5

Practical Data Analysis – Examples

Scatterplot matrices	Scatterplot matrices can give useful insights on data that will be used for regression or related calculations.
Transformation	Data often require transformation prior to entry into a regression model.
Model objects	Fitting a regression or other such model gives, in the first place, a model object.
Generic functions	`plot()`, `print()` and `summary()` are examples of *generic* functions. With a dataframe as argument `plot()` gives a scatterplot matrix. With an `lm` object, it gives diagnostic plots.
Extractor function	Use an extractor function to extract output from a model object. Extractor fucntions are *generic* functions
List objects	An `lm` model object is a list object. Lists are used extensively in R.

This chapter will use examples to illustrate common issues in the exploration of data and the fitting of regression models. It will round out the discussion of Chapters 1 to 2 and 4 by adding some further important technical details.

Issues that will be noted include the use of *generic* functions such as `plot()` and `print()`, the way that regression model objects are structured, and the use of extractor functions to extract information from model objects.

Notation, when referring to datasets

Data will be used that is taken from several different R packages. The notation `MASS::mammals`, which can be used in code as well as in the textual description, makes it clear that the dataset `mammals` that is required is from the *MASS* package. Should another attached package happen to have a dataset `mammals`, there is no risk of confusion.

5.1 The Uses of Scatterplot Matrices

```
## Below, the dataset MASS::mammals will be required
library(MASS, quietly=TRUE)
```

5.1.1 Transformation to an appropriate scale

A first step is to elicit basic information on the columns in the data, including information on relationships between explanatory variables. Is it desirable to transform one or more variables?

Among other issues, is there a wide enough spread of distinct values that data can be treated as continuous.

Transformations are helpful that ensure, if possible, that:

- All columns have a distribution that is reasonably well spread out over the whole range of values, i.e., it is unsatisfactory to have most values squashed together at one end of the range, with a small number of very small or very large values occupying the remaining part of the range.
- Relationships between columns are roughly linear.
- the scatter about any relationship is similar across the whole range of values.

It may happen that the one transformation, often a logarithmic transformation, will achieve all these at the same time.

The scatterplot in Figure 5.1A, showing data from the dataset `MASS::mammals`, is is an extreme version of the common situation where positive (or non-zero) values are squashed together in the lower part of the range, with a tail out to the right. Such a distribution is said to be "skewed to the right".

Code for Figure 5.1A

```
plot(brain ~ body, data=mammals)
mtext(side=3, line=0.5, adj=0, "A: Unlogged data")
```

Figure 5.1B shows the scatterplot for the logged data. Code for Figure 5.1B is:

```
plot(brain ~ body, data=mammals, log="xy")
mtext(side=3, line=0.5, adj=0,
      "B: Log scales (both axes)")
```

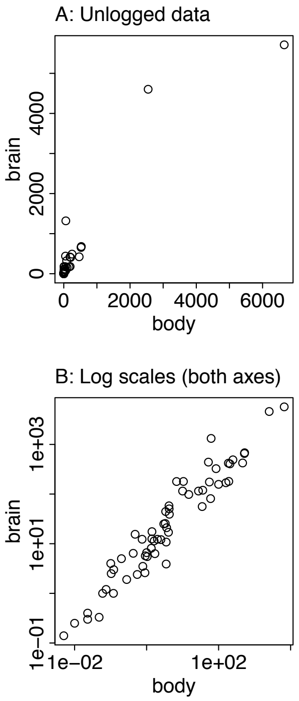


Figure 5.1: Brain weight (g) versus Body weight (kg), for 62 species of mammal. Panel A plots the unlogged data, while Panel B has log scales for both axes, but with axis labels in the original (unlogged) units.

Where, as in Figure 5.1A, values are concentrated at one end of the range, the small number (perhaps one or two) of values that lie at the other end of the range will, in a straight line regression with that column as the only explanatory variable, be a leverage point. When it is one explanatory variable among several, those values will have an overly large say in determining the coefficient for that variable.

As happened here, a logarithmic transformation will often remove much or all of the skew. Also, as happened here, such transformations often bring the added bonus that relationships between the resulting variables are approximately linear.

