Read in the Data

```
oj <- read.table("oj.txt",
    header=TRUE)
names(oj) <- c("machine", paste("Brand",
    c("A", "B", "C", "D", "E", "F")))
oj

##   machine Brand A Brand B Brand C Brand D Brand E Brand F
## 1      M1      89      97      92     105     100      91
## 2      M1      94      96      94     101     103      92
## 3      M2      92     101      94     110     100      95
## 4      M2      90     100      98     106     104      99
## 5      M3      90      98      94     109      99      94
## 6      M3      94      92      96     107      97      98
```

Fix the Data Frame

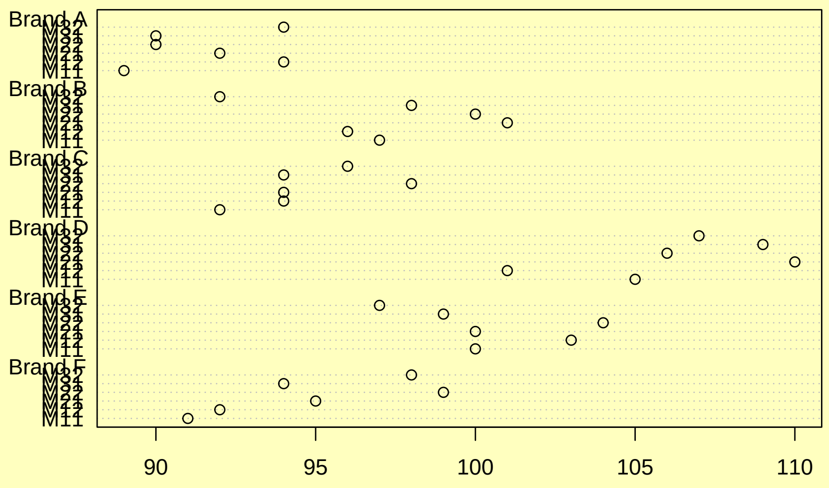
```
rownames(oj) <- paste(oj[,1], rep(1:2,3), sep="")
oj.mat <- as.matrix(oj[,-1])
oj.mat

##     Brand A Brand B Brand C Brand D Brand E Brand F
## M11      89      97      92     105     100      91
## M12      94      96      94     101     103      92
## M21      92     101      94     110     100      95
## M22      90     100      98     106     104      99
## M31      90      98      94     109      99      94
## M32      94      92      96     107      97      98
```

Plot a Dot Chart

The following code produces a dot chart ... but it is a bit hard to read.

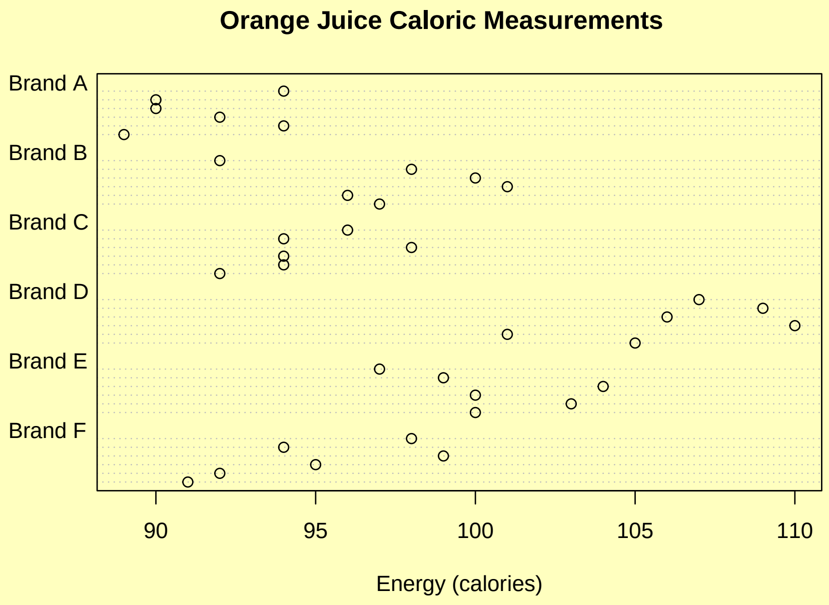
```
dotchart(oj.mat)
```



Fixing the Axis Labels

Remove the M's, since they are cluttering the vertical axis. Add a horizontal axis label and a title.

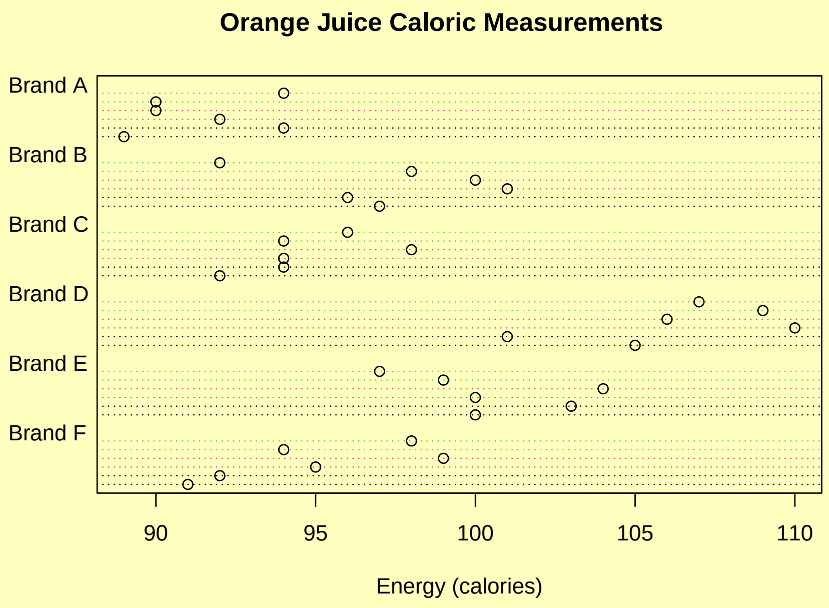
```
dotchart(oj.mat, labels="", xlab="Energy (calories)")
title("Orange Juice Caloric Measurements")
```



Fixing the Plot

Colour the lines:

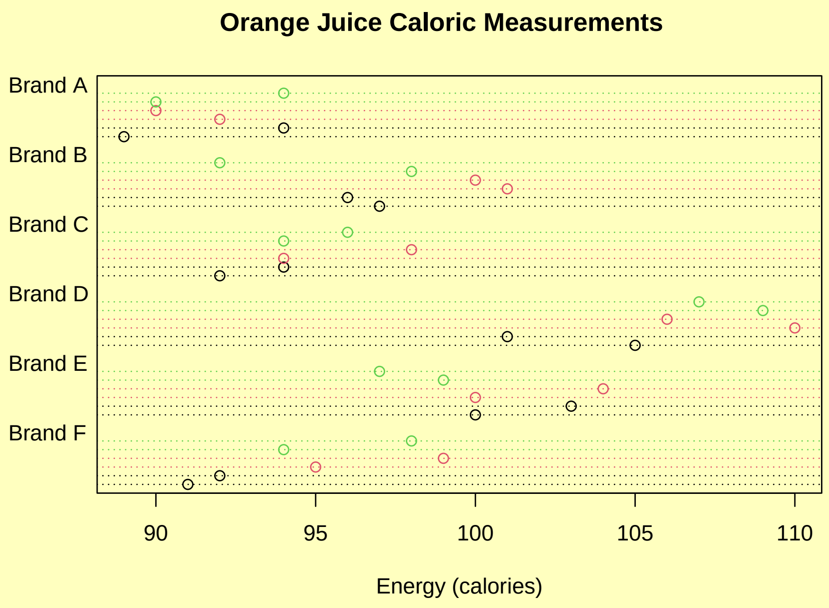
```
dotchart(oj.mat, labels="",  xlab="Energy (calories)",
      lcolor=rep(1:3,rep(2,3)))
title("Orange Juice Caloric Measurements")
```



Fixing the Plot

Colour the points:

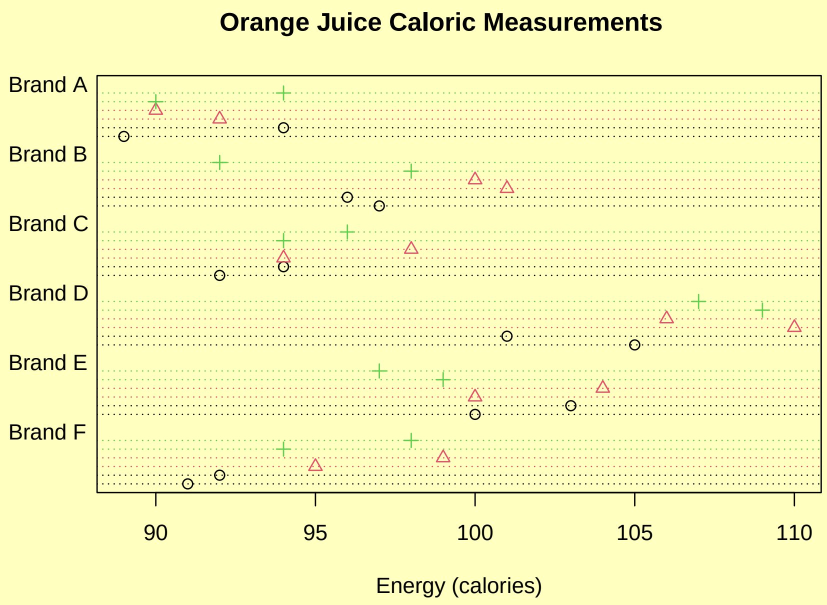
```
dotchart(oj.mat, labels="",     xlab="Energy (calories)",
    lcolor=rep(1:3,rep(2,3)),     color=rep(1:3,rep(2,3)))
title("Orange Juice Caloric Measurements")
```



Fixing the Plot

Use different plotting characters:

```
dotchart(oj.mat, labels="", xlab="Energy (calories)",
  lcolor=rep(1:3,rep(2,3)), color=rep(1:3,rep(2,3)),
  pch=rep(1:3, rep(2,3)))
title("Orange Juice Caloric Measurements")
```



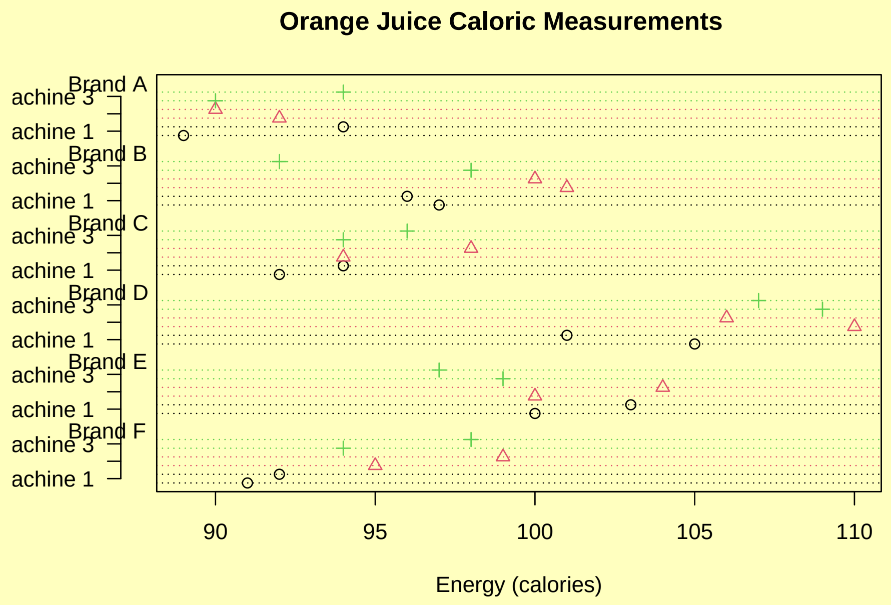
Fixing the Plot

Add axis labels to identify the machines, using `axis()`

```
dotchart(oj.mat, labels="",
    xlab="Energy (calories)",
    lcolor=rep(1:3,rep(2,3)),
    color=rep(1:3,rep(2,3)),
    pch=rep(1:3, rep(2,3)))
title("Orange Juice Caloric Measurements")
lab.locations <- seq(1.5,48,2)[-seq(4,48,4)]
labels <- paste("Machine", rep(1:3, 6))
axis(side=2, at=lab.locations,
                    label=labels, las=2)
```

