             main = paste("bw =",
+                 bw))
+     })
```

Density estimation









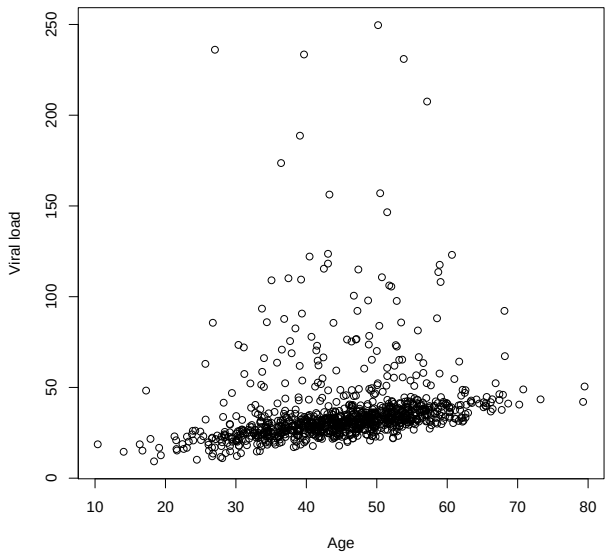
Comparing two variables: scatterplots

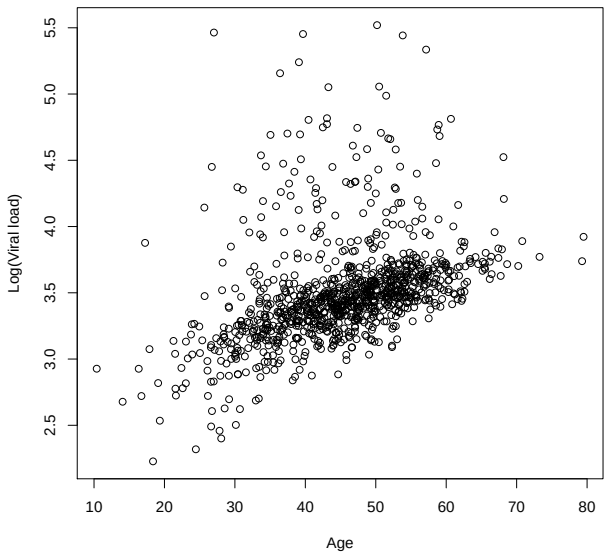
Before thinking about modeling, a visual inspection of the relationships between your data points can be very useful for identifying a number of bi-variate features of your data:

- Linear or non-linear relationships
- Outliers
- Skewness (which may make a data transformation useful)

```
> par(mfrow = c(1, 2))
> plot(x = viralLoad$age, y = viralLoad$viral.load,
+     xlab = "Age", ylab = "Viral load")
> plot(viralLoad$age, log(viralLoad$viral.load),
+     xlab = "Age", ylab = "Log(Viral load)")
```

Comparing two variables: scatterplots



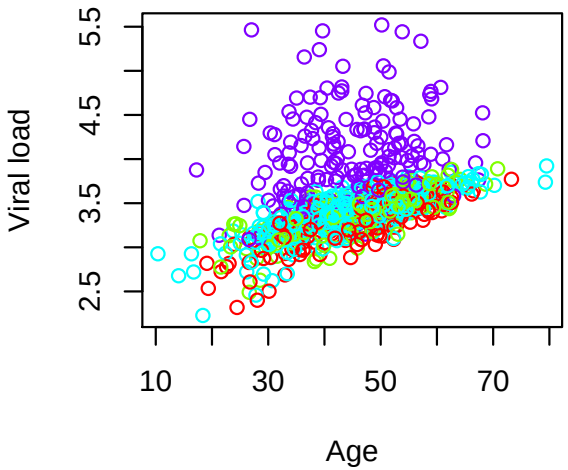


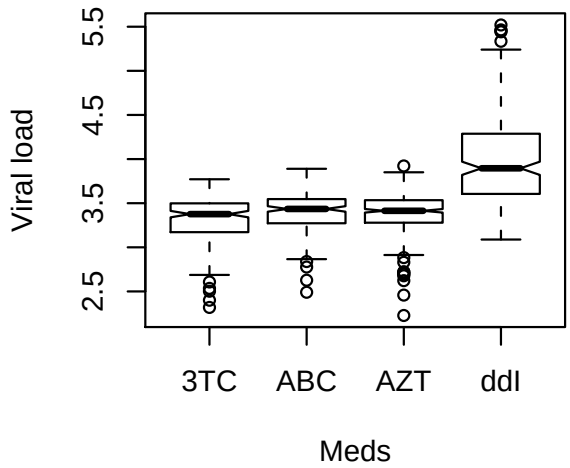
Working with the viral-load data set

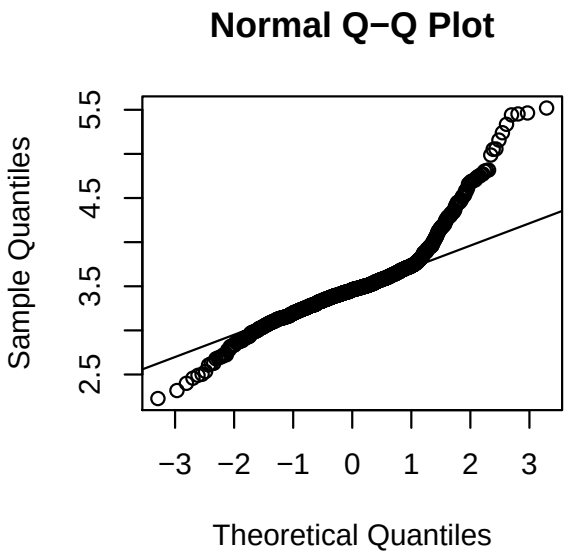
Example

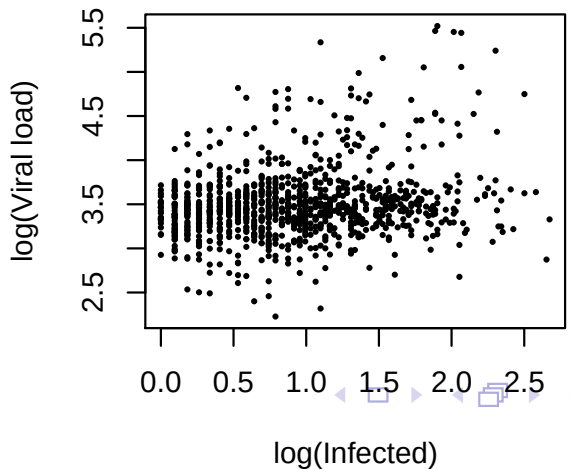
Make some plots of the viral-load data which help understand the relationships between drug type, duration of disease, and viral load (Hint: use different plotting symbols, colors).

Working with the viral-load data set









Indications from EDA

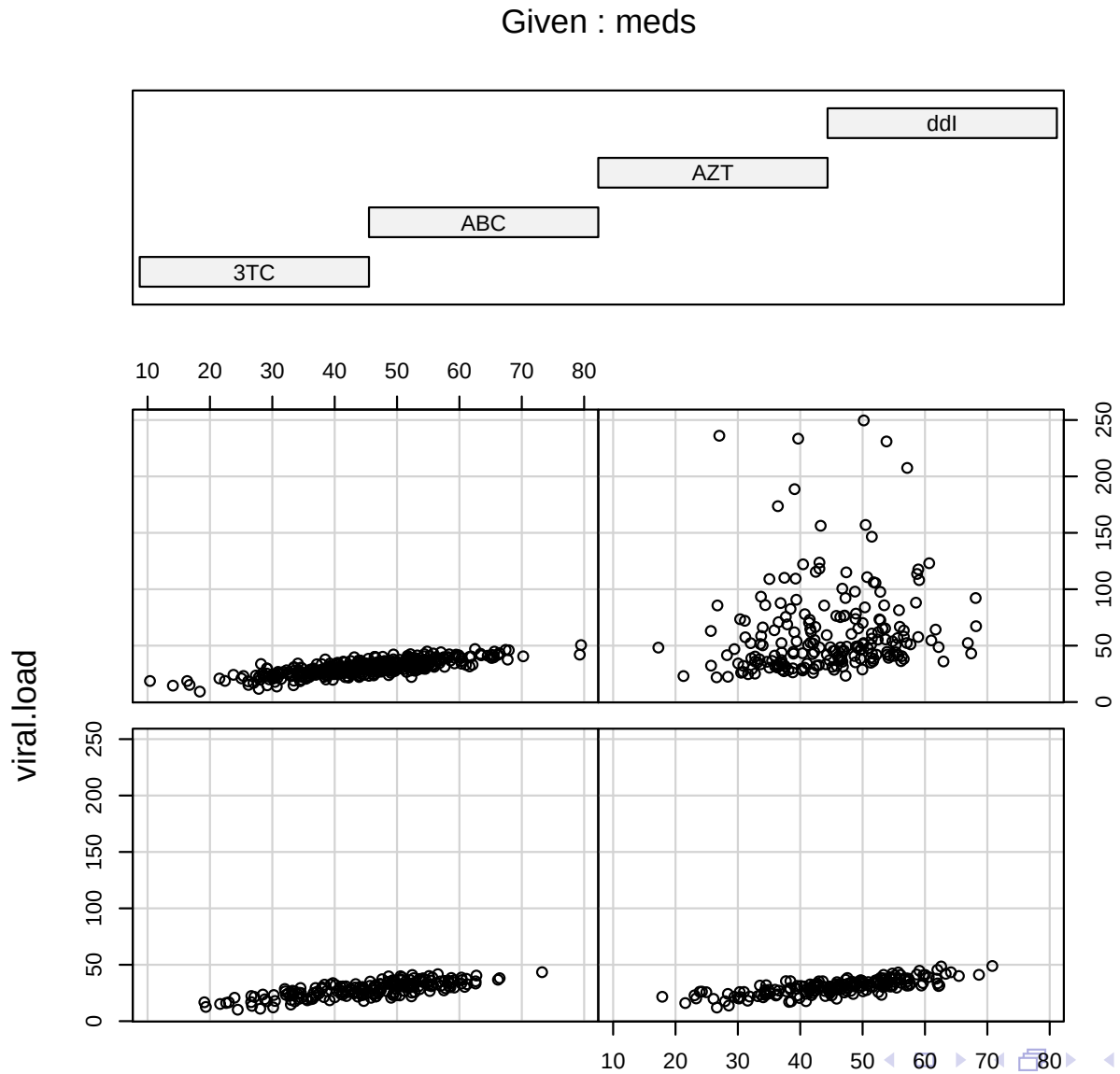
- What problems did you encounter presenting the viral load data set?
