Statistics with R: Computing and Graphics

Kjell Konis
Department of Statistics
University of Oxford
konis@stats.ox.ac.uk

1 Introduction

R is an implementation of the S language that was originally developed by J. M. Chambers at AT&T's Bell Laboratories. The original S is licensed to the Insightful Corporation which sells an enhanced version called S-PLUS. S-PLUS is rather expensive and really not that much better than the the freely (GNU) available R so why not use R?

Many of the examples in this introductory course are borrowed from chapters 1 and 2 of *Modern Applied Statistics with S-PLUS* (MASS) by W. N. Venables and B. D. Ripley. Despite the title, the current version deals with both R and S-PLUS. MASS is a good investment, you should consider buying it or at least getting it from the library before someone else does.

2 Obtaining and Installing the R Software

The *R Project for Statistical Computing* website is located at

`http://www.r-project.org/`

Take a look around. In particular look at the Manuals, FAQs and Mailing Lists links. If you decide to use R for your work it will probably be beneficial to subscribe to the R-help mailing list. Feel free to ask questions there but make sure you follow the posting guidelines.

You can download a copy of R for your own computer from

`http://www.stats.bris.ac.uk/R/`

There are versions available for Windows, Linux, and Macintosh. It will also be worth while to get the *An Introduction to R* manual which you can find at

`http://www.stats.bris.ac.uk/R/manuals.html`

It provides a good reference at a level suitable for new users.

3 Getting Started

Start R. From the Start menu choose All Programs → R → R.

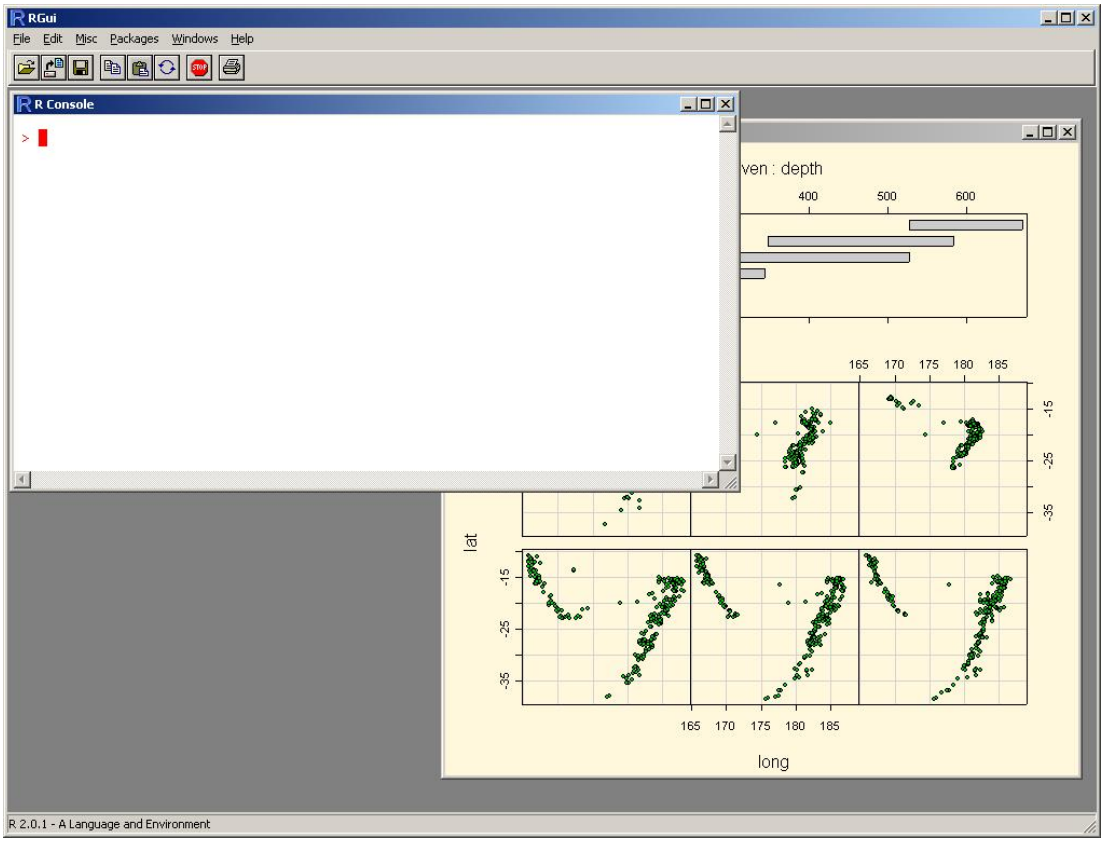

Figure 1: The R desktop

The R desktop is shown in figure 1. The sub-window in the foreground is called the commands window. R is controlled by typing commands into the commands window. The square window in the background is referred to as the graphics device – we'll see how to create graphics just a little later.