Fixing the Plot

Add axis labels to identify the machines, using `axis()`



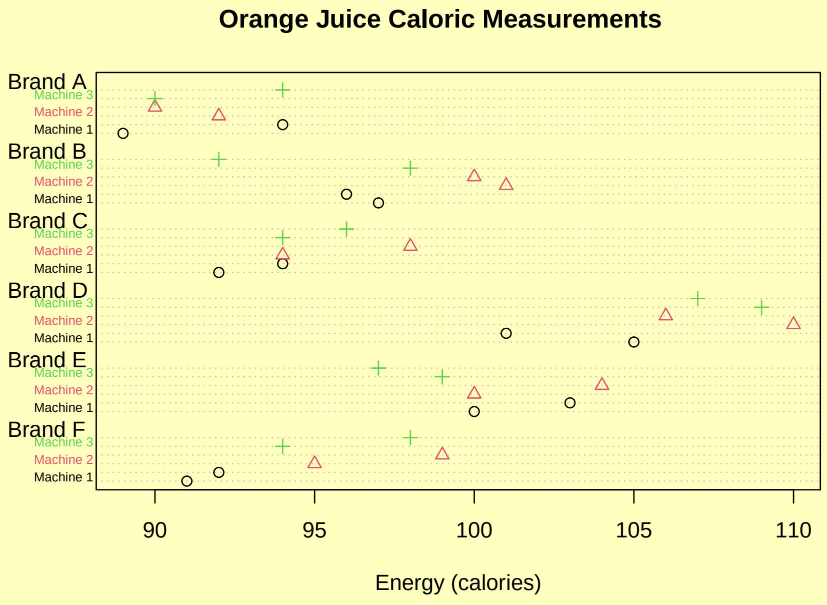
ouch!

Fixing the Plot

Fix the labels to identify the machines using `mtext()`:

```
dotchart(oj.mat, labels="",
    xlab="Energy (calories)",
    lcolor=rep("grey", 18), color=
    rep(1:3,rep(2,3)), pch=rep(1:3, rep(2,3)))
title("Orange Juice Caloric Measurements")
lab.locations <- seq(1.5,48,2)[-seq(4,48,4)]
labels <- paste("Machine", rep(1:3, 6))
mtext(labels, at=lab.locations, side=2,
    las=2, cex=.6, col=1:3, line=-1.25)
```

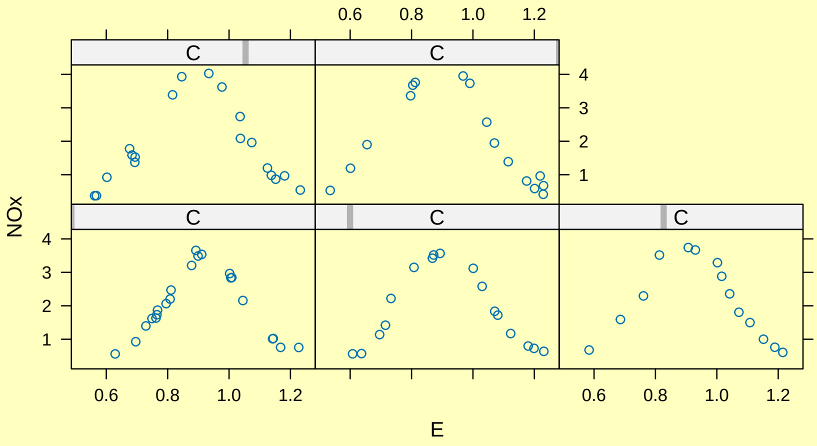
Fixing the Plot



Conditioning Plots - nitrous oxide emissions example

The following code produces a co-plot of nitrous oxide emissions (NOx) vs equivalency ratio (E) for each value of the compression ratio (C).

```
xyplot(NOx ~ E|C, data=ethanol)
```



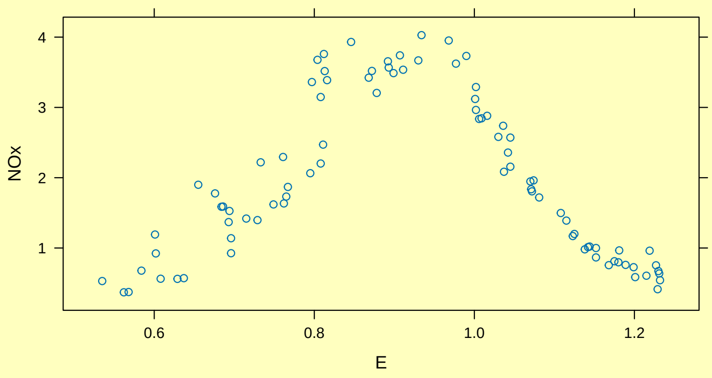
Each panel shows how the nitrous oxide emissions increase with equivalency ratio to a maximum and then decrease again.

The orange bar in the top bar of each panel indicates the relative size of C for that panel. That is, in the lower left panel, the value of C is lowest and in the upper right panel, the value of C is highest.

Conditioning Plots - nitrous oxide emissions example

Why are the conditioning plots useful? We can see this by comparing with what we would get by simply looking at a scatter plot of NOx vs E:

```
xyplot(NOx ~ E,  data=ethanol)
```



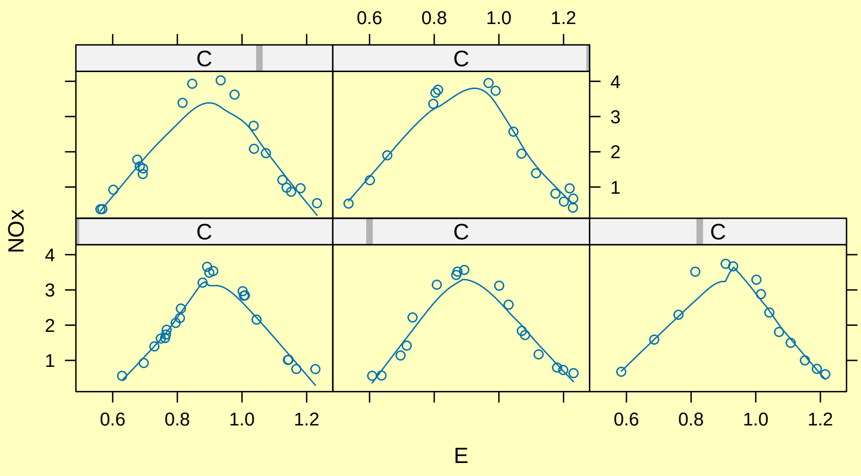
In the conditioning plots, a pattern of increase followed by decrease was clearly evident.

When we ignore the effects of C, we see very complicated looking patterns in the relation between NOx and E - these patterns are due to C, not due to how NOx and E are related.