- Do you see some systematic effects in the data that might influence how we might fit a model?

coplots

- The function coplot makes a conditional plot, popularized by Cleveland.
- We can use the coplot to display the viral load data set.
- In addition, this is our first look at plotting functions from the grid graphics system, although you wouldn't know this just by looking at the plot!
- Can we make a coplot of the viral load data?

```
> coplot(viral.load ~ age | meds,
+     data = viralLoad)
> coplot(log(viral.load) ~ age |
+     infected, data = viralLoad)
```

coplots

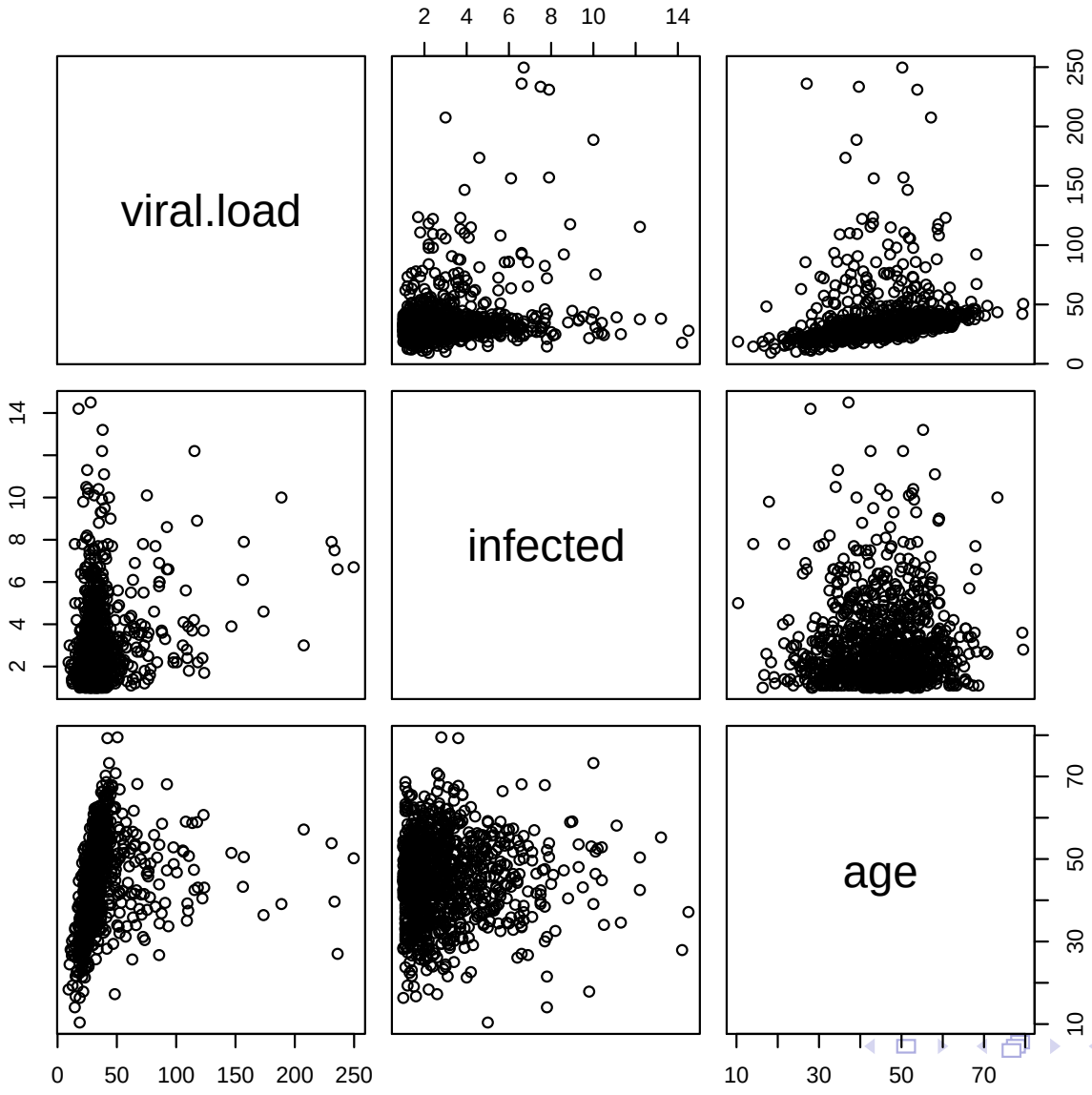

Given : meds
ddI
AZT
ABC
3TC
viral.load
10
20
30
40
50
60
70
80
0
50
100
150
200
250

Pairwise comparisions

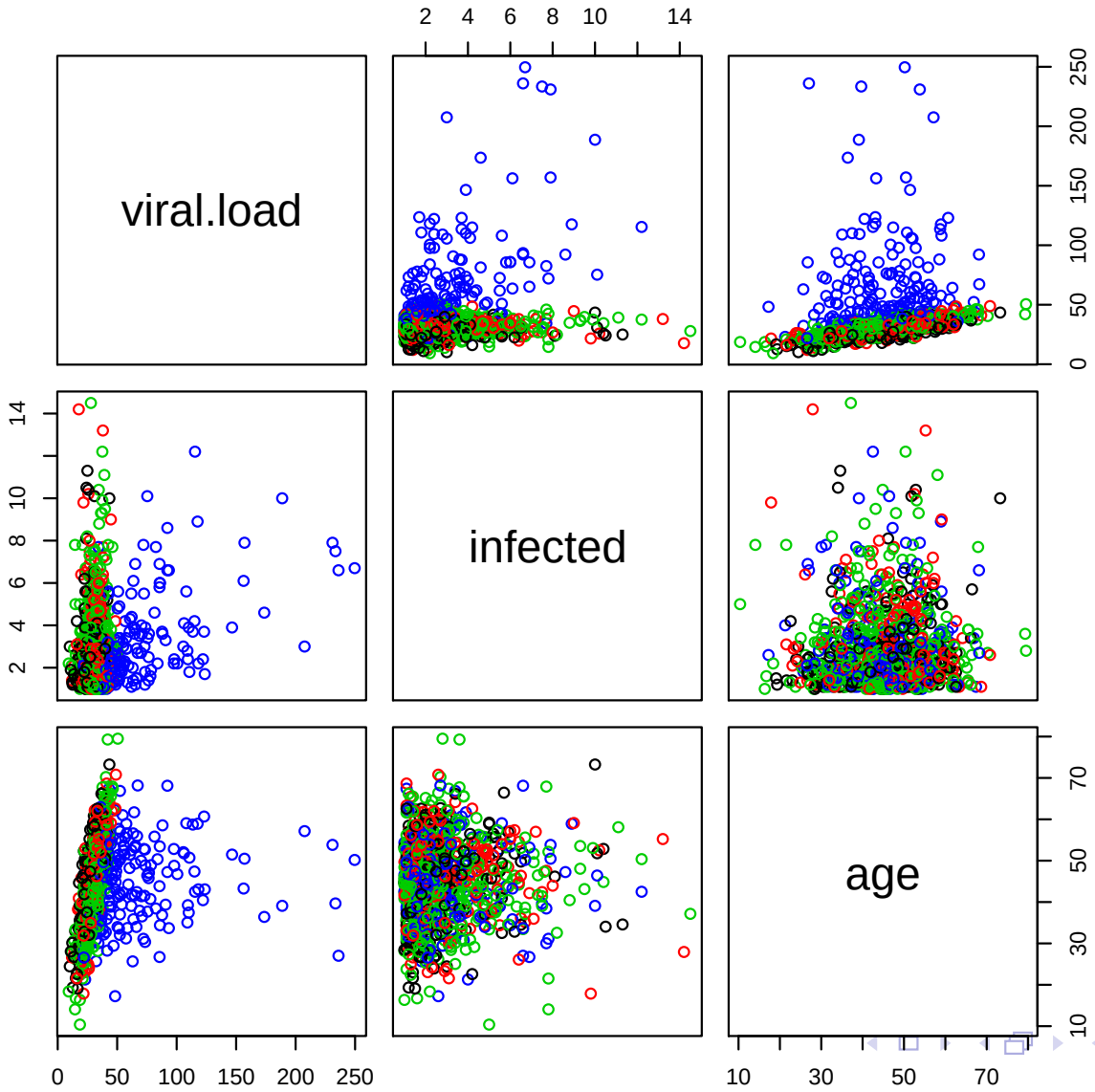
- Another important way of looking at "replicates" is via the pairs function.
- This function takes a matrix and constructs a grid of plots making all of the bivariate comparisons.
