Introduction to the Practice of Statistics using R: Chapter 2

Ben Baumer Nicholas J. Horton*

July 31, 2013

Contents

Introduction

This document is intended to help describe how to undertake analyses introduced as examples in the Sixth Edition of *Introduction to the Practice of Statistics* (2009) by David Moore, George McCabe and Bruce Craig. More information about the book can be found at `http://bcs.whfreeman.com/ips6e/`. This file as well as the associated `knitr` reproducible analysis source file can be found at `http://www.math.smith.edu/~nhorton/ips6e`.

This work leverages initiatives undertaken by Project MOSAIC (`http://www.mosaic-web.org`), an NSF-funded effort to improve the teaching of statistics, calculus, science and computing in the undergraduate curriculum. In particular, we utilize the `mosaic` package, which was written to simplify the use of R for introductory statistics courses. A short summary of the R needed to teach

*Department of Mathematics, Amherst College, nhorton@amherst.edu

introductory statistics can be found in the mosaic package vignette (`http://cran.r-project.org/web/packages/mosaic/vignettes/MinimalR.pdf`).

To use a package within R, it must be installed (one time), and loaded (each session). The package can be installed using the following command:

```
> install.packages('mosaic')                # note the quotation marks
```

The # character is a comment in R, and all text after that on the current line is ignored.

This chapter also references a dataset from the second edition of *The Statistical Sleuth*, so this must be installed as well.

```
> install.packages('Sleuth2')                # note the quotation marks
```

Once the package is installed (one time only), they can be loaded by running the command:

```
> require(mosaic)
> require(Sleuth2)
```

This needs to be done once per session.

We also set some options to improve legibility of graphs and output.

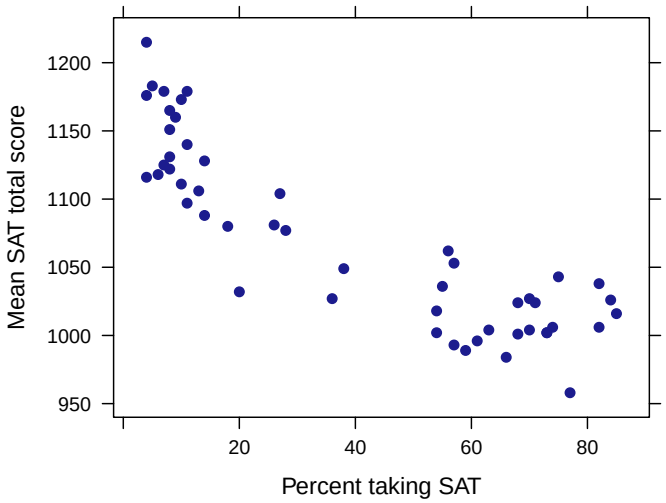
```
> trellis.par.set(theme=col.mosaic())  # get a better color scheme for lattice
> options(digits=3)
```

The specific goal of this document is to demonstrate how to replicate the analysis described in Chapter 2: Looking at Data (Relationships).

1 Scatterplots

Example 2.6 (page 87) shows a scatterplot of the average SAT scores for all 50 states and the District of Columbia.

```
> SAT = read.csv("http://www.math.smith.edu/ips6eR/ch02/eg02_006.csv")
> head(SAT)

       state percent satv satm total
1    Alabama      10  559  552  1111
2     Alaska      55  518  518  1036
3    Arizona      38  524  525  1049
4   Arkansas       6  564  554  1118
5 California      54  499  519  1018
6   Colorado      27  551  553  1104

> plotPoints(total ~ percent, data=SAT, xlab="Percent taking SAT",
   ylab="Mean SAT total score", pch=19)
```



This can also be generated using the `xyplot()` function.

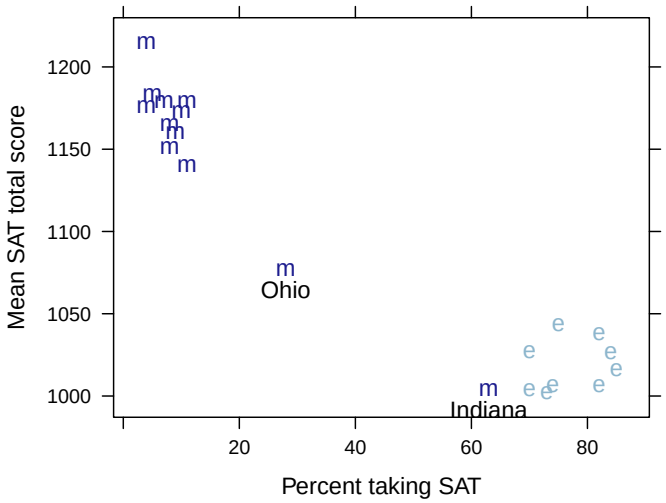
1.1 Adding categorical variables to scatterplots