5.1.2 *The Uses of Scatterplot Matrices*

Subsequent chapters will make extensive use of scatterplot matrices. A scatterplot matrix plots every column against every other column, with the result in the layout used for correlation matrices. Figure 5.2 shows a scatterplot matrix for the `datasets::trees` package.

The `datasets` package is, in an out-of-the-box installation, attached when R starts.

Interpreting Scatterplot Matrices:

For identifying the axes for each panel

- look across the row to the diagonal to identify the variable on the vertical axis.
- look up or down the column to the diagonal for the variable on the horizontal axis.

Each below diagonal panel is the mirror image of the corresponding above diagonal panel.

```
## Code used for the plot
plot(trees, cex.labels=1.5)
  # Calls pairs(trees)
```

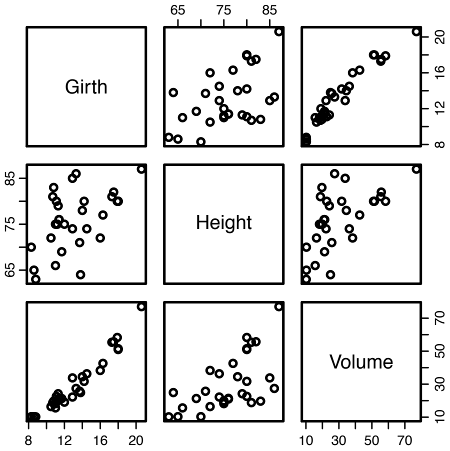


Figure 5.2: Scatterplot matrix for the `trees` data, obtained using the default `plot()` method for data frames. The scatterplot matrix is a graphical counterpart of the correlation matrix.

Notice that `plot()`, called with the dataframe `trees`, has in turn called the plot method for a data frame, i.e., it has called `plot.data.frame()` which has in turn called the function `pairs()`.

The scatterplot matrix may be examined, if there are enough points, for evidence of:

1. Strong clustering in the data, and/or obvious outliers;
2. Clear non-linear relationships, so that a correlation will underestimate the strength of any relationship;
3. Severely skewed distributions, so that the correlation is a biased measure of the strength of relationship.

The scatterplot matrix is best used as an initial coarse screening device. Skewness in the individual distributions is better checked using plots of density estimates.

5.2 *World record times for track and field events*

The first example is for world track and road record times, as at 9th August 2006. Data, copied down from the web page `http://www.gbrathletics.com/wrec.htm`, are in the dataset `DAAG::worldRecords`.

Note also the use of these data in the exercise at the end of Chapter 2 (Section 2.6.2)

Data exploration

First, use `str()` to get information on the data frame columns:

```
library(DAAG, quietly=TRUE)
```

```
str(worldRecords, vec.len=3)
```

```
'data.frame':   40 obs. of  5 variables:
 $ Distance   : num  0.1 0.15 0.2 0.3 0.4 0.5 0.6 0.8 ...
 $ roadORtrack: Factor w/ 2 levels "road","track": 2 2 2 2 2 2 2 2 ...
 $ Place      : chr  "Athens" "Cassino" "Atlanta" ...
 $ Time       : num  0.163 0.247 0.322 0.514 ...
 $ Date       : Date, format: "2005-06-14" ...
```

Distinguishing points for track events from those for road events is easiest if we use lattice graphics, as in Figure 5.3.

```
## Code
library(lattice)
xyplot(Time ~ Distance, scales=list(tck=0.5),
       groups=roadORtrack, data=worldRecords,
       auto.key=list(columns=2), aspect=1)
## On a a colour device the default is to use
## different colours, not different symbols,
## to distinguish groups.
```

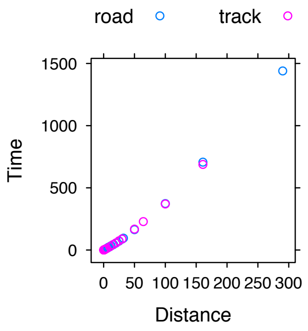


Figure 5.3: World record times versus distance, for field and road events.

Clearly increases in `Time` are not proportional to increases in `Distance`. Indeed, such a model does not make sense; velocity decreases as the length of the race increases. Proportionality when logarithmic scales are used for the two variables does make sense.