- Below we have made a pairs plot from the viral load data set comparing the three numeric variables.
  1. Add colors to the plot below to help see whether or not there are differences between medications.

```
> pairs(viralLoad[, c("viral.load",
+     "infected", "age")])
> pairs(viralLoad[, c("viral.load",
+     "infected", "age")], col = viralLoad$meds)
```

Pairwise comparisions



Adding some color



Smoothing

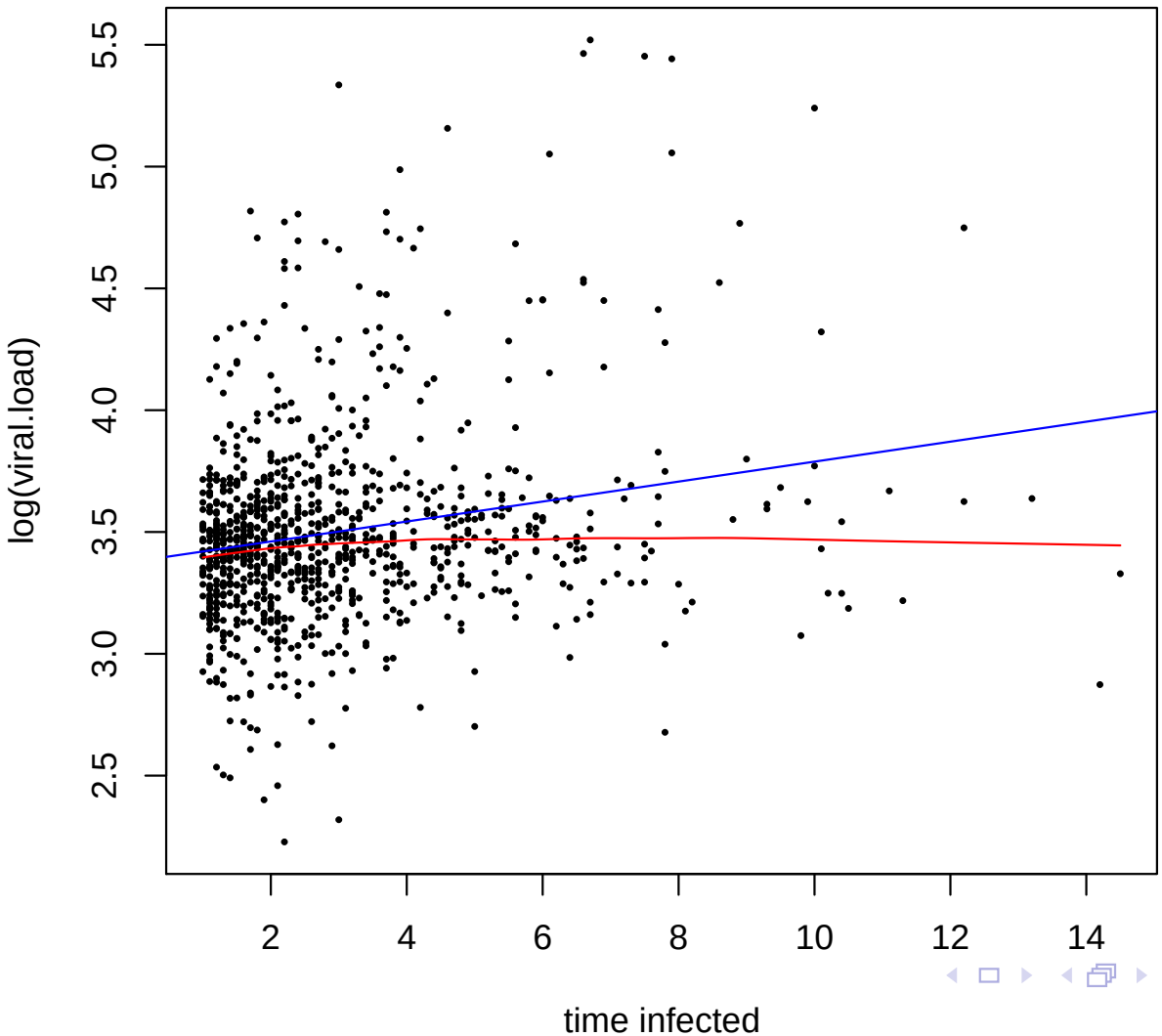
Often it is difficult to see a clear picture because we have too much data, or our eye is drawn to outliers. In these cases it is helpful to look at a "smoothed" versions of the data.

- lowess : lowess, or locally weighted polynomial regression.
- Often lowess helps pick out small trends in data which is not immediately obvious with other smoothed lines.
- Both lowess and loess are functions in R. lowess is older and takes as arguments an x and y, whereas *loess* uses formulas. Also, the defaults are different which can make a difference in the estimated line.

Smoothing


```
> plot(viralLoad$infected, log(viralLoad$viral.load),
+     pch = 16, cex = 0.3, xlab = "time infected",
+     ylab = "log(viral.load)")
> lines(lowess(viralLoad$infected,
+     log(viralLoad$viral.load)),
+     col = "red")
> abline(lm(I(log(viralLoad$viral.load)) ~
+     viralLoad$infected), col = "blue")
```

Smoothing



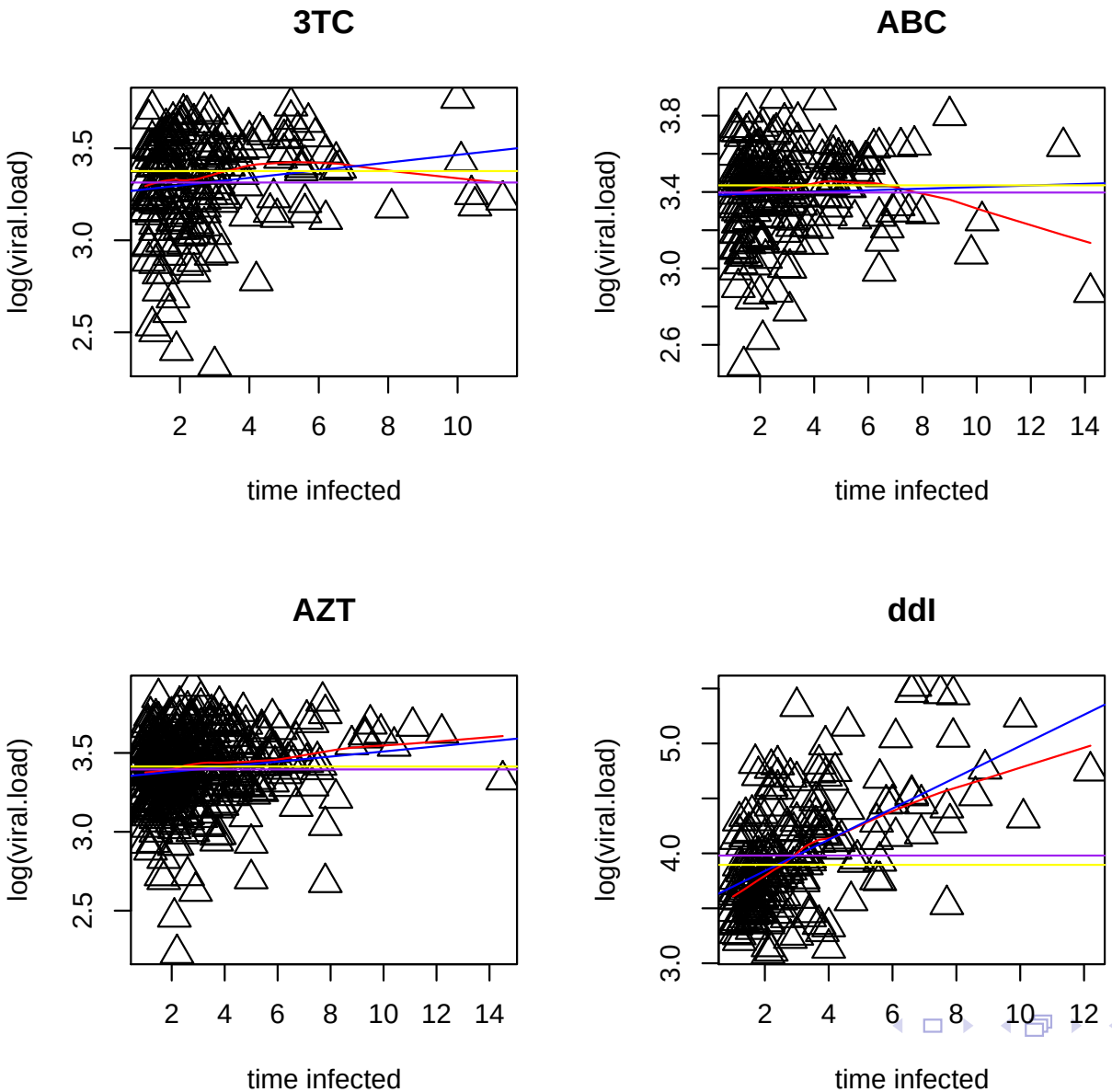
Comparing lowess

Example

In the last plot we saw a slight trend in the lowess line as we tend towards longer periods of infection. What do we see if we make the mean line? The regression line? The median line? Using the viral load data set do the following:

1. Plot the 4 lines in both different line types and different colors.
2. Make the same plots for the each of the different drug categories. Is anything surprising?

Comparing lowess



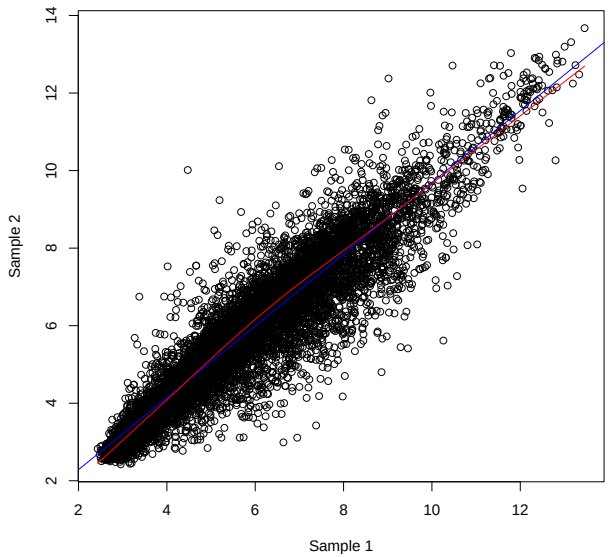
Mean-difference plots

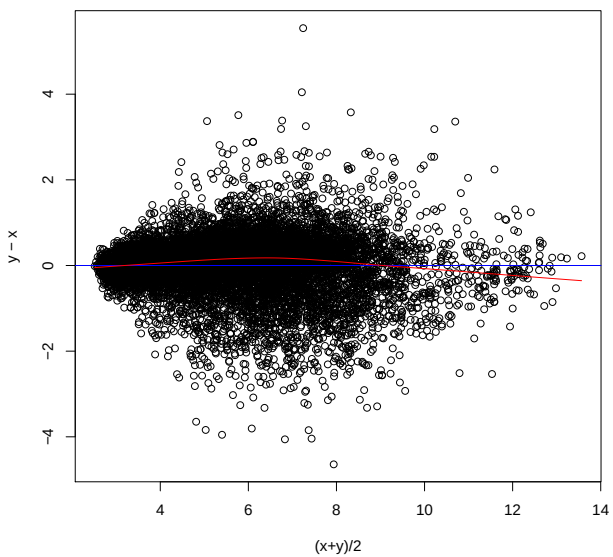
- Scatter plots help us determine the relationship between two variables x and y.
- Often however we want to highlight differences.
- After a normalization of two samples we are more interested in highlighting how different two samples x and y are.
- A mean difference plot generally plots the tuple: $((x + y)/2, y - x)$.

Mean-difference plots


```
> library(ALL)
> data(ALL)
> x <- exprs(ALL)[, 1]
> y <- exprs(ALL)[, 2]
> par(mfrow = c(1, 2))
> plot(x, y, xlab = "Sample 1", ylab = "Sample 2")
> abline(lm(y ~ x), col = "blue")
> lines(lowess(x, y), col = "red")
> plot((x + y)/2, y - x, xlab = "(x+y)/2",
+     ylab = "y - x")
> abline(0, 0, col = "blue")
> lines(lowess((x + y)/2, y - x),
+     col = "red")
```

Mean-difference plots

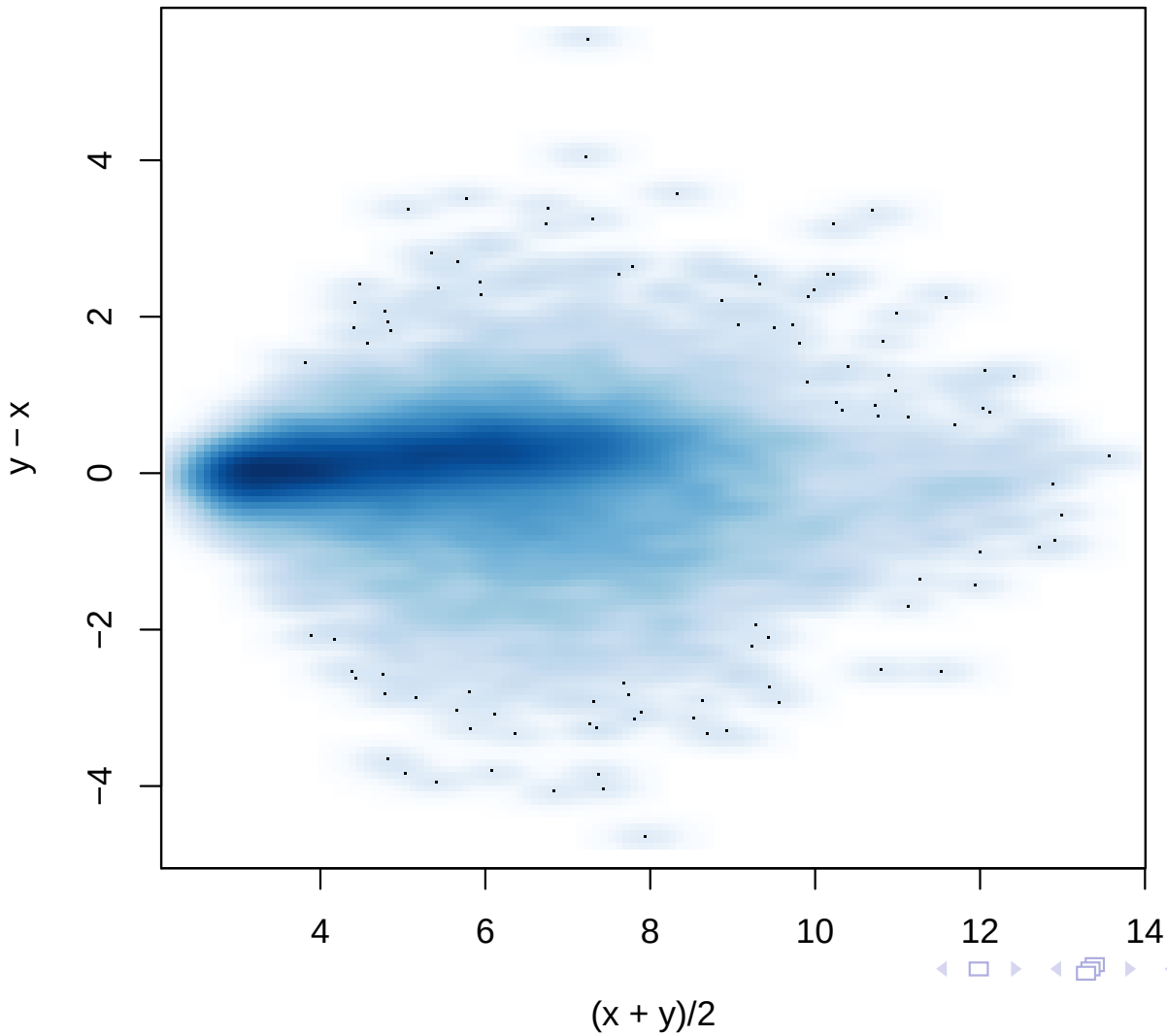




Enhanced scatter plots: smoothScatter


```
> library(geneplotter)
> smoothScatter((x + y)/2, y - x)
```

Enhanced scatter plots: smoothScatter

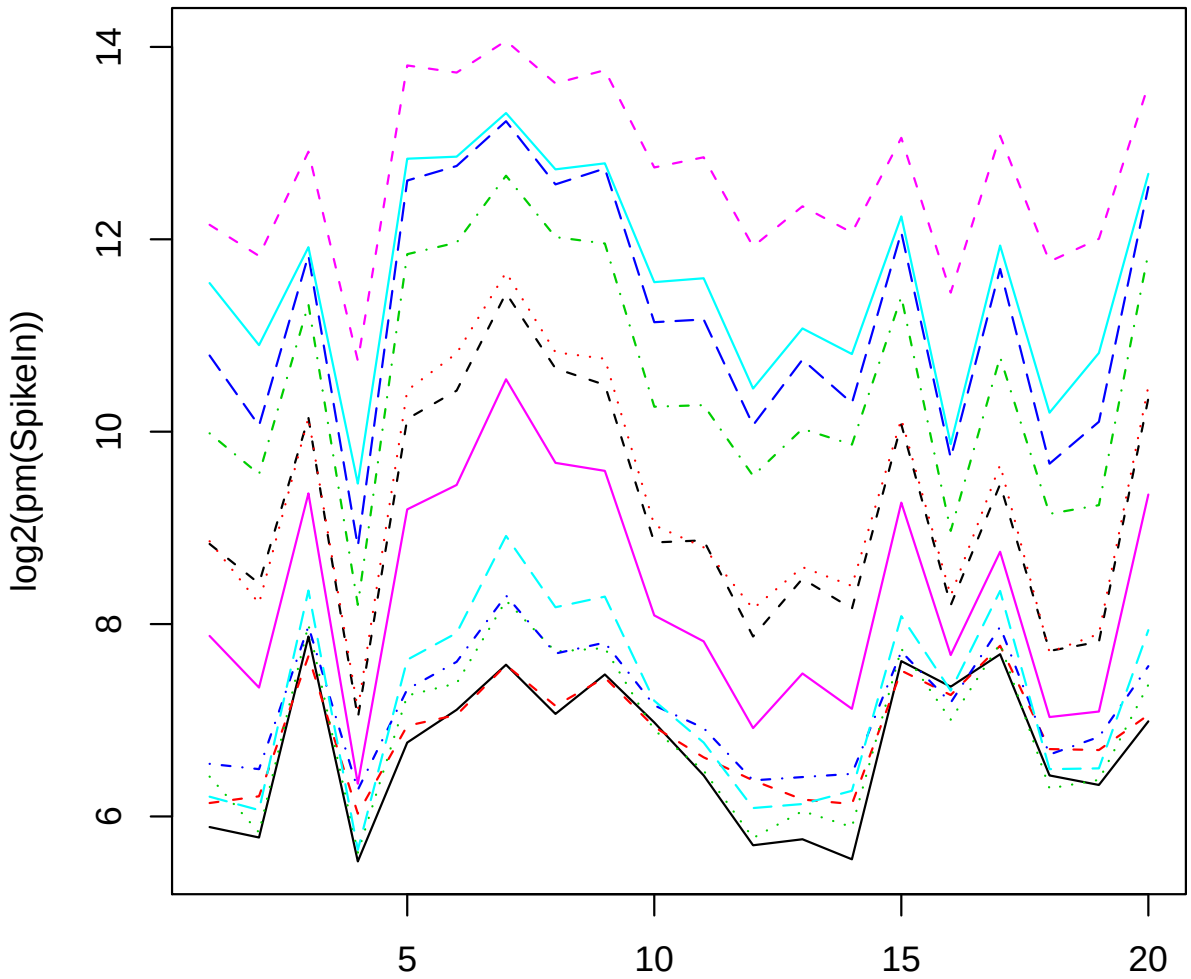


Plotting matrices

- We often have matrices where the columns are "repetitions" or "trials" and the rows are levels
- A good example is the SpikeIn data set in Affy. This data is data on a single probe set for 12 chips. We can use matplot to simplify the plotting of these data.