In R basic commands are generally either assignments or expressions. When an expression is given, it is evaluated, printed, then discarded. An assignment is evaluated and the value is stored as a variable but is not automatically printed. The assignment operator in R is `<-`. Note that the `>` is the R prompt. When R is ready to accept user input the last line in the commands window will display a `>`. If you press return before completing an expression R will allow you to complete it on the next line. When this happens the prompt will turn into a `+`. When you run the examples in this tutorial do not type the `>` or the `+` if it is the first character in the line.

Now lets try some basic interaction with R.

```
> 2 + 3
[1] 5
> sqrt(3/4) / (1/3 - 2/pi^2)
[1] 6.626513
> library(MASS)
> mean(chem)
[1] 4.280417
```

```
> m <- mean(chem)
> v <- var(chem)/length(chem)
> m / sqrt(v)
[1] 3.958487
```

The `[1]` at the beginning of the output lines indicates that the answer is the first element of a vector. In R all numerical values are vectors; scalars are just vectors that contain only a single element. The `library` function is used to attach collections of functions and/or data to the R session. The vector `chem` is in the MASS library; if you try to compute the mean of `chem` without first loading the MASS library, R will generate an error. Also note that R includes some useful constants, for instance `pi`. One that is a little bit tricky is e; use `exp(1)`.

4 How to use the Built in Help System

R includes an on-line help facility that can be invoked from the command line. For example, to get information on the function `var` use the command

```
> help(var)
```

or the shortcut

```
> ?var
```

which does exactly the same thing. For a feature specified by special characters and in a few other cases (one is "`function`"), the argument must must be enclosed in single or double quotes. For example, to see the help for the assignment operator `<-` use one of the following

```
> help("<-")
> ?"<-"
```

Now practice using the help facility. What do the functions `abs` and `pt` do? If you can't guess what `length`, `sqrt`, `mean`, and `var` do look them up too. The help facility works for non-function objects as well (as long as their creator has provided documentation). Use `help(chem)` or `?chem` to see the help file describing the `chem` object in the MASS library.

The `help` function works well if you know the name of the function that you want help with. If you don't know the name of the function you want help with then the situation is a little trickier. R provides a function `help.search` that can be used to search for keywords in help files. Why not use `help(help.search)` to learn how to use it? The help file for `help.search` is rather complicated so scroll down to the bottom and see if you can figure it out from the examples. Use `help.search` to find the R function for fitting linear models. Alternatively, try using Google. Which way is easier?

5 Getting Data into R

One of the trickiest parts of using R is actually getting your data into R. The most basic way (and perhaps least user friendly) is to use the `scan` function and type the data into the commands window. The Forbes data consists of seventeen measurements of the boiling point

of water and the logarithm of the atmospheric pressure made at different altitudes in the Swiss Alps. These data can be entered by using the following commands.

```
> bp <- scan()
1: 194.5 194.3 197.9 198.4 199.4 199.9 200.9 201.1 201.4 201.3 203.6 204.6
209.5 208.6 210.7 211.9 212.2
18:
Read 17 items
> lp <- scan()
1: 131.79 131.79 135.02 135.55 136.46 136.83 137.82 138 138.06 138.05
140.04 142.44 145.47 144.34 146.3 147.54 147.8
18:
Read 17 items
```

These two commands create the vectors `bp` and `lp` in your work space. In R it is generally a good idea to store your data in a structure called a data frame. This is accomplished by the following command.

```
> forbes <- data.frame(Boiling.Point = bp, Log.Pressure = lp))
```

A data frame stores the data in columns. The syntax for the above command creates a data frame with two columns named `Boiling.Point` and `Log.Pressure` and puts the vector `bp` into `Boiling.Point` and the vector `lp` into `Log.Pressure`. You can view the data frame by typing its name into the commands window and pressing return.

```
> forbes
   Boiling.Point Log.Pressure
1          194.5       131.79
2          194.3       131.79
3          197.9       135.02
...
```