Figure 2.2 (page 89) illustrates the use of different plotting symbols as a means of distinguishing levels of a categorical variable in a scatterplot. This can be achieved just as easily using color and the `groups` attribute. First, we assign a region to each state. This can be done using the `transform` and `ifelse` commands.

```
> midwest = c("Ohio", "Michigan", "Wisconsin", "Indiana", "Illinois", "Minnesota"
              , "Iowa", "NorthDakota", "SouthDakota", "Nebraska"
              , "Kansas", "Missouri")
> northeast = c("Maine", "NewHampshire", "Vermont", "Massachusetts", "Connecticut"
                , "RhodeIsland", "NewYork", "NewJersey", "Pennsylvania")
> SAT = transform(SAT, region = ifelse(state %in% midwest, "midwest",
   ifelse(state %in% northeast, "northeast", NA)))
```

Now that the `region` variable is set, we can use the `groups` argument to separate the states by color. Adding specific labels can be achieved by using `ladd` and `panel.text` to add things to an existing plot. Note the use of the `subset()` function to restrict the states which are being plotted and labelled.

```
> plotPoints(total ~ percent, groups=region, data=subset(SAT, !is.na(region)),
   xlab="Percent taking SAT", ylab="Mean SAT total score", pch=c("m", "e"),
   cex=1.5)
> SAT.labels = subset(SAT, state %in% c("Ohio", "Indiana"))
> with(SAT.labels, ladd(panel.text(percent, total, state, pos=1)))
```



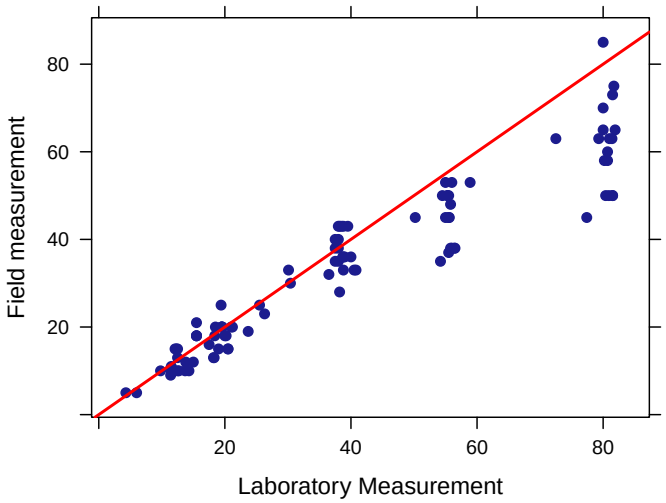
1.2 More examples of scatterplots

Example 2.7 (page 90) concerns the Trans-Alaska Pipeline. The dataset is in the following format:

```
> Oil = read.csv("http://www.math.smith.edu/ips6eR/ch02/eg02_007.csv")
> head(Oil)

  field  lab
1    18 20.2
2    38 56.0
3    15 12.5
4    20 21.2
5    18 15.5
6    36 39.0
```

We can add a straight line to a plot using the `panel.abline` function. In this we specify the line $y = x$ by giving the first argument (the intercept) as 0 and the second argument (the slope) as 1. Thus, the `abline` is $y = a + bx$.

```
> plotPoints(field ~ lab, data=Oil, xlab="Laboratory Measurement",
   ylab="Field measurement", pch=19)
> ladd(panel.abline(a=0, b=1, col="red"))
```

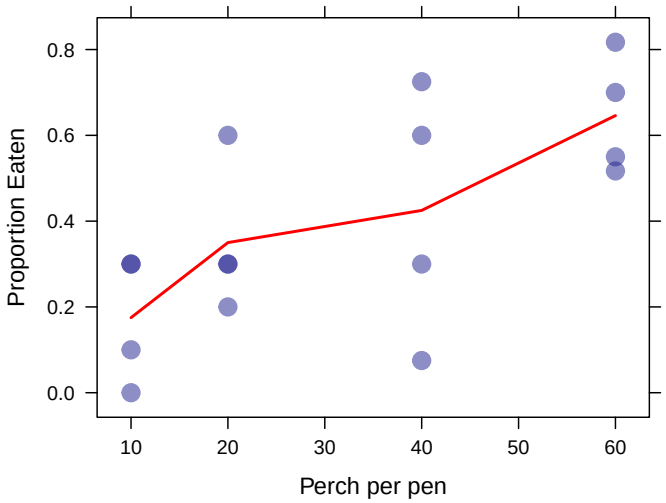


Example 2.8 (page 91) relates the density of perch to the proportion that are killed by predators.

```
> Perch = read.csv("http://www.math.smith.edu/ips6eR/ch02/eg02_008.csv")
> head(Perch)

  perch killed
1    10    0.0
2    10    0.1
3    10    0.3
4    10    0.3
5    20    0.2
6    20    0.3
```