```
> library(affy)
> data(SpikeIn)
> matplot(log2(pm(SpikeIn)), type = "l")
```

Plotting matrices

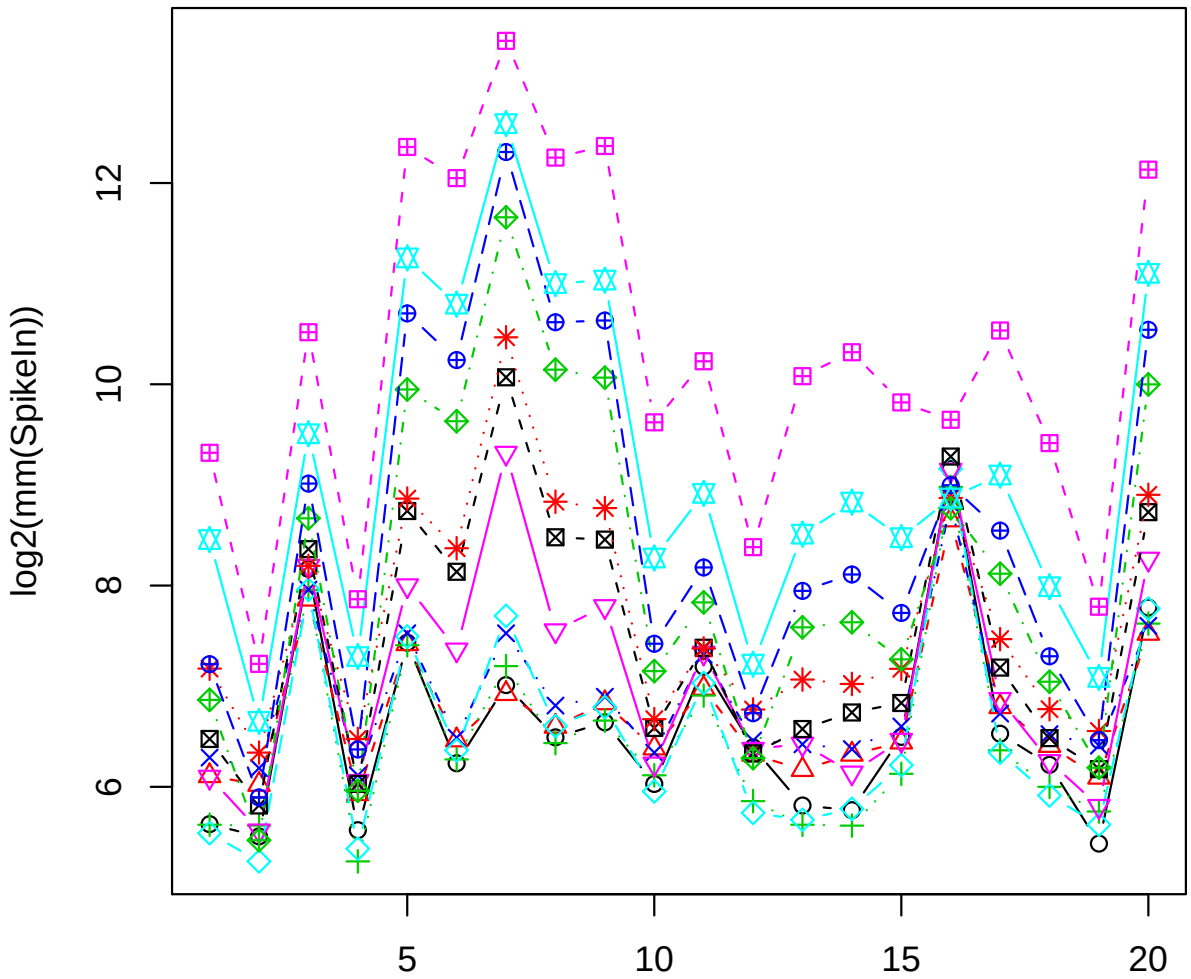


Line types

- In the previous example we used the "type = l" argument to draw a line instead of points, there are other such "type" specifiers - namely: 'p', 'l', 'b', 'o', 'h', 's'.
- We have the "lty" or line type argument which specifies the line type.
- We have the "pch" argument which represents the plotting symbol.
- By looking at `help(par)` you can find out a lot about these graphics parameters and their possible values.
- Lets try some of the others out!

```
> matplot(log2(mm(SpikeIn)), type = "b",
+     pch = 1:ncol(mm(SpikeIn)))
```

Line types

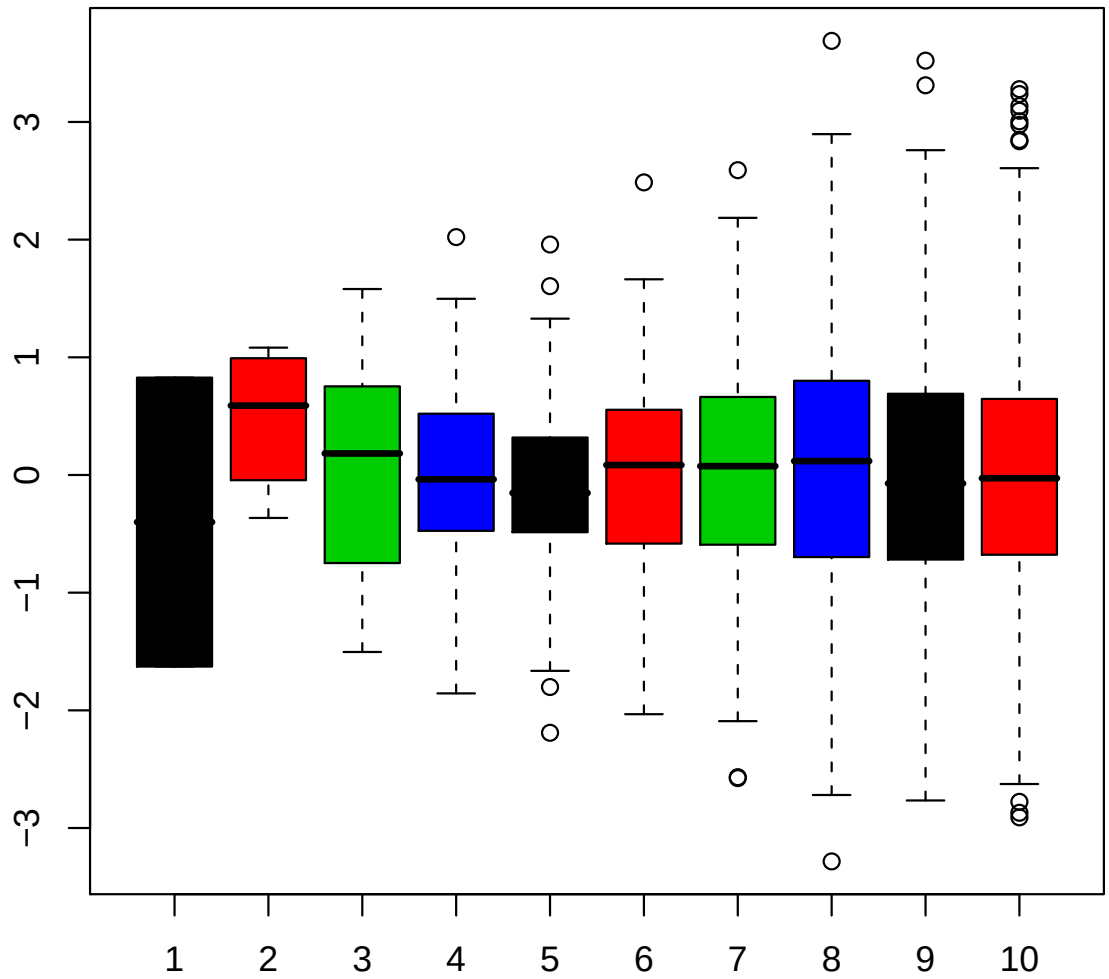


Plotting arguments

- In the previous example we did `pch = 1:ncol(mm(SpikeIn))`, that is we specified a symbol for each of the `ncol(mm(SpikeIn))` chips. Try just `pch = 1:4`
- What happens?
- When we specify plotting arguments we have to realize that they are also recycled. Sometimes this is what you want, other times not!
- In the boxplot below you can use color to indicate a shared covariate between boxplots of the same color.