Overlaying the conditioning plots with smooth curves

Again, we can overlay a smooth curve.

```
xyplot(NOx ~ E|C, data=ethanol, type=c("p", "smooth"), span=.65)
```

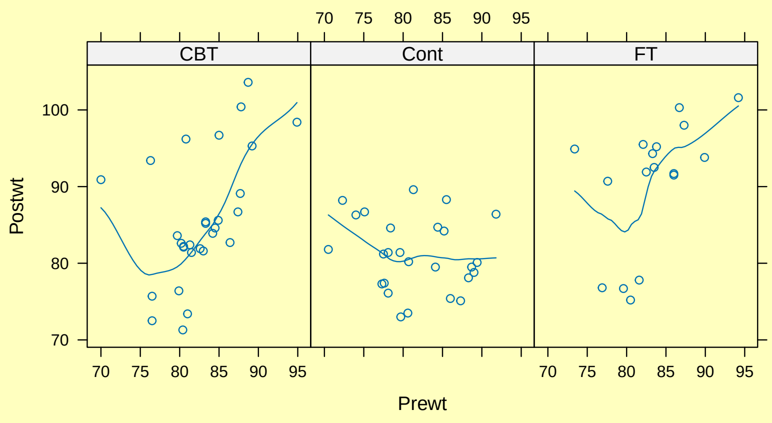


The `span` argument indicates what proportion of the data should be used to estimate each point of the smooth curve.

Another look at the anorexia data

Another way to visualize pre/post data is to use a scatter plot relating the post-study data to the pre-study data. Here we do that for each treatment group.

```
xyplot(Postwt ~ Prewt|Treat, data = anorexia,
    type=c("p", "smooth"),
    span=.75)
```



Now, we see a clear difference between the control group and the treatment groups.

For pre-study weights above 82 pounds, the control is not having a good effect, but the other therapies are. For pre-study weights below about 80-82 pounds, it does not seem to make a difference.

Another look at the anorexia data

We can reconstruct the dot plots now to take this new information into account.

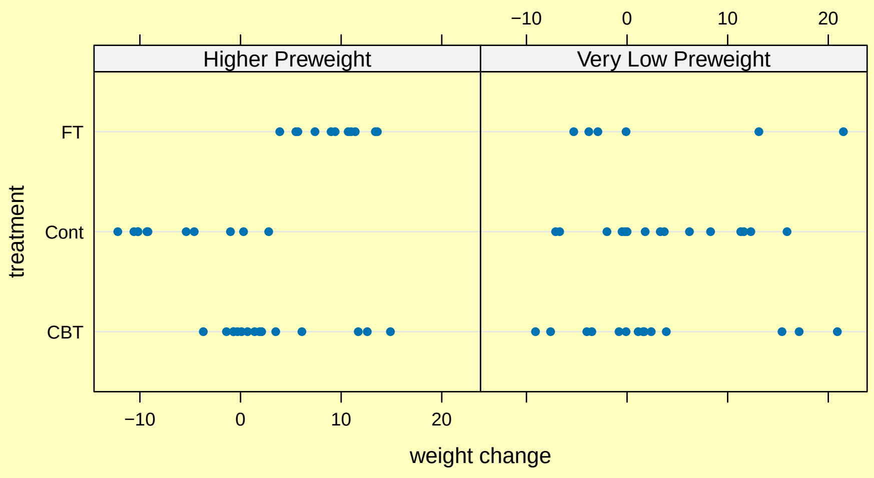
We can create a factor which separates the very low pre-weight subject from the others as follows:

```
anorexia$lowPrewt <- factor(anorexia$Prewt < 82)
levels(anorexia$lowPrewt) <-
     c("Higher Preweight", "Very Low Preweight")
```

Another look at the anorexia data

The dot plots, conditional on whether the pre-study weight was very low or not are constructed as follows:

```
dotplot(Treat ~ change|lowPrewt, data = anorexia,
    xlab = "weight change", ylab="treatment")
```

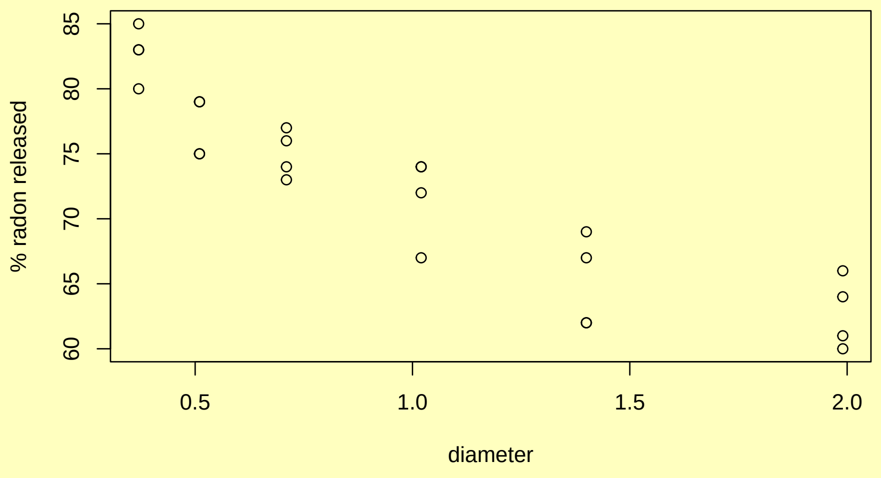


Now, we see that for subjects with a very low pre-study weight, there are no differences, but for subjects with a high enough pre-study weight, the therapies really appear to help, especially the Family Therapy.

Radon release data

Plot the data first:

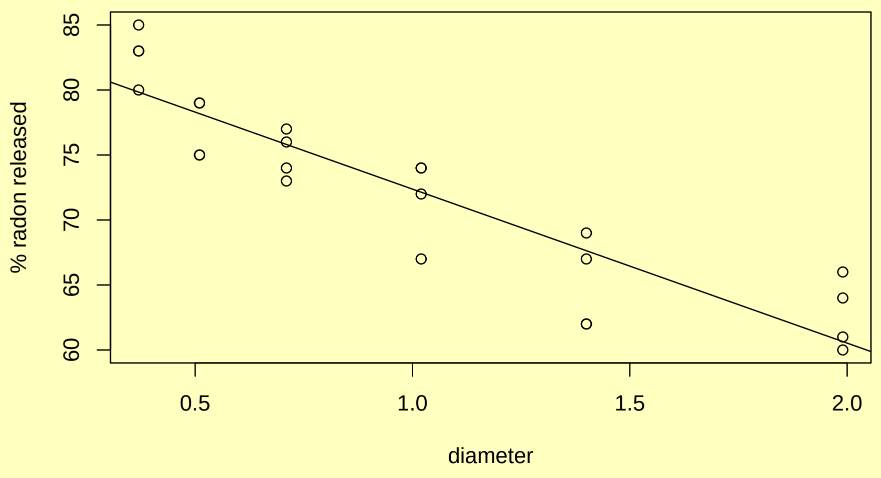
```
plot(percentage ~ diameter, data = radon,
       ylab="% radon released")
```



Goal: to predict the percentage of radon released for a given diameter.