Figure 2.4 (pagee 92) shows the relationship between perch density and the proportion killed. Note that some data points occur multiple times. Although the textbook uses different plot marks to distinguish single instances from those with multiplicity, we can use transparency to achieve the same effect.

```
> plotPoints(killed ~ perch, data=Perch, xlab="Perch per pen",
   ylab="Proportion Eaten", pch=19, cex=1.5, alpha=0.5)
> perch.means = mean(~killed | perch, data=Perch)
> perch.means

   10    20    40    60
0.175 0.350 0.425 0.646

> ladd(panel.lines(names(perch.means), perch.means, col="red"))
```

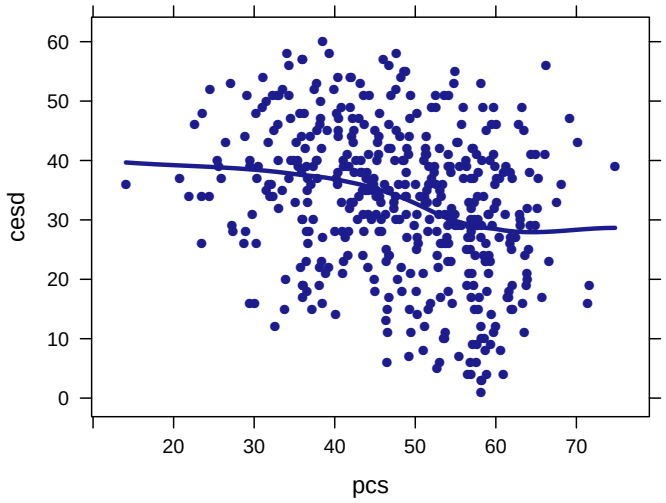


Note the use of the bar notation for computing the groupwise means.

1.3 Smoothing

Because the data for Figure 2.5 (page 93), we display how to add a scatterplot smoother to a figure using data from the HELP (Health Evaluation and Linkage to Primary Care) study.

```
> xyplot(cesd ~ pcs, type=c("p", "smooth"), lwd=3, data=HELPrct)
```



2 Correlation

Correlation can be computed using the `cor()` function. For example, we can create two random vector and compute their correlation:

```
> x = runif(100)
> y = runif(100)
> cor(x,y)

[1] -0.0873
```

Of course, because the random variables are generated independently of each other, the sample correlation should be close to zero. Moreover, you can compute the pairwise correlation coefficients for more than two vectors at one time.

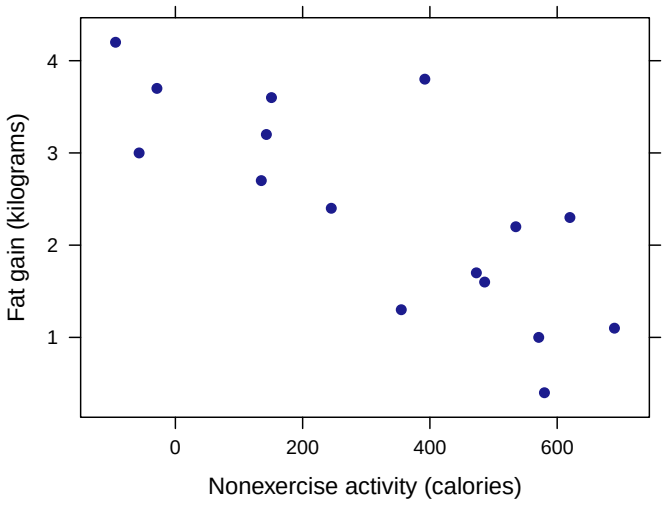
```
> z = runif(100)
> cor(data.frame(x,y,z))

        x       y       z
x  1.0000 -0.0873 -0.0683
y -0.0873  1.0000 -0.1044
z -0.0683 -0.1044  1.0000
```

The 1's along the diagonal indicate the correlation of a vector with itself.