```
> boxplot(lapply(1:10, function(i) {
+     rnorm(2^i)
+ }), col = 1:4)
```

Plotting arguments



Spatial data

	x	y	intensity
1	1	1	0.06
2	2	1	0.60
3	3	1	0.13
4	4	1	1.29
5	5	1	0.73
6	6	1	0.43

Table 3: Some intensity by location data

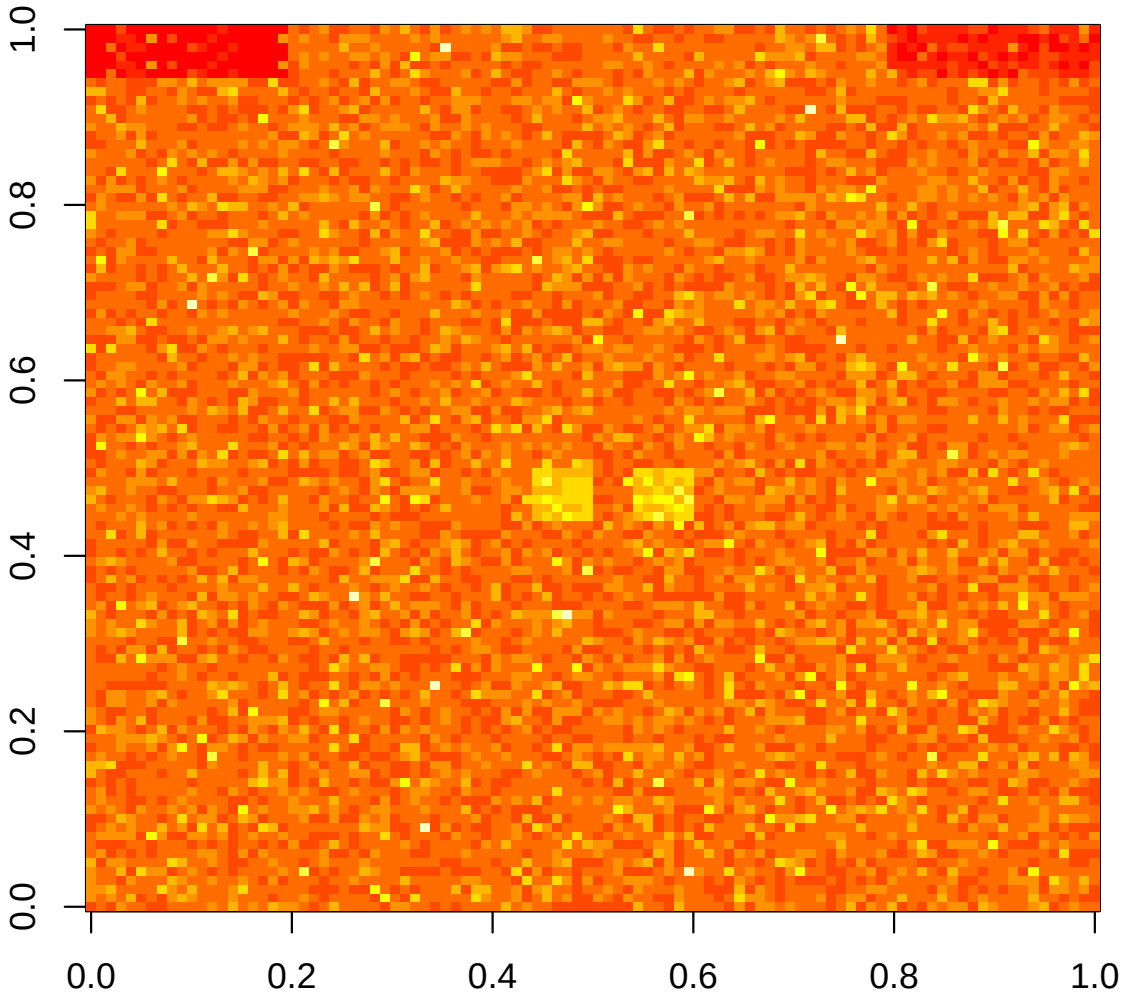
- How do we deal with spatial data?
- What types of plots are relevant here?

Spatial data

- We want to return to our spatial data example; this is an example of trivariate data or the tuple (x, y, z) where z represents an intensity and x and y represent points on the grid. The traditional way that these have been presented is using the image function.
- image when you have a discrete space and you are looking for trends in the data or anomalies as we see in the image below they jump right out!

```
> image(matrix(spatial[, 3], nrow = 100),
+       col = heat.colors(10))
```

Spatial data



More higher dimensional plotting functions

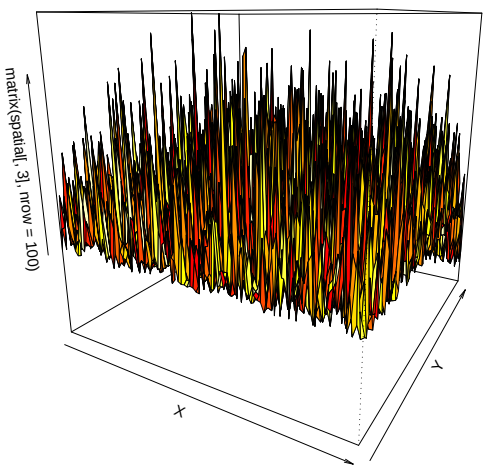
- In addition to image we have persp, contour

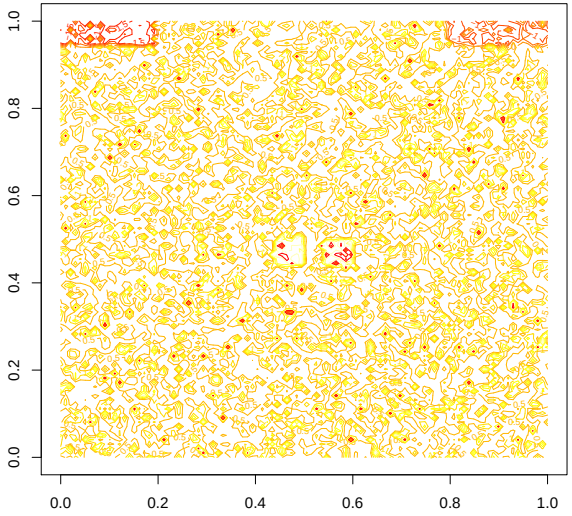
Example

Make a couple of plots using persp and contour of the spatial data. The data is located in "data/spatial.dta" - First, it will be helpful to have a look at the examples and understand what type of objects these functions take.

```
> par(mfrow = c(1, 2))
> persp(z = matrix(spatial[, 3],
+     nrow = 100), col = heat.colors(10),
+     theta = 30)
> contour(matrix(spatial[, 3], nrow = 100),
+     col = heat.colors(10), nlevels = 20)
```

More higher dimensional plotting functions





Other tools and techniques

- Hexbin : A Bioconductor function to make two dimensional density plots where we divide the grid into hexagonal bins and use a color or gray scale to portray density in the bin.
- Running Means : A simple technique where nearby values are averaged together to produce a "local" average.
- smooth : To perform different types of running median smoothing.

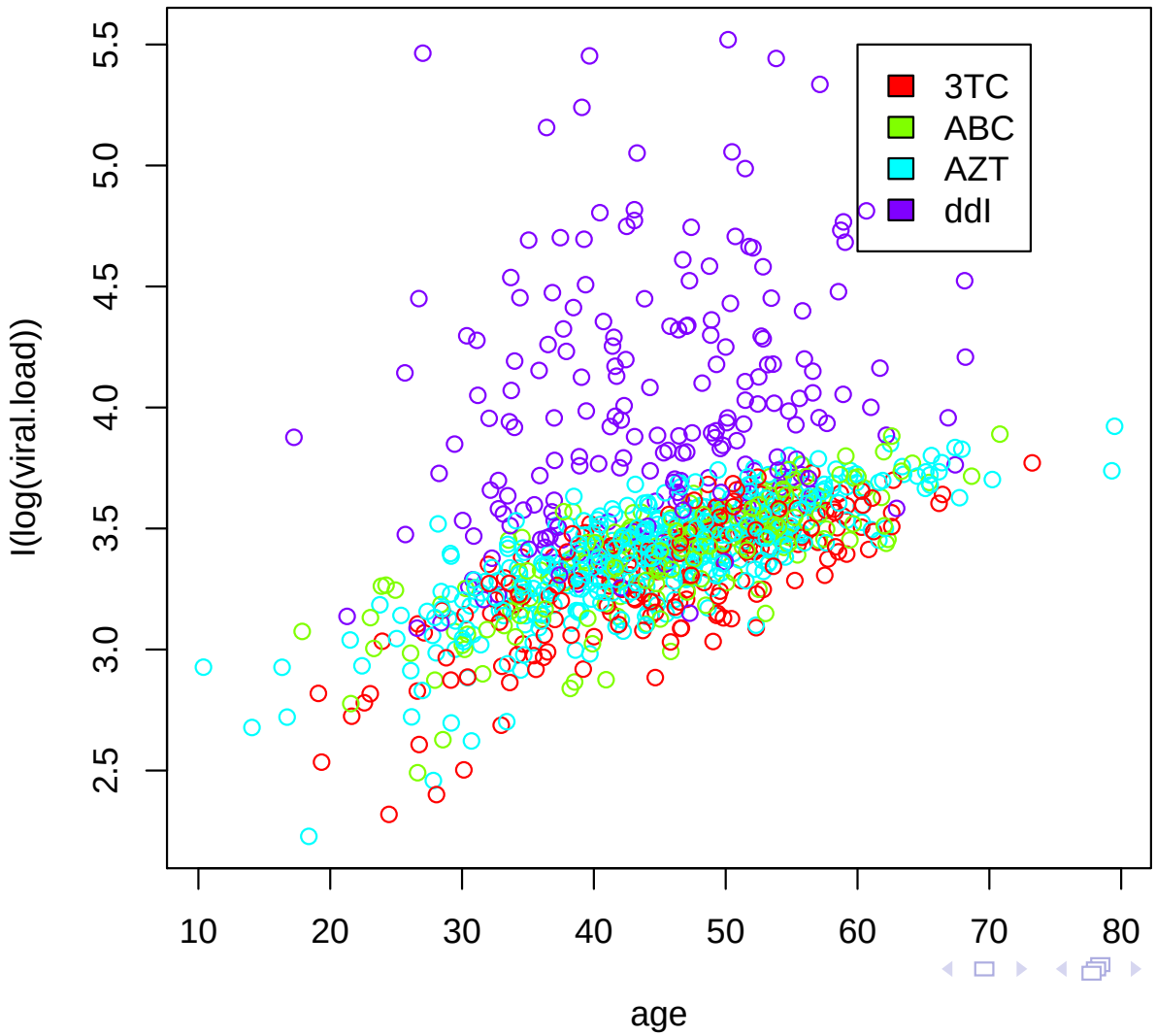
Legends

In order to most clearly present ideas we need to be able to annotate plots and choose colors, plotting symbols, line types, etc. There are lots of R functions to do this and we'll start off with the basics.