Another perhaps more user friendly way to get data into R is to use the `read.table` function. The `read.table` function reads the data in a specially formatted text file and returns a data frame, all in one step. Start by downloading the file *brains.xls* from

`http://www.stats.ox.ac.uk/~konis/OUCS/`

and opening it in Microsoft Excel. This file contains the average body weights and brain weights for 28 different species. Note that the first row contains the names of the columns. The first step towards getting these data into R is to save this Excel worksheet as a tab delimited text file. From the file menu choose 'Save As ...'. In the 'Save As' dialog box there is a list at the very bottom called 'Save as type'; choose 'Text (Tab delimited)(*.txt)'. Finally, click on 'Save'. The file *brains.txt* is in a format that the R `read.table` function can understand.

Bring the R desktop back into the foreground and type the following command into the commands window (you will have to replace the ... with the appropriate path).

```
> brains <- read.table(".../brains.txt", row.names = 1, header = TRUE)
```

You can find the complete path to the file *brains.txt* by dragging its icon and dropping it into the R commands window. R will try to load it using the `load` function and this will fail but the path you want will be displayed.

```
> load("C:/Documents and Settings/konis/Desktop/brains.txt")
Error: bad restore file magic number (file may be corrupted)-- no data loaded
> brains <- read.table("C:/Documents and Settings/konis/Desktop/brains.txt",
+ row.names = 1, header = TRUE)
```

If everything works then you should have a data frame named brains in your workspace. Take a look at it by typing its name into the commands window and pressing return.

```
> brains
                     Body  Brain
Mountain.Beaver     1.350    8.1
Cow               465.000  423.0
Gray.wolf          36.330  119.5
Goat               27.660  115.0
...
```

6 Graphical Data Analysis: Box and Scatter Plots

Box plots provide a good way to take a first look at your data. So lets make box plots for the brain data.

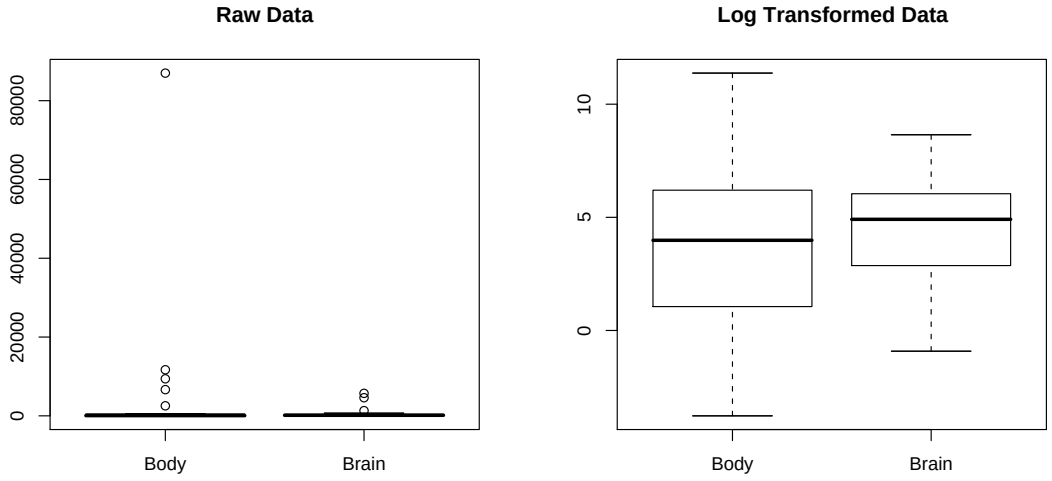

Figure 2: Box Plots of the Brain Data and the Log Transformed Brain Data

In R box plots are generated using the function `boxplot`. If your data is stored in a data frame then you can produce the left panel of figure 2 with the following command.

```
> boxplot(brains, main = "Raw Data")
```

The box plots of the untransformed (or raw) data show that these data highly skewed. One way of dealing with skewed data is to take logs. First, create a data frame `lbrains` containing the log transformed observations then generate the box plots.

```
> attach(brains)
> lbrains <- data.frame(Body = log(Body), Brain = log(Brain))
> boxplot(lbrains, main = "Log Transformed Data")
```

The box plots of the log transformed data are shown in the right panel of figure 2.