3 Least-Squares Regression

Figure 2.11 (page 109) shows the relationship between non-exercise activity (*nea*) and fat gain after 8 weeks of overeating.

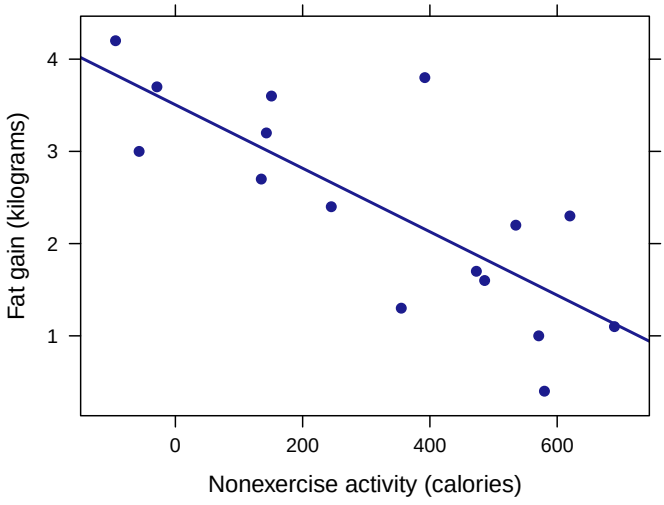
```
> Fat = read.csv("http://www.math.smith.edu/ips6eR/ch02/eg02_012.csv")
> head(Fat)

  nea fat
1 -94 4.2
2 -57 3.0
3 -29 3.7
4 135 2.7
5 143 3.2
6 151 3.6

> xyplot(fat ~ nea, data=Fat, xlab="Nonexercise activity (calories)",
   ylab="Fat gain (kilograms)", pch=19)
```



3.1 Fitting a line to data

If we just want to add the least squares regression line to the plot, we can specificy the "r" value for the `type` argument to the plotting function.

```
> plotPoints(fat ~ nea, xlab="Nonexercise activity (calories)",
   ylab="Fat gain (kilograms)", pch=19, type=c("p", "r"), data=Fat)
```



However, in addition to plotting the line, we often will want to extract further information about the regression model that we built. To build the model, we use the `lm` command, and to see the coefficients from the model, we use `coef`.

```
> fm = lm(fat ~ nea, data=Fat)
> coef(fm)

(Intercept)         nea
    3.50512    -0.00344
```

3.2 Prediction

There are a few different ways of using our model, now that we've built it. A nice way is to run `makeFun()` on the model to convert it into a function. Then, we can simply ask the resulting function to compute estimate based on our model.

```
> fit.fat = makeFun(fm)
```

Note that the arguments to this function are named.

In Example 2.14 (page 111), we use the model for fat gain to estimate the fat gain for an individual whose NEA increases by 400 calories.

```
> fit.fat(nea = 400)

   1
2.13
```

This is certainly easier than calculating it manually using R as a calculator:

```
> 3.50512 - 0.00344*400

[1] 2.13
```

In Example 2.15 (page 112), we use the model for fat gain to esimate the fat gain for an individual whose NEA increases by 1500 calories.

```
> fit.fat(nea = 1500)

    1
-1.66
```

This information can also be extracted using the `predict` function. In this case, we can ask the model to return values for many different inputs at once.

```
> nea.new = data.frame(nea = c(400, 1500))
> predict(fm, newdata = nea.new)

    1     2
 2.13 -1.66
```

3.3 Least-squares regression

In Example 2.16, we verify that the coefficients returned by the regression model can be computed directly from the correlation coefficient, along with the means and standard deviations of the two variables. The correlation coefficient is:

```
> cor(fat, nea, data=Fat)

[1] -0.779
```

The means and standard deviation are given below.

```
> favstats(~fat, data=Fat)

 min   Q1 median  Q3 max mean   sd  n missing
 0.4 1.53   2.35 3.3 4.2 2.39 1.14 16       0

> favstats(~nea, data=Fat)

 min  Q1 median  Q3 max mean  sd  n missing
 -94 141    374 544 690  325 258 16       0
```

The slope of the regression line is the product of the correlation coefficient and the ratio of the standard deviations. We can verify this:

```
> coef(fm)["nea"]

     nea
-0.00344

> slope = with(Fat, cor(fat, nea) * (sd(fat) / sd(nea)))
> slope

[1] -0.00344
```