Radon release data and best-fit line

```
plot(percentage ~ diameter, data = radon,
       ylab="% radon released")
radon.lm <- lm(percentage ~ diameter, data = radon)
abline(radon.lm)
```

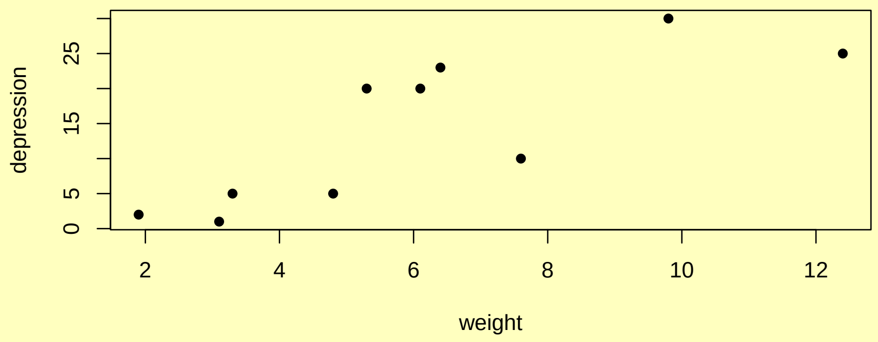


The goal of these slides is to understand how the best-fit line is calculated.

Another example: lawn roller data

Different weights of roller (in kilograms) were used to roll over different parts of a lawn, and the depth of the depression (in millimeters) was recorded at various locations.

```r
library(DAAG) # DAAG contains the roller data
plot(depression ~ weight, data = roller, pch=16)
```



Again, we want to predict the size of the depression for a given weight. We start by studying models for such data. The first model to look at is the Simple Linear Regression Model.

The simple linear regression model

- **Measurement of Y (response) changes in a linear fashion with a setting of the variable x (predictor):**

$$\mathrm{Y} = \underbrace{\beta_0 + \beta_1 \mathrm{x}}_{\text{linear relation}} + \underbrace{\varepsilon}_{\text{noise}}$$

  - **The linear relation is deterministic (non-random).**
  - **β_0 represents the intercept of the line. This *parameter* is unknown and must be estimated from data.**
  - **β_1 represents the slope of the line. This *parameter* is unknown and must be estimated from data.**

The simple linear regression model

- ○ The noise or error is random.

- Noise accounts for the variability of the observations about the straight line.
  No noise $\Rightarrow$ relation is deterministic.
  Increased noise $\Rightarrow$ increased variability.

- The variability in the noise is due to any factors, other than the weight of the roller. For example, the type of soil (sandy or hard clay, etc), or amount of moisture in different parts of the lawn, and so on.

- The variability in the noise is summarized by the parameter σ (pronounced "sigma"). Larger values of σ lead to larger amounts of noise.

The simple linear regression model

Experiment with this simulation program:

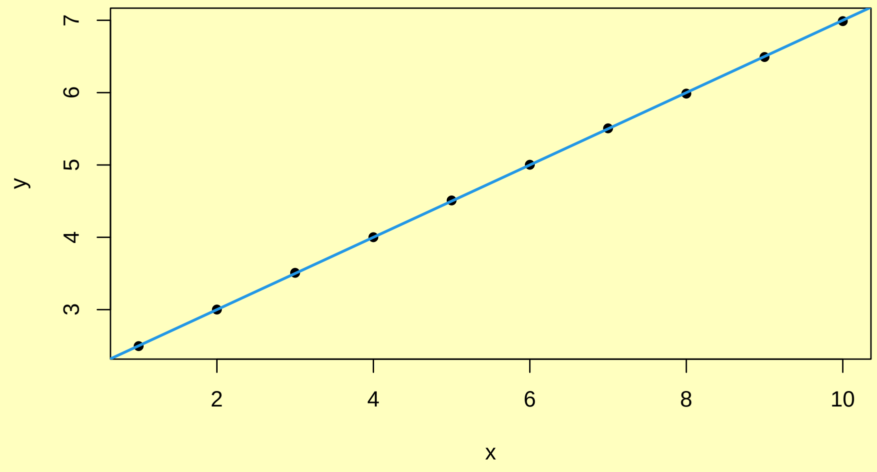
```
simple.sim <- function(intercept=0,slope=1,x=seq(1,10),
     sigma=1){
noise <- rnorm(length(x),sd = sigma)
y <- intercept + slope*x + noise
plot(x,y,pch=16) # plot noisy data
abline(intercept,slope,col=4,lwd=2) # blue line, no noise
}
```

Download *simplesim.R* from Canvas and source it into R. Also, watch the videos *simpleSimVideo.mp4* and *simpleSimVideo2.mp4* to see how data noisier as the value of σ (sigma) increases.

The simple linear regression model

Example 1: $y = 2 + 0.5x + \varepsilon$**, with** $\sigma = 0.01$**:**

```
simple.sim(intercept = 2, slope = 0.5, sigma=.01)
```

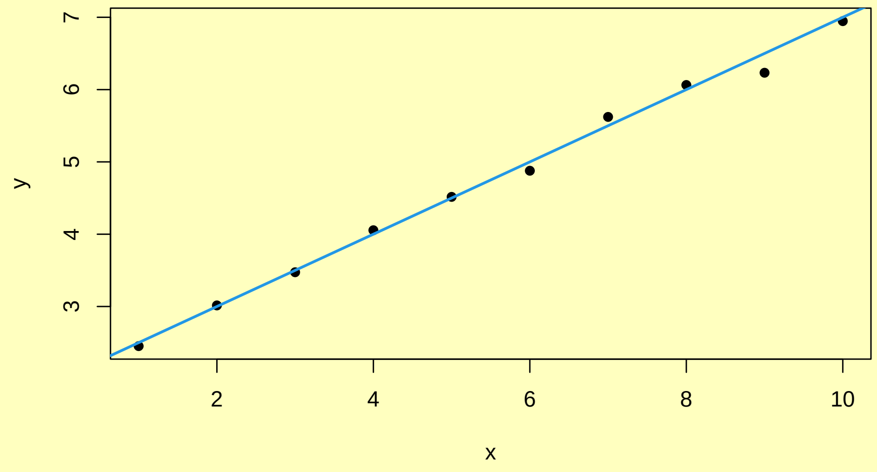


Very little noise, so the points lie very close to the straight line.