Scatter plots also provide a good tool for exploratory data analysis; especially when there are only 2 variables.

```
> plot(brains, pch = 16, main = "Raw Data")
> plot(lbrains, pch = 16, main = "Log Transformed Data")
```

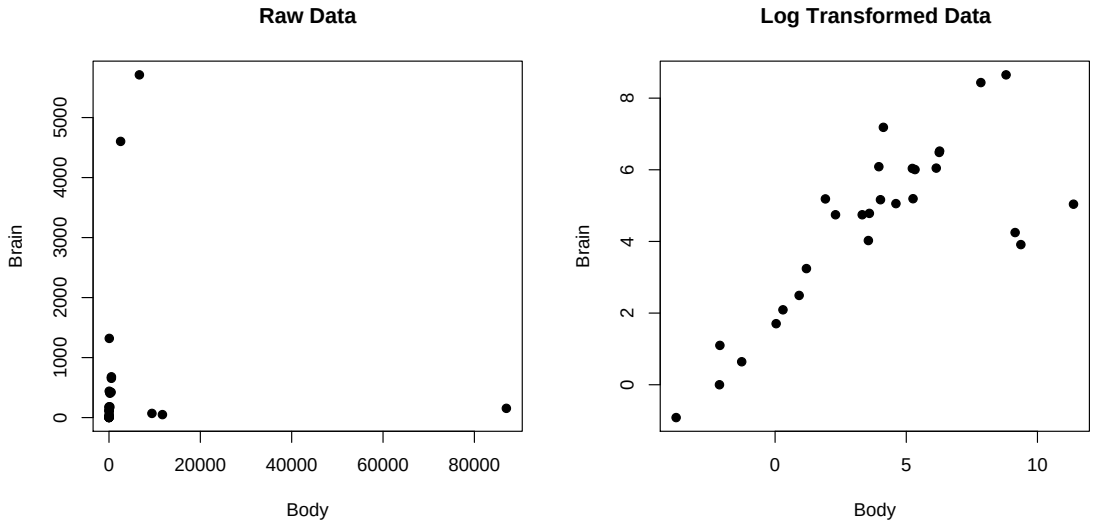

Figure 3: Scatter Plots of the Brain Data and the Log Transformed Brain Data

The scatter plot of the log transformed brain data suggests a linear relationship between log brain weight and log body weight. However, there are three points that do not appear to fit a linear model. To see which ones they are use the `identify` function.

```
> identify(lbrains, labels = row.names(brains))
```

Click on some points in the scatter plot and they will be labeled. You can choose where the label goes by clicking on the around the point. For instance, to put the label below the point click slightly below the point. When you are finished labeling points right click and select stop. The three outlying points are all dinosaurs and they are the only three dinosaurs in the data.

Making Plots Pretty

You may have already figured out that the optional argument `main` adds a title to the plot. The optional argument `pch` allows you to select the plot character. By default R plots with

open circles (see the box plots in figure 2). Using `pch = 16` causes R to plot with filled circles which I think looks better. Two other useful arguments to the plotting functions in R are `xlab` and `ylab` which allow you to specify the axis labels. The syntax is similar to that for `main`. See if you can use these commands to produce the plot in figure 4.