With the slope in hand, we can then compute the intercept, since we know that the point $(\bar{x}, \bar{y})$ is on the regression line.

```
> coef(fm)["(Intercept)"]

(Intercept)
       3.51

> intercept = with(Fat, mean(fat) - slope * mean(nea))
> intercept

[1] 3.51
```

3.4 Correlation and Regression

The `summary` command will show a table containing information about a regression model that is similar to the information shown in Figure 2.14 (page 116).

```
> summary(fm)


Call:
lm(formula = fat ~ nea, data = Fat)

Residuals:
   Min     1Q Median     3Q    Max
-1.109 -0.390 -0.104  0.413  1.644

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.505123   0.303616   11.54  1.5e-08 ***
nea         -0.003441   0.000741   -4.64  0.00038 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.74 on 14 degrees of freedom
Multiple R-squared:  0.606,Adjusted R-squared:  0.578
F-statistic: 21.5 on 1 and 14 DF,  p-value: 0.000381
```

For the purposes of Chapter 2 of IPS, regression is being used to describe relationships, so the primary focus is the regression parameter estimates in the first numeric column.

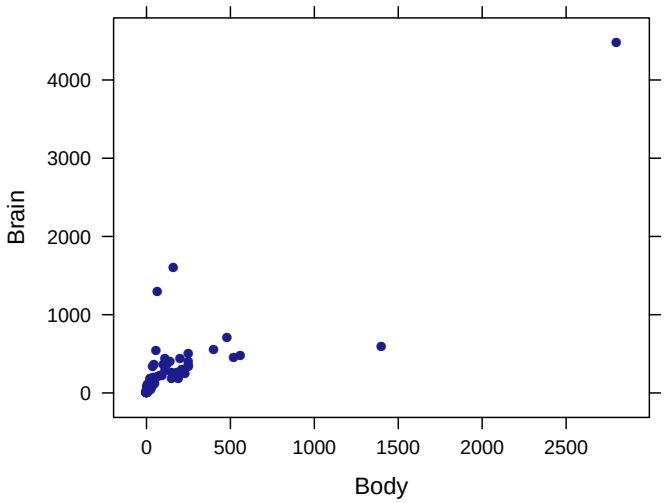
For least squares regression of one variable, the square of the correlation coefficient is equal to the coefficient of determination (or R-squared aka R^2).

```
> cor(fat, nea, data=Fat)^2

[1] 0.606

> r.squared(fm)

[1] 0.606
```

3.5 Transforming Relationships

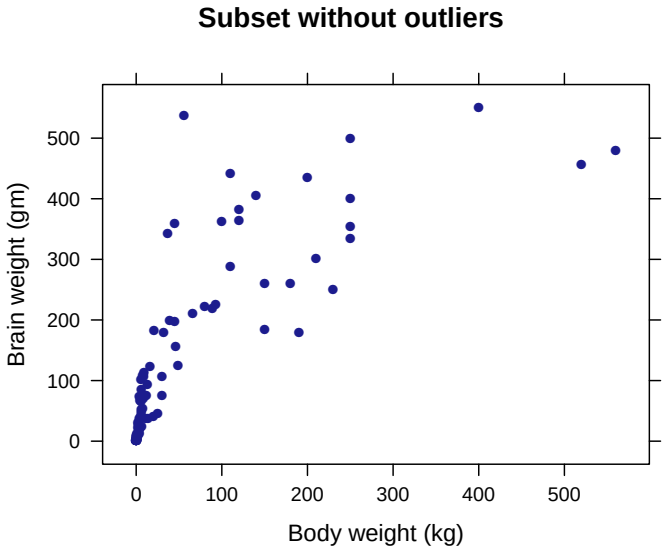
The dataset from Example 2.17 (page 119), Example 2.18 and Figure 2.18 resemble data from Chapter 9 of the Statistical Sleuth.

```
> xyplot(Brain ~ Body, data=case0902)
```