The simple linear regression model

Example 2: $y = 2 + 0.5x + \varepsilon$**, with** $\sigma = 0.1$**:**

```
simple.sim(intercept = 2, slope = 0.5, sigma=.1)
```

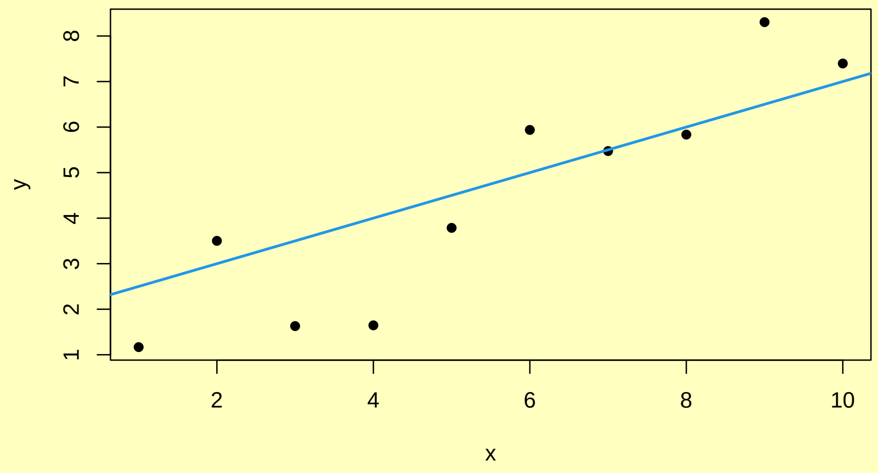


A little more noise, so the points are not as close to the straight line.

The simple linear regression model

Example 3: $y = 2 + 0.5x + \varepsilon$, **with** $\sigma = 1$**:**

```
simple.sim(intercept = 2, slope = 0.5, sigma=1)
```

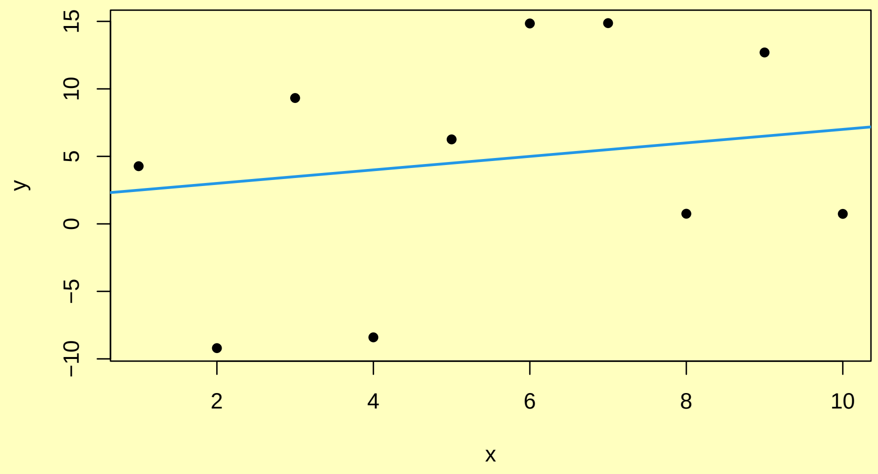


A lot more noise, so the points are scattered about the straight line.

The simple linear regression model

Example 4: $y = 2 + 0.5x + \varepsilon$, **with** $\sigma = 10$**:**

```
simple.sim(intercept = 2, slope = 0.5, sigma=10)
```



Mostly noise, so the line is no longer very recognizable from the points.

The Setup

- **Assumptions**:
  1. Expected value of $y = \beta_0 + \beta_1 \mathrm{x}$.
  2. Standard Deviation$(\varepsilon) = \sigma$.
- **Data**: Suppose data $\mathrm{Y}_1, \mathrm{Y}_2, \ldots, \mathrm{Y}_n$ are obtained at settings $\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_n$, respectively. Then the model on the data is

$$\mathrm{Y}_i = \beta_0 + \beta_1 \mathrm{x}_i + \varepsilon_i$$

  Either
  1. the x's are fixed values and measured without error (controlled experiment) - Example: Radon Data

     OR
  2. the analysis is conditional on the observed values of x (observational study) - Example: Roller Data

Parameter estimation, fitted values and residuals

Least Squares Estimation

- **Assumptions**:
  1. The mean of the noise term ε_i is 0.
  2. The standard deviation of ε_i is σ.
  3. ε_i's are independent.

Method

- We want to choose the parameters (or regression coefficients) β_0 and β_1: $\widehat{\beta}_0$ and $\widehat{\beta}_1$ so that the fitted line passes as close to all of the points as possible.

○ Aim: small Residuals (observed - fitted response values):

$$\mathbf{e}_i = \mathbf{Y}_i - \widehat{\beta}_0 - \widehat{\beta}_1 \mathbf{x}_i$$

Setting up the predictive model for brain weight

In order to find out how brain weight relates to both body weight and litter size, we can use the following model:

$$\texttt{brainwt} = \beta_0 + \beta_1 \texttt{bodywt} + \beta_2 \texttt{lsize} + \varepsilon$$

This is an example of a *multiple regression model*. It is a little more complicated to fit than a simple regression model, but the `lm` function still applies.

There is still a response variable `brainwt` on the left side of the model formula, but now there are two predictor variables `bodywt` and `lsize` on the right side of the model formula:

```
brainwt ~ bodywt + brainwt
```

Fitting the model in R

```
litters.lm <- lm(brainwt ~ bodywt + lsize, data = litters)
coef(litters.lm)

## (Intercept)      bodywt       lsize
## 0.178246962 0.024306344 0.006690331
```

The fitted model is then

$$\widehat{y} = .18 + .024x_1 + .0067x_2$$

where Y is brain weight, x_1 is body weight and x_2 is litter size.

Note that this fitted model says that for a fixed body weight, brain weight is actually higher for larger litters.

This is consistent with what is known as 'brain sparing': nutritional deprivation that results from large litter sizes has a proportionately smaller effect on brain weight than on body weight.

The brain sparing effect can be visualized - but only with effort

Our earlier visualization with the `pairs` plot did not reveal the brain sparing effect, but a conditional plot can. We need condition on different levels of body weight to see this.

We will use the `cut` function to turn the numeric `bodywt` variable into a factor with interval based categories.

Example of use of `cut`, where we find intervals $(5, 6], (6, 7], ..., (9, 10]$ which contain the different body weights:

```
cut(litters$bodywt, 5:10)

##  [1] (9,10] (9,10] (9,10] (9,10] (8,9]  (9,10] (8,9]  (8,9]  (7,8]
## [10] (8,9]  (7,8]  (7,8]  (6,7]  (7,8]  (6,7]  (6,7]  (7,8]  (6,7]
## [19] (5,6]  (6,7]
## Levels: (5,6] (6,7] (7,8] (8,9] (9,10]
```

The output tells us that the first four body weights are in the interval $(9, 10]$ and the next two are in the interval $(8, 9]$, which agrees with the tabular display on slide 3.