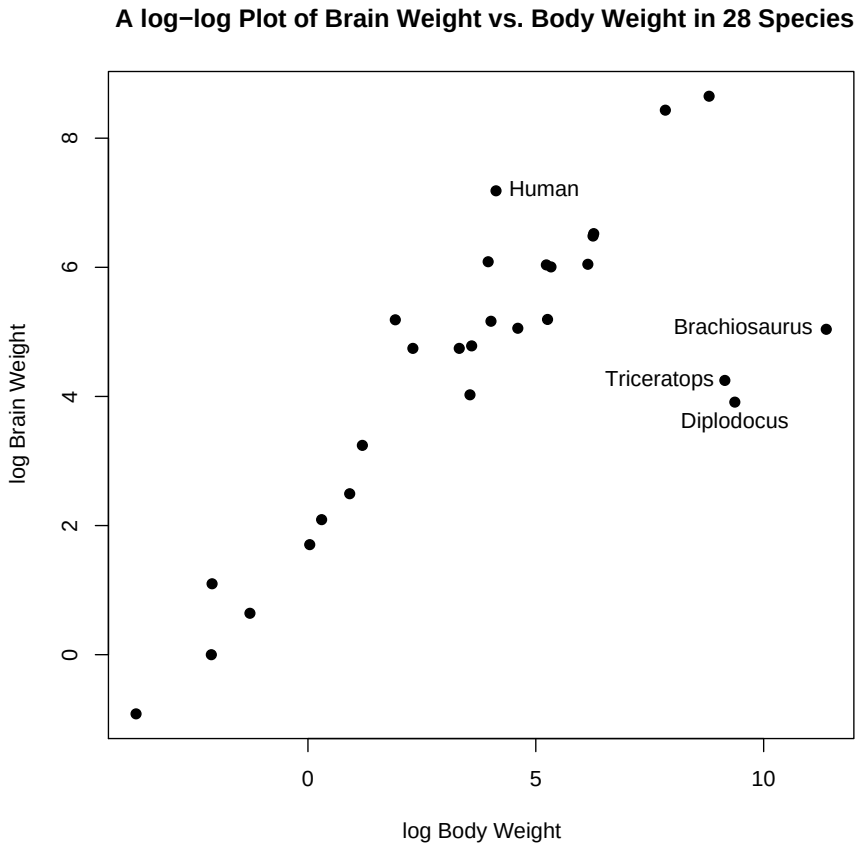

Figure 4: Hey, That's a Nice Looking Plot

Finally, when you have a plot you are happy with – for instance the plot in figure 4 – save it or copy it as a Windows metafile. To do this right click on the graphics device and choose either 'Copy as metafile' or 'Save as metafile...'. You can then either paste the plot in another program (Word or Powerpoint for example) or import it from the saved file.

7 Statistical Modeling: Linear Regression

The scatter plot of log brain weight versus log body weight suggests a linear relationship. The function for fitting linear models in R is called `lm`. It takes two arguments: formula and data. The formula is a description of the model you wish to fit in the form

Response $\sim$ *Predictors*

where the predictors are separated by `+`s if there are more than one. By default an intercept term (the b in $mx + b$) is included. Transformations of either the response or the predictors (or

both) can be included in the formula. Thus to fit a linear model to the raw data you would use the formula

```
Brain ~ Body
```

and to fit a linear model to the log transformed data you would use

```
log(Brain) ~ log(Body)
```

The data argument should be the name of a data frame containing the variables in the formula. The following command fits a linear model to the log transformed brain weight data.

```
> brains.lm <- lm(log(Brain) ~ log(Body), data = brains)
```

Note that the data argument is `brains` and not `lbrains`. This is because the log transformation is specified in the formula.

To view the parameter estimates (the slope and intercept) just type the name of the fitted model object in the commands window and press return.

```
> brains.lm

Call:
lm(formula = log(Brain) ~ log(Body), data = brains)

Coefficients:
(Intercept)    log(Body)
      2.555        0.496
```

The call is simply an image of the command that created the object. The coefficients indicate that the best fit line has slope 0.496 and y-intercept 2.555. For more detailed output use the `summary` method.

```
> summary(brains.lm)

Call:
lm(formula = log(Brain) ~ log(Body), data = brains)

Residuals:
    Min      1Q  Median      3Q     Max
-3.2890 -0.6763  0.3316  0.8646  2.5835

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.55490    0.41314   6.184 1.53e-06 ***
log(Body)    0.49599    0.07817   6.345 1.02e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.532 on 26 degrees of freedom
Multiple R-Squared: 0.6076,Adjusted R-squared: 0.5925
F-statistic: 40.26 on 1 and 26 DF,  p-value: 1.017e-06
```

There is also a `plot` method for fitted linear models. These plots are intended to help assess how well the linear model describes your data. The following command generates the plots in figure 5.

```
> plot(brains.lm)
```