Figure 5.4 uses logarithmic scales on both axes. The two panels differ only in the labeling of the scales. The left panel uses labels on scales of $\log_e$, while the right panel has labels in the orginal units. Notice the use of `auto.key` to obtain a key.

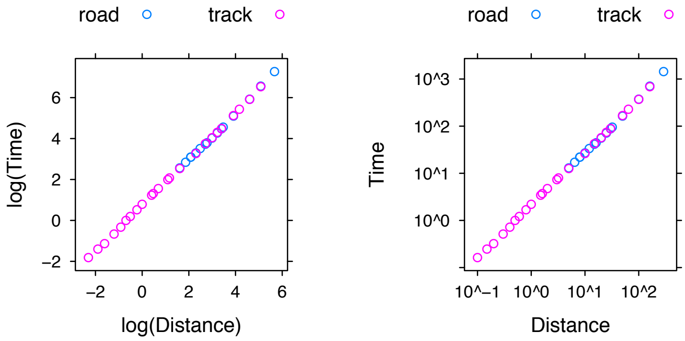


Figure 5.4: World record times versus distance, for field and road events, using logarithmic scales. The left panel uses labels on scales of $\log_e$, while in the right panel, labeling is in the orginal units, expressed as powers of 10.

```
## Code for Left panel
xyplot(log(Time) ~ log(Distance),
       groups=roadORtrack, data=worldRecords,
       scales=list(tck=0.5),
       auto.key=list(columns=2), aspect=1)
## Right panel
```

```
xyplot(Time ~ Distance, groups=roadORtrack,
       data=worldRecords,
       scales=list(log=10, tck=0.5),
       auto.key=list(columns=2), aspect=1)
```

Fitting a regression line

The plots suggest that a line is a good fit. Note however that the data span a huge range of distances. The ratio of longest to shortest distance is almost 3000:1. Departures from the line are of the order of 15% at the upper end of the range, but are so small relative to this huge range that they are not obvious.

The following uses the function `lm()` to fit a straight line fit to the logged data, then extracting the regression coefficients:

The name `lm` is a mnemonic for linear model.

```
worldrec.lm <- lm(log(Time) ~ log(Distance),
                  data=worldRecords)
coef(worldrec.lm)
```

```
  (Intercept) log(Distance)
       0.7316        1.1248
```

The equation gives predicted times:

$$\begin{aligned}\widehat{\text{Time}} &= e^{0.7316} \times \text{Distance}^{1.1248} \\ &= 2.08 \times \text{Distance}^{1.1248}\end{aligned}$$

This implies, as would be expected, that kilometers per minute increase with increasing distance. Fitting a line to points that are on a log scale thus allows an immediate interpretation.

There is no difference that can be detected visually between the track races and the road races. Careful analysis will in fact find no difference.

5.2.1 Summary information from model objects

In order to avoid recalculation of the model information each time that some different information is required, we store the result from the `lm()` calculation in the model object `worldrec.lm`.

The name `worldrec.lm` is used to indicate that this is an `lm` object, with data from `worldRecords`. Use any name that seems helpful!

Note that the function `abline()` can be used with the model object as argument to add a line to the plot of `log(Time)` against `log(Distance)`.

Plot points; add line:

```
plot(log(Time) ~ log(Distance),
     data = worldRecords)
abline(worldrec.lm)
```

Diagnostic plots

Insight into the adequacy of the line can be obtained by examining the "diagnostic" plots, obtained by "plotting" the model object. Figure 5.5 following shows the first and last of the default plots:

```
## Code
plot(worldrec.lm, which=c(1,5),
     sub.caption=rep("",2))
```

By default, there are four "diagnostic" plots. Panel A is designed to give an indication whether the relationship really is linear, or whether there is some further systematic component that should perhaps be modeled. It does show systematic differences from a line.

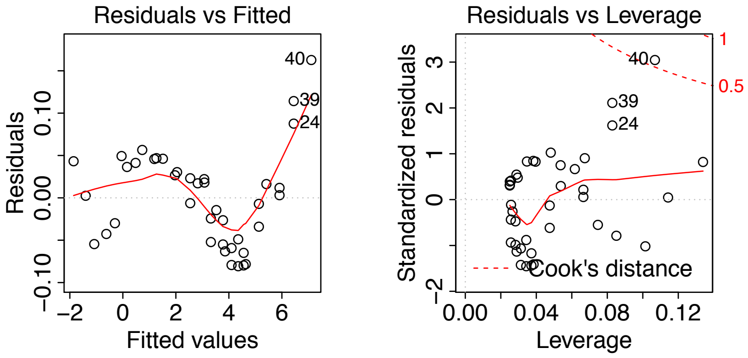


Figure 5.5: First and last of the default diagnostic plots, from the linear model for log(record time) versus log(distance), for field and road events.