The brain sparing effect can be visualized - but only with effort

We will choose cutpoints that divide the body weights into 6 approximately equal-sized groups.

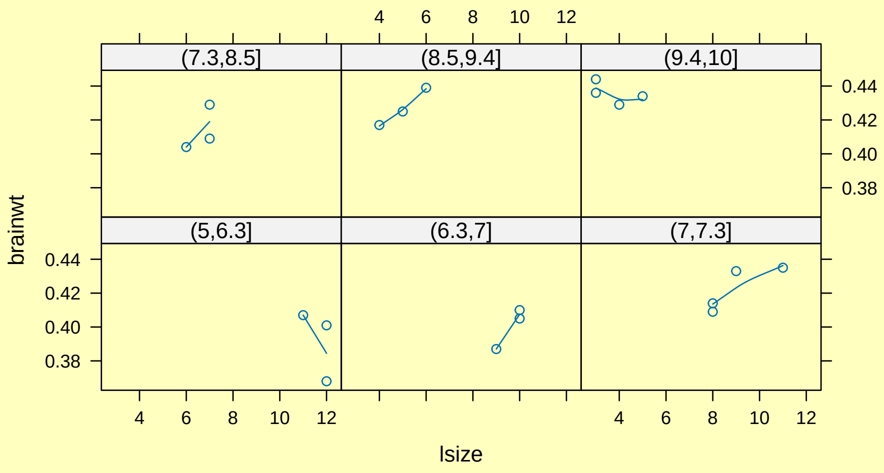
```
cutpoints <- c(5, 6.3, 7, 7.3, 8.5, 9.4, 10)
cutpoints

## [1]  5.0  6.3  7.0  7.3  8.5  9.4 10.0
```

The brain sparing effect can be visualized - but only with effort

Then we plot brain weight against litter size for each of these groups with the `xyplot`:

```
xyplot(brainwt ~ lsize|cut(bodywt, cutpoints),
    data = litters, type=c("p", "smooth"), span = 2)
```

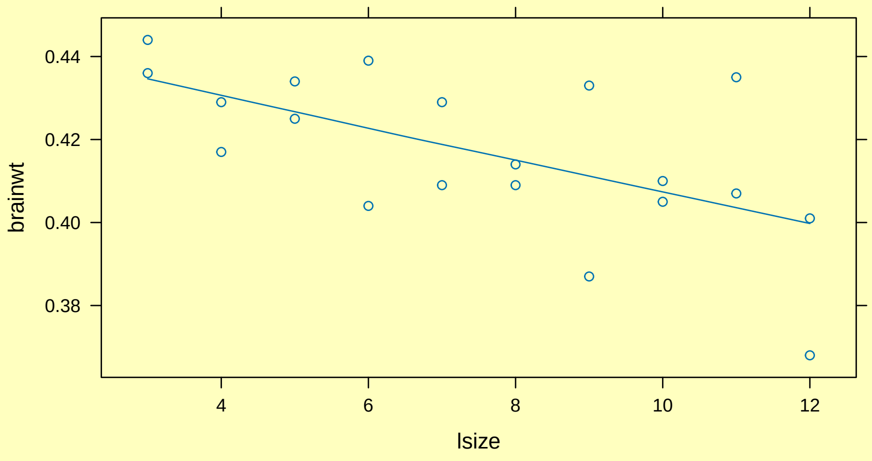


This plot shows that for relatively fixed values of body weight, the brain weight is somewhat more likely to grow with litter size than to decrease.

The brain sparing effect can be visualized - but only with effort

As stated earlier, the brain sparing effect is completely hidden if we don't try to condition on fixed values of body weight:

```
xyplot(brainwt ~ lsize, data = litters, type=c("p", "smooth"), span = 2)
```

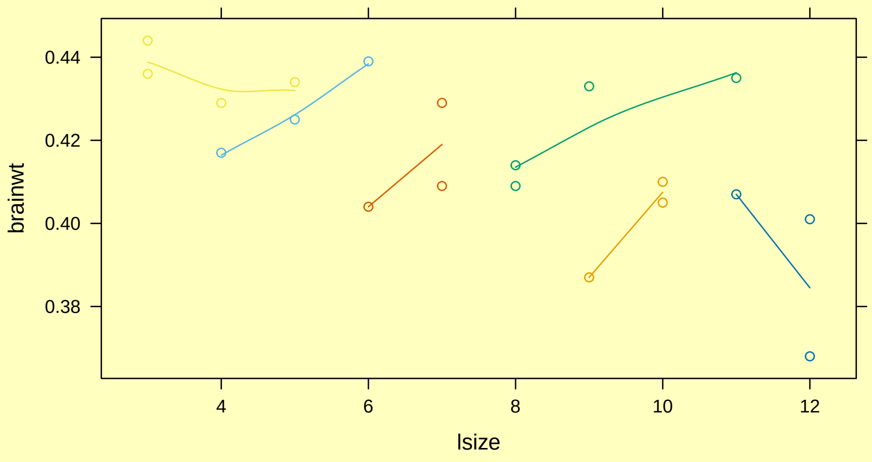


This plot hides the body weight effects.

The brain sparing effect can be visualized - but only with effort

Here is another view:

```
xyplot(brainwt ~ lsize, data = litters, groups=cut(bodywt, cutpoints),
    type=c("p", "smooth"), span = 2)
```



This time, we use different colours, using the `groups` argument, to represent the points and smooths corresponding to the different body weights. Now, we see that, often, for fixed body weight, brain weight increases with litter size. The light green and blue points are exceptions.

Making predictions with the fitted model

If we have a mouse born in a litter of size 6 with a body weight of 8.5, we can predict the brain weight from the fitted model:

$$\widehat{y} = .18 + .024(8.5) + .0067(6) = 0.42487.$$

We can do this automatically with the `predict` function:

```
predict(litters.lm, newdata =
    data.frame(bodywt = 8.5, lsize = 6))


##      1
## 0.425
```

We can also obtain a 95% prediction interval:

```
predict(litters.lm, newdata =
    data.frame(bodywt = 8.5, lsize = 6), interval="prediction")

##     fit   lwr   upr
## 1 0.425 0.399 0.451
```

Leave-one-out Cross-Validation: PRESS

- **PRESS residuals:**

$$e_{(i)} = y_i - \widehat{y}_{(i)}.$$

Here, y_i is the ith observed response and $\widehat{y}_{(i)}$ is the predicted value at the ith observation based on the regression of y against the xs, omitting the ith observation.