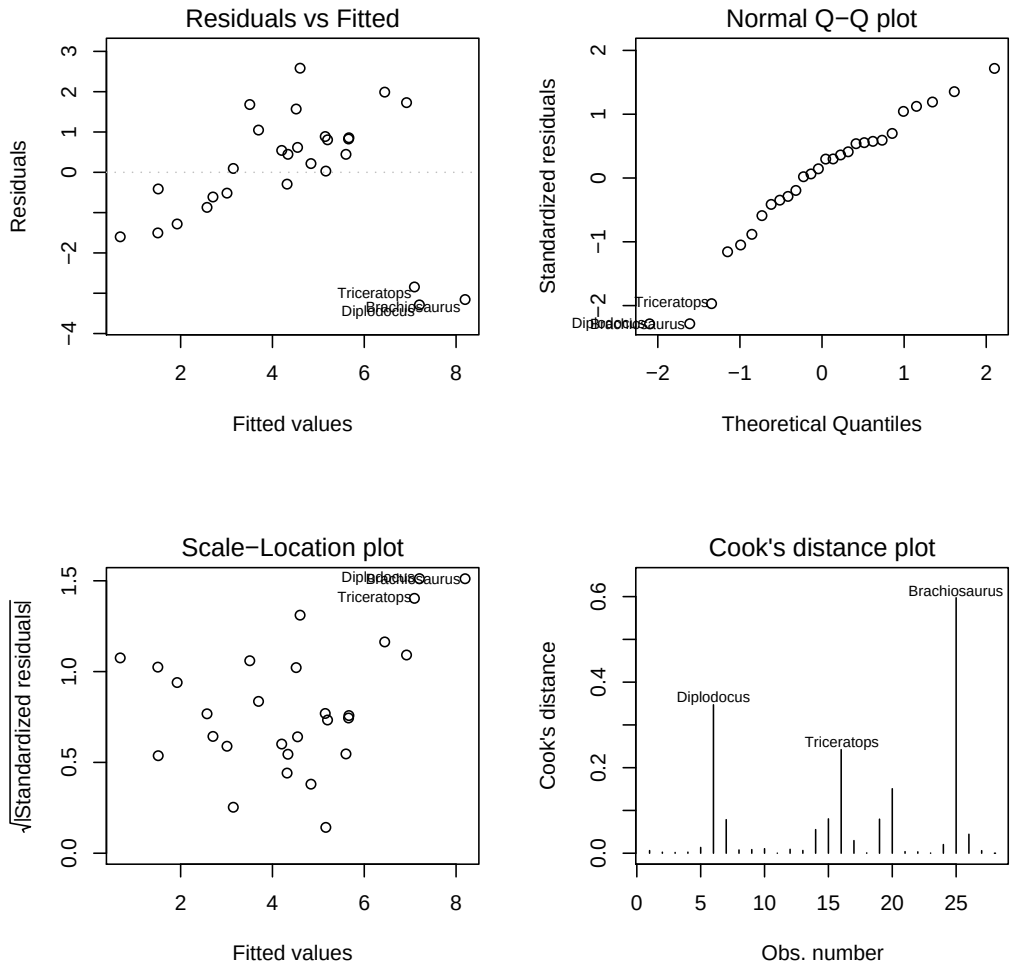

Figure 5: Diagnostic plots for linear model objects

Finally, one last useful trick is to add a best fit line to the log-log scatter plot. Start by recreating the scatter plot

```
> plot(lbrains, pch = 16,
+ main = "A log-log Plot of Brain Weight vs. Body Weight in 28 Species",
+ xlab = "log Body Weight", ylab = "log Brain Weight")
> identify(lbrains, labels = row.names(brains))
[1]  6 14 16 25
```

then include the best fit line from the fitted model.

```
> abline(brains.lm)
```

Just out of curiosity lets see how the best fit line looks if we exclude dinosaurs from the model. Observations 6, 16, and 25 are the dinosaurs. To remove them from the model use the optional argument `subset`

```
> no.dinos <- lm(log(Brain) ~ log(Body), subset = -c(6, 16, 25),
+ data = brains)
```

where the minus sign in front of the subset vector indicates that these observations should not be used. Add the no dinosaurs fit to the plot using a different line style (`lty`) and add a legend.

```
> abline(no.dinos, lty = 2)
> legend("bottomright", legend = c("Dinosours", "No Dinosaurs"),
+ lty = c(1, 2))
```