The largest difference is more than a 15% difference.[1] There are mechanisms for using a smooth curve to account for the differences from a line, if these are thought important enough to model.

The plot in panel B allows an assessment of the extent to which individual points are influencing the fitted line. Observation 40 does have both a very large leverage and a large Cook's distance. The plot on the left makes it clear that this is the point with the largest fitted time. Observation 40 is for a 24h race, or 1440 min. Examine

```
worldRecords["40", ]
```

```
   Distance roadORtrack Place Time       Date
40    290.2        road Basle 1440 1998-05-03
```

5.2.2 *The model object*

Functions that are commonly used to get information about model objects are: `print()`, `summary()` and `plot()`. These are all *generic* functions. The effect of the function depends on the class of object that is printed (ie, by default, displayed on the screen) or or plotted, or summarized.

The function `print()` may display relatively terse output, while `summary()` may display more extensive output. This varies from one type of model object to another.

Compare the outputs from the following:

```
print(worldrec.lm)     # Alternatively, type worldrec.lm
```

```

Call:
lm(formula = log(Time) ~ log(Distance), data = worldRecords)

Coefficients:
  (Intercept)  log(Distance)
        0.732          1.125
```

[1] A difference of 0.05 on a scale of $\log_e$ translates to a difference of just over 5%. A difference of 0.15 translates to a difference of just over 16%, i.e., slightly more than 15%.

```
summary(worldrec.lm)
```

```

Call:
lm(formula = log(Time) ~ log(Distance), data = worldRecords)

Residuals:
    Min      1Q  Median      3Q     Max 
-0.0807 -0.0497  0.0028  0.0377  0.1627 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.73160    0.01241      59   <2e-16
log(Distance)  1.12475    0.00437     257   <2e-16

Residual standard error: 0.0565 on 38 degrees of freedom
Multiple $R^2$:  0.999,	Adjusted $R^2$:  0.999 
F-statistic: 6.63e+04 on 1 and 38 DF,  p-value: <2e-16
```

Used with `lm` objects, `print()` calls `print.lm()`, while `summary()` calls `summary.lm()`. Note that typing `worldrec.lm` has the same effect as `print(worldrec.lm)`.

Internally, `summary(wtvol.lm)` calls `UseMethod("summary")`. As `wtvol.lm` is an lm object, this calls `summary.lm()`.

5.2.3 *The `lm` model object is a list*

The model object is actually a list. Here are the names of the list elements:

```
names(worldrec.lm)
```

```
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"
```

These different list elements hold very different classes and dimensions (or lengths) of object. Hence the use of a list; any collection of different R objects can be brought together into a list.

The following is a check on the model call:

```
worldrec.lm$call
```

```
lm(formula = log(Time) ~ log(Distance), data = worldRecords)
```

Commonly required information is best accessed using generic extractor functions. Above, attention was drawn to `print()`, `summary()` and `plot()`. Other commonly used extractor functions are `residuals()`, `coefficients()`, and `fitted.values()`. These can be abbreviated to `resid()`, `coef()`, and `fitted()`.