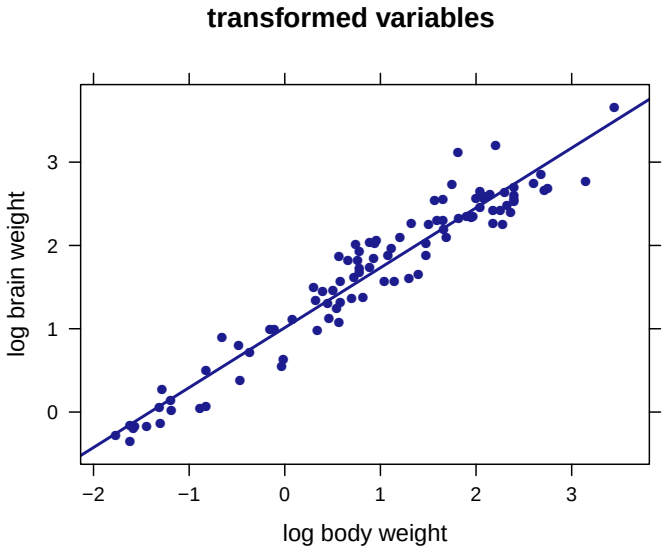
We can remove outliers, which yields a plot similar to that in Figure 2.18 (page 120).

```
> smaller = subset(case0902, Brain < 700 & Body < 1000)
> xyplot(Brain ~ Body, xlab="Body weight (kg)", ylab="Brain weight (gm)",
   main="Subset without outliers", data=smaller)
```



We can also transform the two variables, as seen in Figure 2.19 (page 121).

```
> case0902 = transform(case0902, logbrain = log10(Brain))
> case0902 = transform(case0902, logbody = log10(Body))
> xyplot(logbrain ~ logbody, xlab="log body weight", ylab="log brain weight",
   main="transformed variables", type=c("p", "r"), data=case0902)
```



4 Cautions about Correlation and Regression

The residuals can be extracted easily from a linear regression model, as displayed in Example 2.19 (page 126).

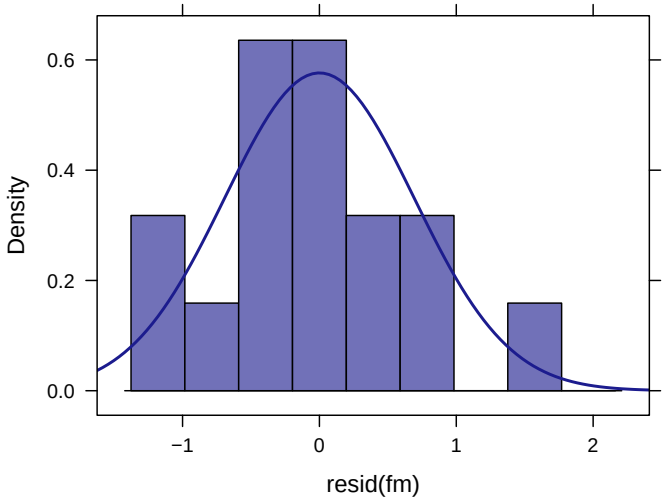
```
> resid(fm)

      1       2       3       4       5       6       7       8       9
 0.3714 -0.7013  0.0951 -0.3405  0.1870  0.6145 -0.2620 -0.9834  1.6439
     10      11      12      13      14      15      16
-0.1773 -0.2326  0.5361 -0.5400 -1.1091  0.9286 -0.0305
```

Note that the mean of the residuals is always (by mathematical construction) equal to zero.

```
> mean(resid(fm))

[1] 0
```

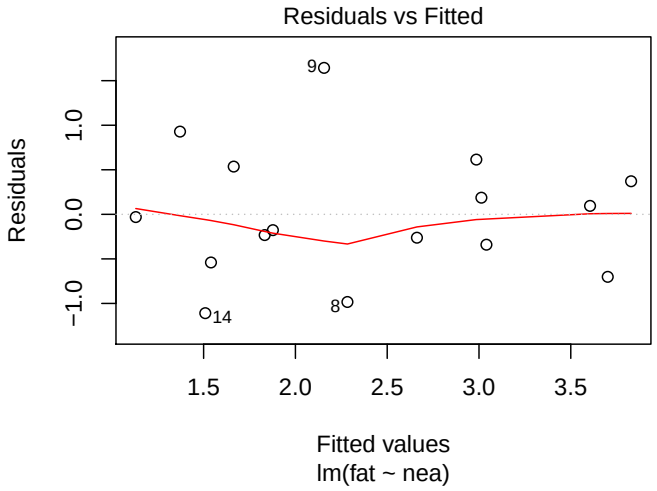
It's straightforward to display the univariate distribution of the residuals, to help assess the normality assumption.

```
> histogram(~ resid(fm), fit="normal")

Loading required package:  MASS
```