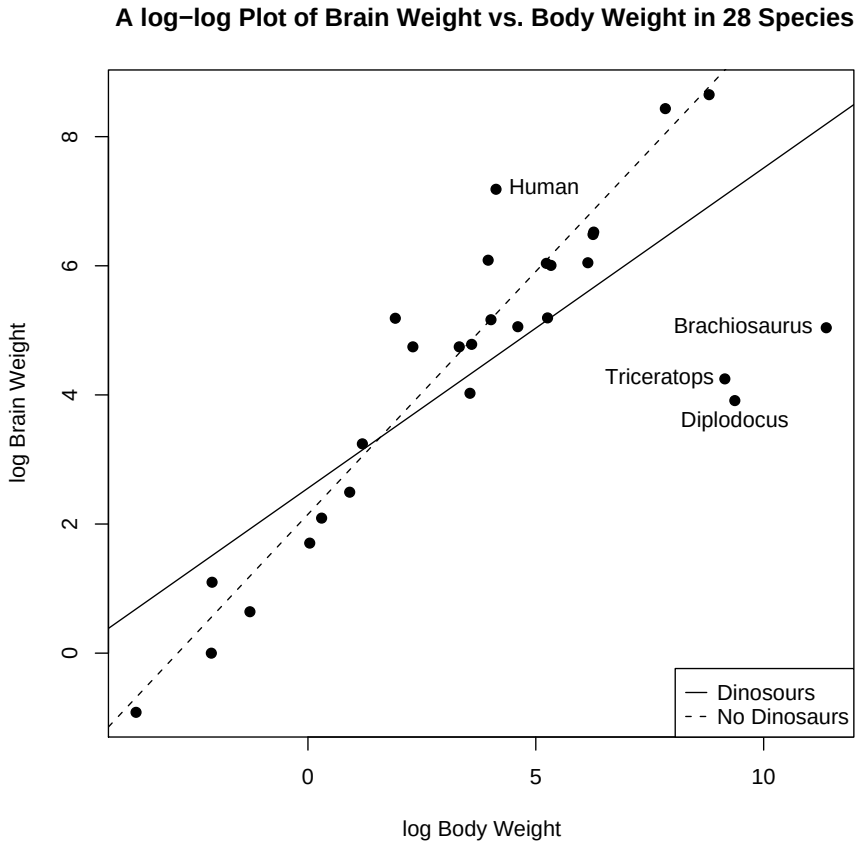

Figure 6: Linear Models

8 Extending R: Writing Your Own Functions

R can be extended by writing new functions which can then be used in the same way as built-in functions. This is very easy. For example, to define functions to compute the standard deviation and the two-tailed p-value of a t statistic we write

```
> std.dev <- function(x) sqrt(var(x))

> t.test.p <- function(x)
+ {
+    n <- length(x)
+    t <- sqrt(n) * (mean(x) - 0.0) / std.dev(x)
+    2 * (1 - pt(abs(t), n-1))
+ }
```

The return value for an R function is the last expression in the function. Make sure that the last instruction in your functions is not an assignment.

Use the command `x <- rnorm(300, 5, 2)` to generate a vector of 300 independent $N(5, 4)$ (no, that's not a typo) random numbers and store it in a variable named `x`. Then use the function `t.test.p` to test the hypothesis that $\mu = 0.0$.

9 Practice with R

This section is designed to make you familiar with R syntax and graphics. Some of the expressions which appear below will not be familiar; this is deliberate and you should access the Help facility to find out what they do and how they are used. In the four exercises which follow, the left column contains the R code and the right column explains what you are trying to achieve.

The data sets are in the files `anorexia.txt`, `hills.txt` and `moses.txt`. You will have to import them into R. Use the function `read.table`:

```
> hills.dat <- read.table("http://www.stats.ox.ac.uk/~konis/bs1/hills.txt",
+ row.names = 1, header = TRUE)
> moses.dat <- read.table("http://www.stats.ox.ac.uk/~konis/bs1/moses.txt",
+ header = TRUE)
> anorexia.dat <- read.table(
+ "http://www.stats.ox.ac.uk/~konis/bs1/anorexia.txt",
+ header = TRUE, fill = TRUE)
```

Now we use R to examine these data.