Use extractor function `coef()`:

```
coef(worldrec.lm)
```

5.3 *Regression with two explanatory variables*

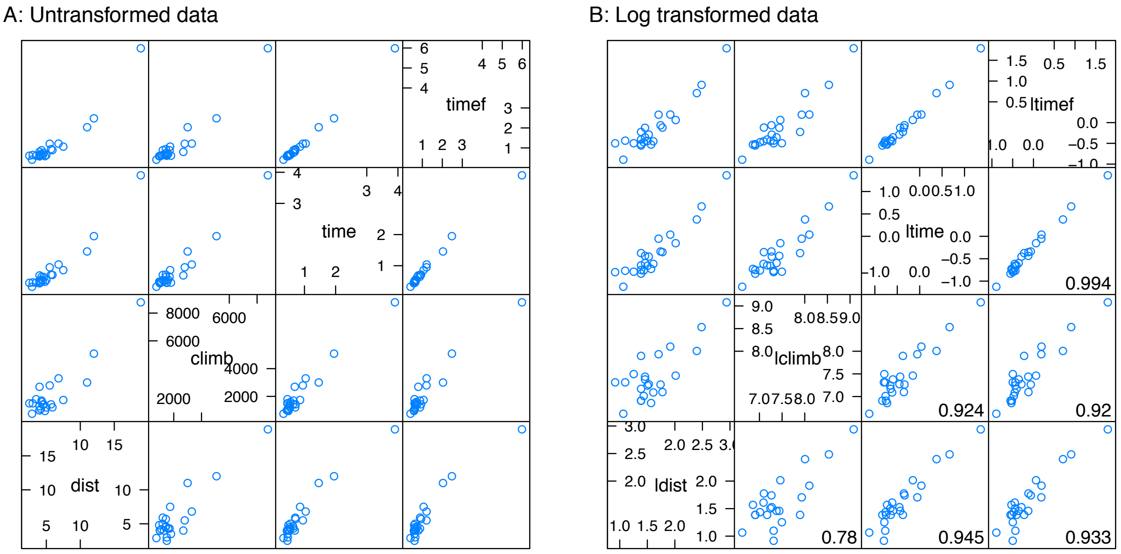
The dataset `nihills` in the *DAAG* package will be used for a regression fit in Section 11.6. This has record times for Northern Ireland mountain races. Overview details of the data are:

```
str(nihills)
```

```
'data.frame':   23 obs. of  4 variables:
 $ dist : num  7.5 4.2 5.9 6.8 5 4.8 4.3 3 2.5 12 ...
 $ climb: int  1740 1110 1210 3300 1200 950 1600 1500 1500 5080 ...
 $ time : num  0.858 0.467 0.703 1.039 0.541 ...
 $ timef: num  1.064 0.623 0.887 1.214 0.637 ...
```

Figure 5.6 uses the lattice function `splom()` (from the *lattice* package) to give scatterplot matrices, one for the unlogged data, and the other for the logged data. The left panel shows the unlogged data, while the right panel shows the logged data:

The function `splom()` is a *lattice* alternative to `pairs()`, giving a different panel layout.


A: Untransformed data

B: Log transformed data

Figure 5.6: Scatterplot matrices for the Northern Ireland mountain racing data. The left panel is for the unlogged data, while the right panel is for the logged data. Code has been added that shows the correlations, in the lower panel.

The following panel function was used to show the correlations:

```
showcorr <- function(x,y,...){
    panel.xyplot(x,y,...)
    xy <- current.panel.limits()
    rho <- paste(round(cor(x,y),3))
    eps <- 0.035*diff(range(y))
    panel.text(max(x), min(y)+eps, rho,
               pos=2, offset=-0.2)
}
```

Code for the scatterplot matrix in the left panel is:

```
## Scatterplot matrix; unlogged data
library(lattice)
splom(~nihills,  xlab="",
       main=list("A: Untransformed data", x=0,
       just="left", fontface="plain"))
```

For the right panel, create a data frame from the logged data:

```
lognihills <- log(nihills)
names(lognihills) <- paste0("l", names(nihills))
## Scatterplot matrix; log scales
splom(~ lognihills, lower.panel=showcorr, xlab="",
       main=list("B: Log transformed data", x=0,
       just="left", fontface="plain"))
```

Note that the data are positively skewed, i.e., there is a long tail to the right, for all variables. For such data, a logarithmic transformation often gives more nearly linear relationships. The relationships between explanatory variables, and between the dependent variable and explanatory variables, are closer to linear when logarithmic scales are used. Just as importantly, issues with large leverage, so that the point for the largest data values has a much greater leverage and hence much greater influence than other points on the the fitted regression, are greatly reduced.

Unlike `paste()`, the function `paste0()` does not leave spaces between text strings that it pastes together.

Notice also that the correlation of 0.913 between `climb` and `dist` in the left panel of Figure 5.6 is very different from the correlation of 0.78 between `lclimb` and `ldist` in the right panel. Correlations where distributions are highly skew are not comparable with correlations where distributions are more nearly symmetric. The statistical properties are different.