- We use the legend function to add legends to plots

```
> colors <- rainbow(4)[as.numeric(viralLoad$meds)]
> plot(I(log(viral.load)) ~ age,
+      data = viralLoad, col = colors)
> legend(60, 5.5, legend = levels(viralLoad$meds),
+      fill = rainbow(4))
```

Legends

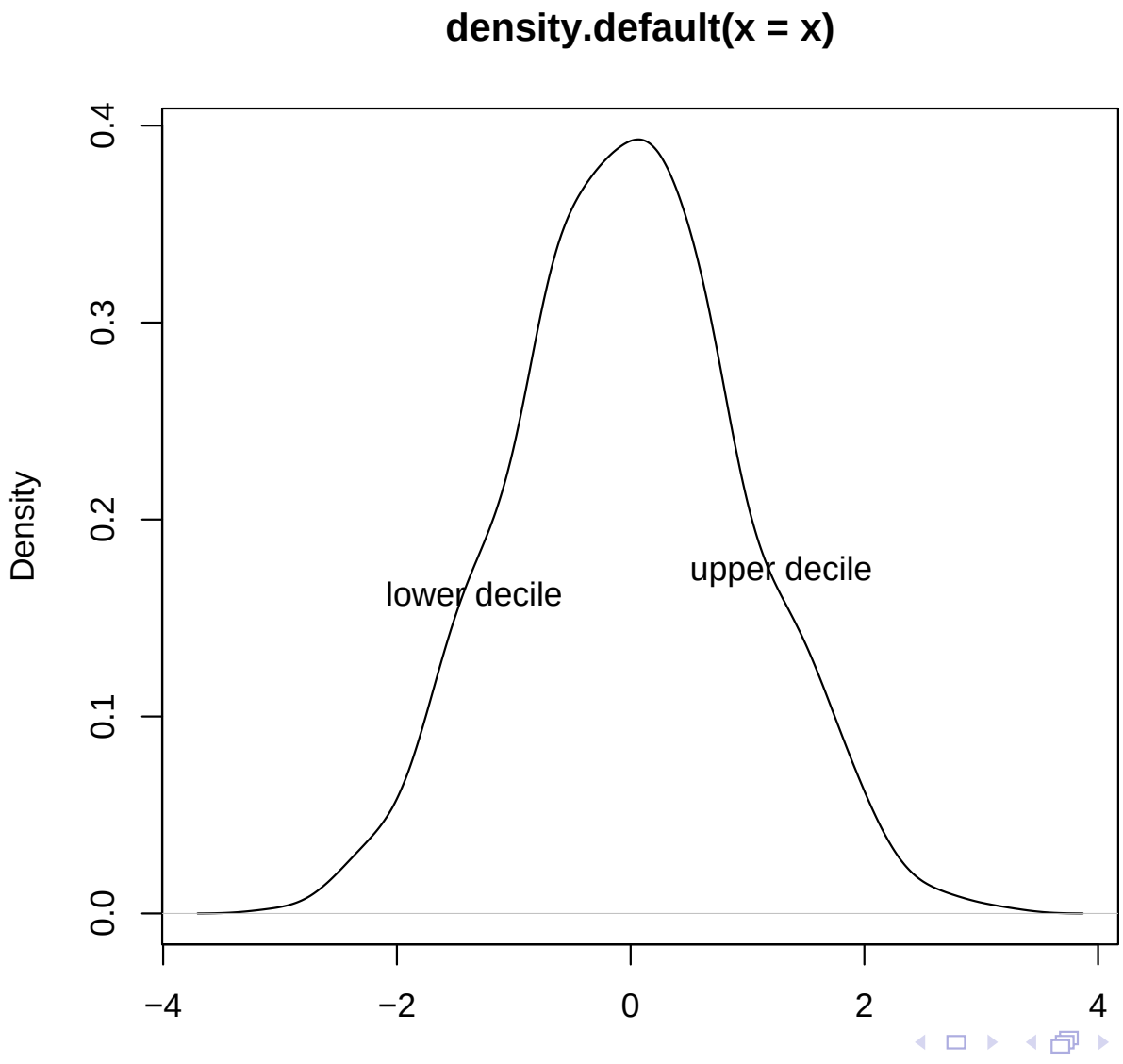

3TC
ABC
AZT
ddI
I(log(viral.load))
2.5
3.0
3.5
4.0
4.5
5.0
5.5
10
20
30
40
50
60
70
80
age

Adding text

- In order to add text to a plot we use the text.
- In order to add text to the margins of plots we use the function mtext. We will not cover mtext and many more advanced features of the plotting system, the excellent resource "R Graphics" by Paul Murrell covers these things in great detail.

```
> x <- rnorm(1000)
> plot(density(x))
> qtiles <- quantile(x, prob = seq(0,
+     1, length = 11))
> text(qtiles[2], dnorm(qtiles[2]),
+     "lower decile")
> text(qtiles[10], dnorm(qtiles[10]),
+     "upper decile")
```

Adding text


density.default(x = x)
Density
0.0
0.1
0.2
0.3
0.4
−4
−2
0
2
4
lower decile
upper decile
N = 1000 Bandwidth = 0.2214

Adding lines/arrows

- Often we want to add lines or arrows to a plot. We can do this using a number of functions: segments, points, lines, arrows, abline

Example

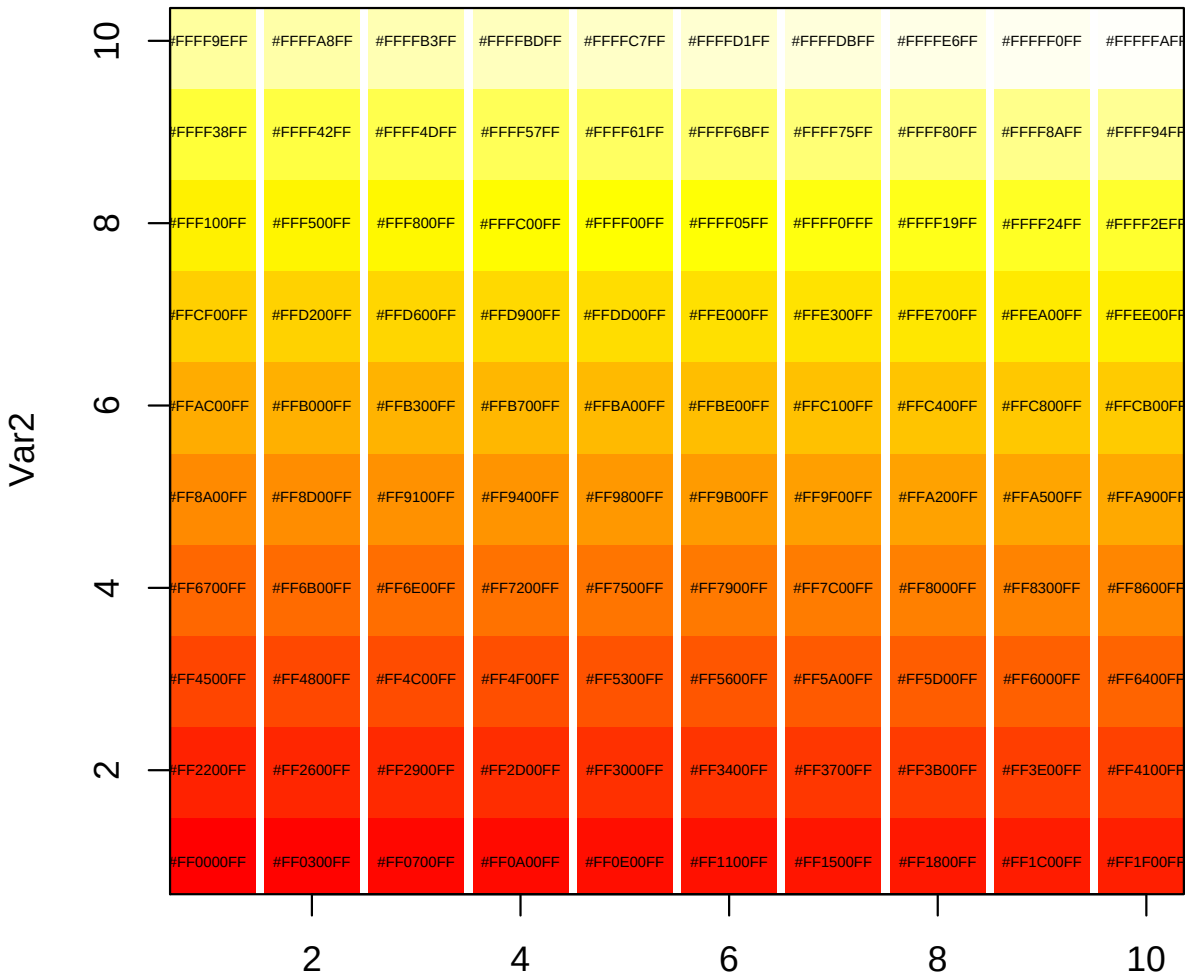
Recreate the normal distribution plot from above, but instead of adding text add arrows pointing to the lower and upper deciles. Label the arrows using the text command. Also, use the segments function to draw vertical lines at the boundaries where we have no data - that is draw vertical lines showing where the kernel density function density has interpolated values.