1. | | |
   |---|---|
   | `attach(moses.dat)` | Make the columns in the data frame visible as variables. |
   | `hist(Ed.Moses)` | Histogram of Ed Moses' times. |
   | `boxplot(Ed.Moses)` | Boxplot of Ed Moses' times. |
   | `x <- seq(1, length(Ed.Moses))` | Creates a vector of integers. Since the |
   | `plot(x, Ed.Moses)` | times are in chronological order, the scatterplot shows how his times varied throughout his career. How did his performance vary as he became older? |
   | `detach(moses.dat)` | Remove `moses.dat` from the search path. |

2. | | |
   |---|---|
   | `x <- rnorm(10000)` | Generate two pseudo random normal |
   | `y <- rnorm(10000)` | vectors with 10,000 entries and assign them to `x` and `y`. |
   | `hist(c(x, y+2), breaks = 25)` | Histogram of a mixture of normal distributions. Experiment with the number of bins (25) and the shift (2) of the second component. |
   | `rm(x, y)` | Remove objects no longer needed. (The clean up phase.) |

3.

```
x <- seq(1,20,0.5)
x
w <- 1 + x/2
y <- x + w*rnorm(x)

ugh <- data.frame(x,y,w)
ugh
rm(x,y,w)
fit <- lm(y ~ x, data=ugh)
summary(fit)
fit1 <- lm(y ~ x, data=ugh,
weight=1/w^2)
summary(fit1)
lrf <- loess(y ~ x, data = ugh)

attach(ugh)

plot(x,y)
```

Make $x = (1, 1.5, 2, \ldots, 19.5, 20)$ and print it.

w will be usd as a 'weight' vector and to give the standard deviations of the errors.

Make a data frame of three columns named x, y, w, and look at it. Remove the original x, y, w.

Fit a simple linear regression of y on x and look at the analysis.

Since we know the standard deviations, we can do a weighted regression.

Fit a smooth regression curve using a modern regression function.

Make the columns in the data frame visible as variables.

Make a standard scatterplot. To this plot we shall add the three regression lines (or curves) as well as the known true line.

`lines(spline(x,fitted(lrf)))`	First add in the local regression curve using a spline interpolation between the calculated points.
`abline(0, 1, lty=3)`	Add in the true regression line, with intercept 0, slope 1 and a different line type.
`abline(fit)`	Add in the unweighted regression line. `abline()` is able to extract the information it needs from the fitted regression object.
`abline(fit1, lty=4)`	Finally add in the weighted regression line using line type 4. This one should be the most accurate estimate, but may not be, of course.
	You should be able to make a hard copy of the graphics window by selecting the `Print` option from the menu.
`plot(fitted(fit), residuals(fit), xlab='Fitted Values', ylab='Residuals')`	A standard regression diagnostic plot to check for unequal variances. The data have been generated from a heteroscedastic (unequal variances) process and you should be able to detect this.
`qqnorm(resid(fit))` `qqline(resid(fit))`	A normal scores plot to check for skewness, kurtosis and outliers. (Note that the heteroscedasticity may show as apparent non-normality.)
`detach(ugh)` `rm(fit,fit1,lrf,ugh)`	Remove the data frame from the search path and clean up again.

4. Now we look at another real data set on record times of Scottish hill races against distance and total height climbed.

`hills.dat`	List the data
`pairs(hills.dat)`	Show a matrix of pairwise scatterplots
`attach(hills.dat)`	Make columns available by name.
`plot(dist, time)`	Use the left mouse button to identify
`identify(dist, time,`	outlying points. Their row numbers are
`row.names(hills.dat))`	returned. Use the right-hand mouse button to quit.
`abline(lm(time ~ dist))`	Show the least squares regression line.
`detach(hills.dat)`	Clean up again.

5. Finally, look at the data file `anorexia.dat`. It contains data on weights, in kg, of young girls receiving three different kinds of anorexia treatment over a fixed period of time. The control group received the standard treatment (STD), the second group received cognitive behavioural treatment (CBT) and the third received family therapy (FT). Weights were measured before and after treatment, so that STDb and STDa refer to the respective before and after measurements for the standard treatment, and so on. The objective is to compare the three treatments.

   (i) Look at the three scatterplots of *after/before*. What preliminary conclusions can you suggest?

   (ii) Construct three vectors of *before* $-$ *after* differences. Produce comparative boxplots.

   (iii) Use t-tests to assess whether or not each treatment produces a significant difference. Is this a sensible way to proceed?

   (iv) Write a function which will use a pooled variance estimate to carry out t-tests for comparison of the control group with the other treatments. Produce a normal scores plot to investigate the residuals for normality. What conclusions can you draw?