The following regresses `log(time)` on `log(climb)` and `log(dist)`:

```
nihills.lm <- lm(ltime ~ lclimb + ldist,
                 data=lognihills)
```

5.4 One-way Comparisons

The dataset `tomato` has weights of plants that were grown under one of four different sets of experimental comditions. Five plants were grown under each of the treatments:

A common strategy for getting a valid comparison is to grow the plants in separate pots, with a random arrangement of pots.

```
- water only
- conc nutrient
- 2-4-D + conc nutrient
- x conc nutrient
```

Notice that "water only" has been made the referrence level. This is done as a preferred starting point for the analysis of variance calculations that appear below. Figure 5.7, created using the function `quickplot()` from the *ggplot2* package, shows the plant weights. Are the apparent differences between treatments large enough that they can be distinguished statistically?

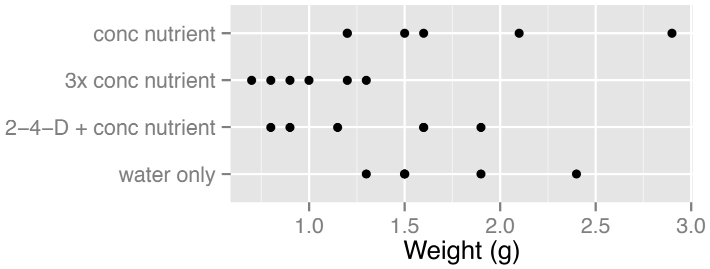


Figure 5.7: Weights (g) of tomato plants grown under four different treatments.

```
## Code
library(ggplot2)
tomato <- within(tomato, trt <- relevel(trt, ref="water only"))
quickplot(weight, trt, data=tomato, xlab="Weight (g)", ylab="")
```

The command `aov()`, followed by a call to `summary.lm()`, can be used to analyse these data, thus:

```
tomato.aov <- aov(weight ~ trt, data=tomato)
round(coef(summary.lm(tomato.aov)), 3)
```

Observe that, to get estimates and SEs of treatment effects, `tomato.aov` can be treated as an `lm` (regression) object.

```
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)             1.683      0.187   9.019    0.000
trt2-4-D + conc nutrient  -0.358      0.264  -1.358    0.190
trt3x conc nutrient    -0.700      0.264  -2.652    0.015
trtconc nutrient        0.067      0.264   0.253    0.803
```

Because we made "`water only`" the reference level, "`(Intercept)`" is the mean for `water only`, and the other coefficients are for differences from `water only`. None of the treatments can be distinguished statistically from `water only`.

Growing conditions in a laboratory or glasshouse or growth chamber — temperature, humidity and air movement — are rarely totally uniform. The preferred way to deal with this is to repeat the comparison between treatments in several different locations.[2] Conditions will vary between locations, but each comparison between treatments is conducted under the relatively uniform conditions at one particular location.

The dataset `rice` (*DAAG*) is from such experiment. There are two treatment factors — three types of fertilizer and two varieties of rice plant. The six treatment combinations are each repeated six times, in each of two blocks.

For these data, Figure 5.8 gives a clear picture of the result. For fertilizers `NH4Cl` and `NH4NO3`, any difference between the varieties

[2] Technically, each different location (or set of conditions) is known as a *block*.

is inconsequential. There is strong "interaction" between fert and variety. A formal analysis will confirm what is already rather clear. Experimenters are rarely thus fortunate.

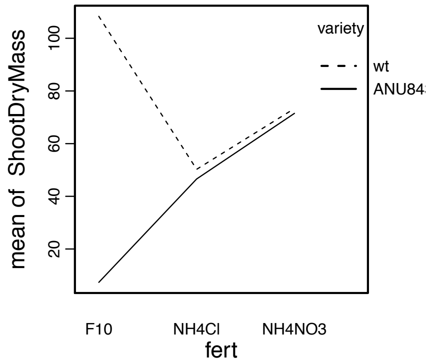


Figure 5.8: Interaction plot for the terms fert and variety, with ShootDryMass as the dependent variable. Notice that for fertilizer F10, there is a huge variety difference in the response. For the other fertilizers, there is no difference of consequence.

```
## Code
library(DAAG)
with(rice, interaction.plot(x.factor=fert,
                            trace.factor=variety,
                            ShootDryMass,
                            cex.lab=1.4))
```

5.5 Time series – Australian annual climate data

The data frame bomregions2012 from the *DAAG* package has annual rainfall data, both an Australian average and broken down by location within Australia, for 1900 – 2012. Figure 5.9 shows annual rainfall in the Murray-Darling basin, plotted against year.