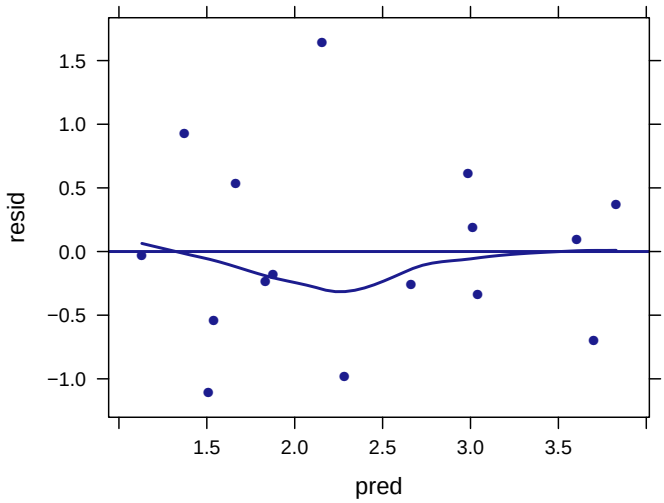
R has a built-in function for showing residual plots. You can simply `plot` the model object, and ask for only the first (of four) diagnostic plots.

```
> plot(fm, which=1)
```



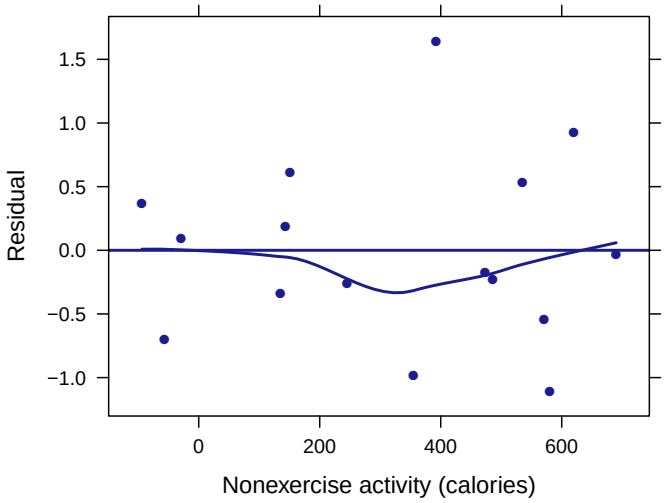
Alternatively this can be created in parts, using `plotPoints()`.

```
> Fat = transform(Fat, pred = fitted(fm))
> Fat = transform(Fat, resid = residuals(fm))
> plotPoints(resid ~ pred, type=c("p", "r", "smooth"), data=Fat)
```



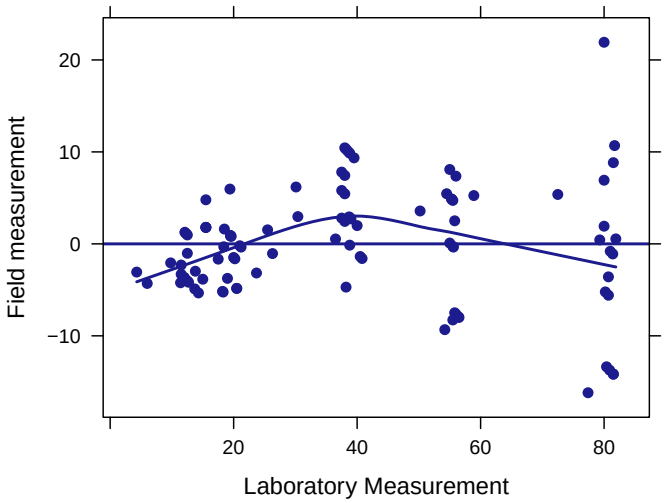
For both of these R plots the residuals against the fitted values. However, in Figure 2.20 (page 127), the residuals are plotted against the values of the explanatory variable.

```
> plotPoints(resid(fm) ~ nea, ylab="Residual",
   xlab="Nonexercise activity (calories)", type=c("p", "r", "smooth"), data=Fat)
```



A similar plot is shown in Figure 2.21 (page 128).

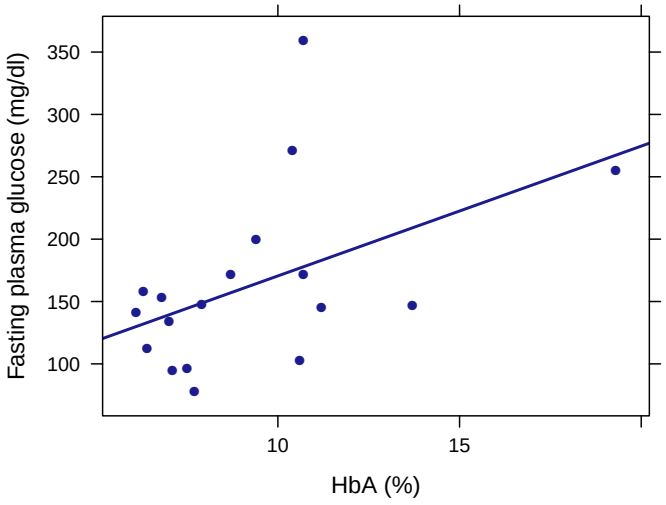
```
> fm.oil = lm(field ~ lab, data=Oil)
> plotPoints(resid(fm.oil) ~ lab, xlab="Laboratory Measurement",
   ylab="Field measurement", pch=19, type=c("p", "r", "smooth"),
   data=Oil)
```