Colors

- R offers a number of tools to work with colors and color scales.
- Check out: topo.colors, heat.colors, gray.colors, colors, rainbow, cm.colors.
- Another useful resource is the website research.stowers-institute.org/efg/R/Color/Chart/

```
> cols <- heat.colors(10^2)[matrix(1:100,
+     nrow = 10)]
> plot(expand.grid(1:10, 1:10), col = cols,
+     pch = 15, cex = 6)
> text(expand.grid(1:10, 1:10), cols,
+     cex = 0.4)
```

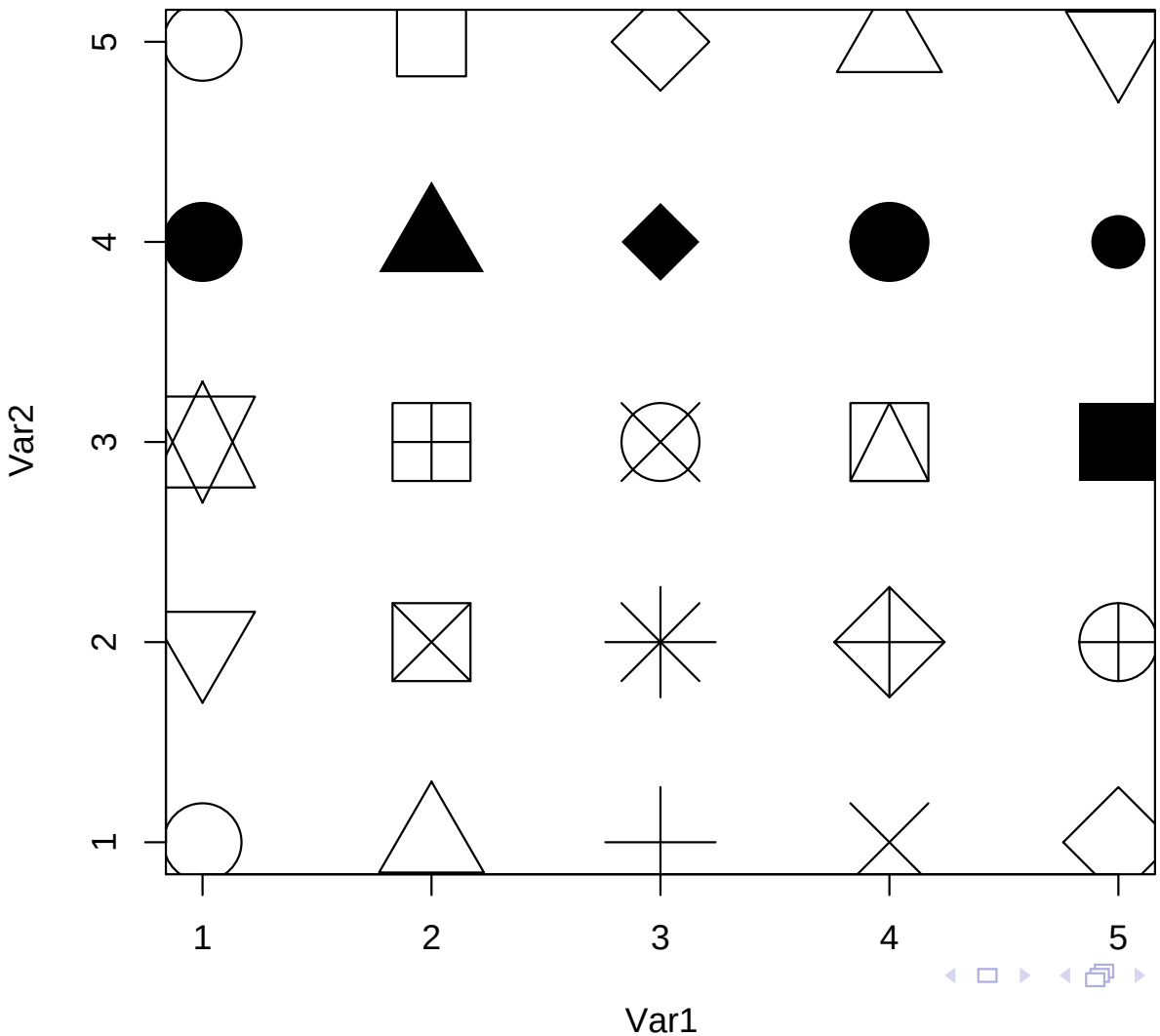
Colors


Var2
10
8
6
4
2
#FFFF9EFF #FFFFA8FF #FFFFB3FF #FFFFBDFF #FFFFC7FF #FFFFD1FF #FFFFDBFF #FFFFE6FF #FFFFF0FF
#FFFF38FF #FFFF42FF #FFFF4DFF #FFFF57FF #FFFF61FF #FFFF6BFF #FFFF75FF #FFFF80FF #FFFF8AFF #FFFF94FF
#FFF100FF #FFF500FF #FFF800FF #FFFC00FF #FFFF00FF #FFFF05FF #FFFF0FFF #FFFF19FF #FFFF24FF #FFFF2EFF
#FFCF00FF #FFD200FF #FFD600FF #FFD900FF #FFDD00FF #FFE000FF #FFE300FF #FFE700FF #FFEA00FF #FFEE00FF
#FFAC00FF #FFB000FF #FFB300FF #FFB700FF #FFBA00FF #FFBE00FF #FFC100FF #FFC400FF #FFC800FF #FFCB00FF
#FF8A00FF #FF8D00FF #FF9100FF #FF9400FF #FF9800FF #FF9B00FF #FF9F00FF #FFA200FF #FFA500FF #FFA900FF
#FF6700FF #FF6B00FF #FF6E00FF #FF7200FF #FF7500FF #FF7900FF #FF7C00FF #FF8000FF #FF8300FF #FF8600FF
#FF4500FF #FF4800FF #FF4C00FF #FF4F00FF #FF5300FF #FF5600FF #FF5A00FF #FF5D00FF #FF6000FF #FF6400FF
#FF2200FF #FF2600FF #FF2900FF #FF2D00FF #FF3000FF #FF3400FF #FF3700FF #FF3B00FF #FF3E00FF #FF4100FF
#FF0000FF #FF0300FF #FF0700FF #FF0A00FF #FF0E00FF #FF1100FF #FF1500FF #FF1800FF #FF1C00FF #FF1F00FF
2
4
6
8
10
Var1

Quick questions

1. What does expand.grid do?
2. What does `cex = .4` do?
3. If I tell you that there are 25 different plotting symbols from 1 to 25 can you make a similar plot as the one above with the blocks of colors replaced by the plotting symbols?

Quick questions



Literate Programming

- Sweave provides a tool for doing "literate programming" and "reproducible research" in R.
- "Literate programming" is an ideal coined by Donald Knuth - very related to the quote by Knuth: *Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.*
- Sweave was written by Friedrich Leisch, the manual can be found in the resources folder, additionally a paper by Robert Gentleman and Duncan Temple Lang about reproducible research can also be found in the resources folder.

Sweave

- Sweave works by gluing markup (latex) together with code (R) to construct documents - We want the analysis and the document to be one and the same, that way we can easily see that they match up.
- Sweave can generate a latex document or an HTML document.
- In order to work with Sweave we need to know a little `latex` and a little `R`.
- All my lectures, quizes, and homeworks were done using Sweave. All of the example code, answers are embedded in the document. All you have to do to get all the answers is do: `Stangle("slides.Rnw")`.

Nature Medicine (Baggerly 2007)

Recently, Potti et al. published an article in Nature Medicine reporting an approach predicting whether a tumor will respond to chemotherapy. In Microarrays: retracing steps, Baggerly et al. attempt to reproduce their analysis using the same data and code. They find that they are unable to reproduce the results claimed in the paper unless they deviate from the content of the paper. A particular quote is especially telling:

`We do not believe that any of the errors we found were intentional. We believe that the paper demonstrates a breakdown that results from the complexity of many bioinformatics analyses. This complexity requires extensive double-checking and documentation to ensure both data validity and analysis reproducibility. We believe that this`

Nature Medicine (Baggerly 2007)

> situation may be improved by an approach that allows a complete, auditable trail of data handling and statistical analysis. We use Sweave, a package that allows analysts to combine source code (in R) and documentation (in LaTeX) in the same file.

A Simple Example

1. In the "src/Sweave" directory of the course copy the file "simple.Rnw" locally and execute the following commands from within R

   ```
   > Sweave("simple.Rnw")
   ```

2. Now from the command line we need to run pdflatex on the generated tex file, this should be as easy as:

   ```
   thales:~ bullard$ pdflatex simple.tex
   ```

3. Now we should be able to open the newly created pdf file.

Look at the directory where we have run things - see how many files have been created! Generally it is a good idea to have a seperate directory for each .Rnw file.