- **PRedicted Error Sum of Squares:**

$$\text{PRESS} = \sum_{i=1}^{n} e^2_{(i)}$$

This gives an idea of how well a regression model can predict new data. Small values of PRESS are desired.

PRESS - `litters` example

```
# regression of brain weight against body weight and litter size:
> litters.lm <- lm(brainwt ~ bodywt + lsize, data = litters)
 PRESS(litters.lm)
[1] 0.0035 # same regression as above, but without the intercept
term:
> litters.0 <- lm(brainwt ~ bodywt + lsize -1, data=litters)
> PRESS(litters.0)
[1] 0.00482 # regression of brain weight against body weight only,
with intercept:
> litters.1 <- lm(brainwt ~ bodywt, data=litters)
> PRESS(litters.1)
[1] 0.00385 # regression of brain weight against both variables
plus an interaction term:
> litters.2 <- lm(brainwt ~ bodywt + lsize + lsize:bodywt, data=litters)
> PRESS(litters.2)
[1] 0.0037 # best predictor is the 1st model!
```

Example - winning football games

The data in `table.b1` in the *MPV* package concern the number of games won `y` in a 14 game season, together with measurements on specific aspects of the game, such as the number of yards that the football was passed, and kicked, and so on. There are 9 variables like this, labeled `x1` through `x9`.

```
library(MPV) # contains table.b1 - football example
```

We will use `PRESS` to help choose between some possible models.

Example - winning football games

We can fit the model that relates `y` to ALL of the `x`'s by using a dot (.):

```
all.lm <- lm(y ~ . , data = table.b1)
PRESS(all.lm)   # calculate PRESS value for this full model

## [1] 145.9
```

Fit a model that only contains `x2`, `x4`, `x7`, `x8` and `x9`:

```
five.lm <- lm(y ~ x2 + x4 + x7 + x8 + x9, data = table.b1)
PRESS(five.lm)

## [1] 97.13
```

Compare with a similar model that does not have `x7`:

```
four.lm <- lm(y ~ x2 + x4  + x8 + x9, data = table.b1)
PRESS(four.lm)

## [1] 119.2
```

Example - winning football games

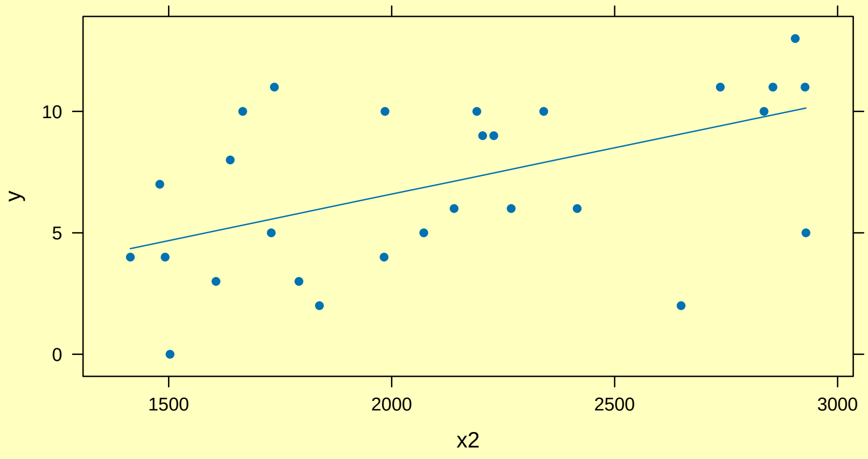
Since the `PRESS` value is smallest for the five variable model, we would prefer that one.

We can partially visualize the effects of the various variables on the response `y`, using the conditioning plots and by use of the `groups` argument.

Example - winning football games

If we do not condition at all, but view `y` as a function of `x2`, we have

```
xyplot(y ~ x2, data = table.b1, pch=16, type=c("smooth", "p")
```

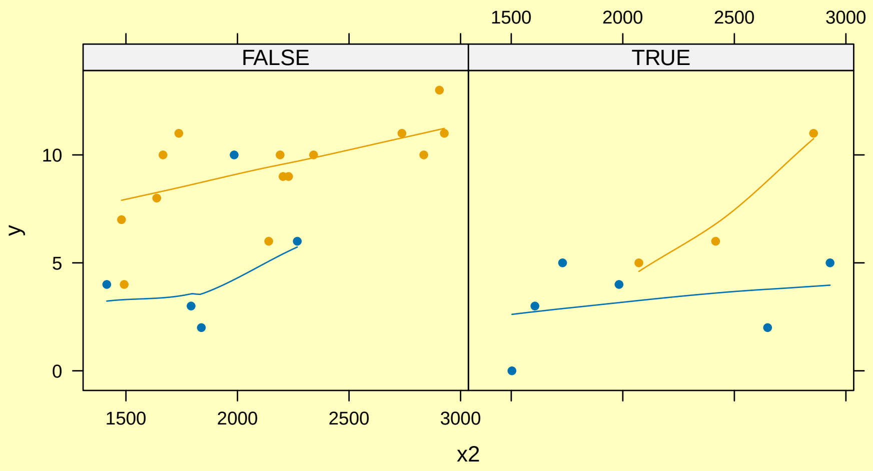


There is a lot of noise around our predictions.

Example - winning football games

Splitting x7 roughly in half, and also x8 - as the group variable, we have

```
xyplot(y ~ x2|(x7 < 56), groups=(x8 < 2100), data = table.b1,
     pch=16, type=c("smooth", "p"), span=2)
```



By considering roughly fixed values of x7 and x8, we have more precise predictions of y based on x2.

Example - winning football games

We can predict the number of wins for a team with 2000 passing yards x2, 60% field goal percentage x4, 80% rushing x7, 1900 opponent rushing yards x8 and 1800 opponent passing yards x9:

```
predict(five.lm, newdata = data.frame(x2 = 2000, x4 = 60,
    x7 = 80, x8 = 1900, x9 = 1800))


##     1
## 12.99
```

We would predict that this team would win 13 out of the 14 games.

Example - winning football games

A prediction interval can be obtained from

```
predict(five.lm, newdata = data.frame(x2 = 2000, x4 = 60,
    x7 = 80, x8 = 1900, x9 = 1800), interval="prediction")

##      fit   lwr   upr
## 1 12.99 7.669 18.31
```

The prediction interval is pretty wide (7 to 18), and since there are only 14 games, some of the interval is impossible. We could be very confident that this team will win at least one-half of its games.