4.1 Outliers and influential observations

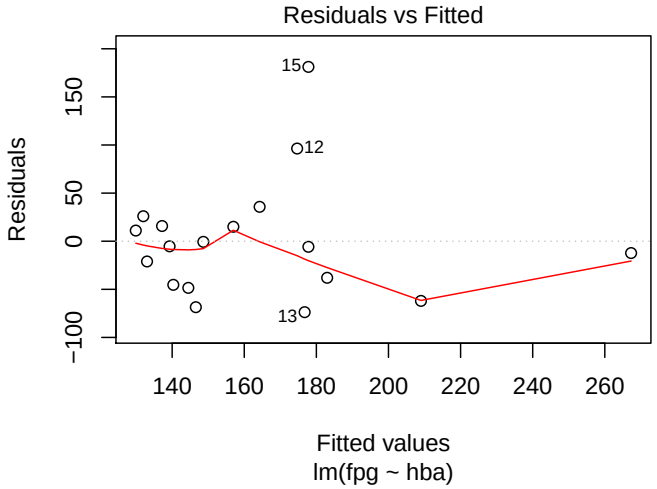
The analyses displayed in Example 2.21 (page 129) and Figures 2.22 and 2.23 (page 130) can be generated in a straightforward manner.

```
> diabetes = read.csv("http://www.math.smith.edu/ips6eR/ch02/ta02_005.csv")
> xyplot(fpg ~ hba, xlab="HbA (%)", ylab="Fasting plasma glucose (mg/dl)",
       type=c("p", "r"), data=diabetes)
```



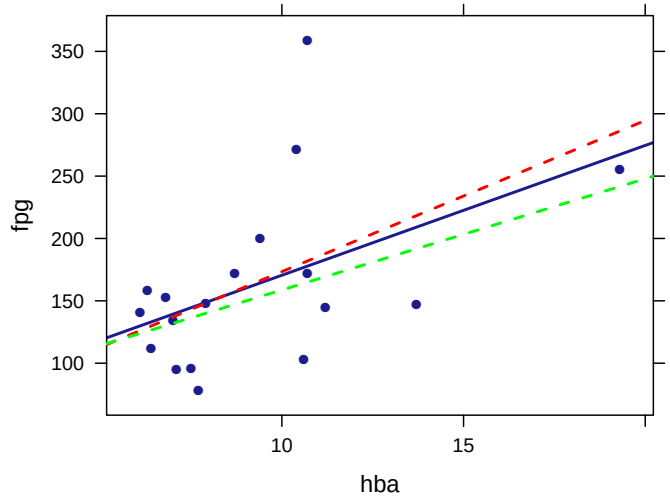

```
> outliermodel = lm(fpg ~ hba, data=diabetes)
> coef(outliermodel)
```

```
(Intercept)         hba
       66.4        10.4

> plot(outliermodel, which=1)
```



To replicate the display in Figure 2.24 (page 131), we need to fit the model twice more dropping either subject 15 or subject 18, and generating predicted lines for each of these models.

```
> no18 = lm(fpg ~ hba, data=subset(diabetes, obs != 18))
> no15 = lm(fpg ~ hba, data=subset(diabetes, obs != 15))
> plotPoints(fpg ~ hba, type=c("p", "r"), data=diabetes)
> ladd(panel.abline(no18, col="red", lty=2))
> ladd(panel.abline(no15, col="green", lty=2))
```



4.2 Beware the lurking variable

In Example 2.24 (page 133), the association between private health care spending and goods imported to the United States is examined each year between 1990 and 2001.

```
> Imports = read.csv("http://www.math.smith.edu/ips6eR/ch02/eg02_024.csv")
> head(Imports)

  imports health
1     498    696
2     491    762
3     537    827
4     590    888
5     669    937
6     749    990
```

Figure 2.25 (page 133) illustrates the relationship.

```
> plotPoints(health ~ imports, data=Imports, pch=19)
```