Data are from the website http://www.bom.gov.au/climate/change/

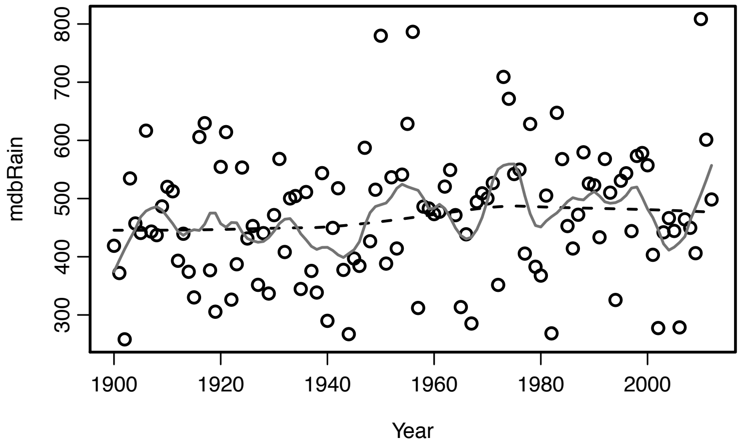


Figure 5.9: Annual rainfall in the Australian Murray-Darling Basin. by year. The lowess() function is used to The dashed curve with f=2/3 captures the overall trend, while the solid curve with f=0.1 captures trends on a scale of around eleven years. (10% of the 113 year range from 1900 to 2012 is a little more than 11 years.)

```
## Code
library(DAAG)
```

```
plot(mdbRain ~ Year, data=bomregions2012)
## Calculate and plot curve showing long-term trend
with(bomregions2012, lines(lowess(mdbRain ~ Year, f=2/3), lty=2))
## Calculate and plot curve of short-term trends
with(bomregions2012, lines(lowess(mdbRain ~ Year, f=0.1),
                           lty=1, col="gray45"))
```

The `lowess()` function has been used to fit smooth curves, formed by moving a smoothing window across the data. The dashed curve with `f=2/3` (include 2/3 of the data in the smoothing window) captures the overall trend in the data. The choice of `f=0.1` for the solid curve has the effect that somewhat more than ten years of data are used in determining each fitted value on the smooth.

For each smoothing window, a line or other simple response function is fitted. Greatest weight to points near the centre of the smoothing window, with weights tailing off to zero at the window edge.

This graph is exploratory. A next step might to model a correlation structure in residuals from the overall trend. There are extensive abilities for this. For graphical exploration, note `lag.plot()` (plot series against lagged series).

The functions `acf()` and `pacf()` might be used to examine the correlation structure in the residuals.

The cube root of average rainfall has a more symmetric distribution than rainfall. Thus, use this in preference to average rainfall when fitting models.

5.6 *Exercises*

1. Plot `Time` against `Distance`, for the `worldRecords` data. Ignoring the obvious curvature, fit a straight line model. Use `plot.lm` to obtain diagnostic plots. What do you notice?

2. The data set `LakeHuron` (*datasets* package) has mean July average water surface elevations (ft) for Lake Huron, for 1875-1972. The following reates a data frame that has the same information:

   ```
   Year=as(time(LakeHuron), "vector")
   huron <- data.frame(year=Year, mean.height=LakeHuron)
   ```

   (a) Plot `mean.height` against year.

   (b) To see how each year's mean level is related to the previous year's mean level, use

   ```
   lag.plot(huron$mean.height)
   ```

   This plots the level in each year against the level in the previous year.

   (c) *Use the function `acf()` to plot the autocorrelation function. Compare with the result from the `pacf()` (partial autocorrelation). What do the graphs suggest?

   For an explanation of the autocorrelation function, look up "Autocorrelation" on Wikepedia.