Linear Statistical Models, R-tutorial Fall 2009

Lab 1: Introduction, Plotting, Data manipulation

If you have never used Splus or R before, check out these texts and help pages;
`http://cran.r-project.org/doc/manuals/R-intro.html`,
`http://pages.pomona.edu/ jsh04747/courses/R.pdf`. There are many more tutorials to be found on-line.
Do you need to use R? No, you can use e.g. matlab if you like, but I will write the labs for and in R. R is a good package to know. It is free, it is the program most statisticians publish their code and packages in, and it is very flexible.
For help with installing R on your computer, check out `http://cran.us.r-project.org/`. You can download R for windows, linux and unix at this site, as well as the add-on packages. There is an extensive manual text (pdf-file), though I would regard this as a reference, not a text to study.

1 Getting Started

START: To start R on your computers, find the R icon and click on it or initiate from the program list. To be able to save your work, go to the file menu and scroll down to change directories. First create a new directory for the lab called e.g. LAB1. Then change the directory to the LAB1 directory, or you can browse until you find the directory you want to work from. If you're running on linux or unix, simply call R from the prompt. R will be running from the current directory.
HELP: You can always get information about a command by typing `help(command)` at the prompt. If you don't know the command name, try `help.search(''phrase'')`.
EDITORS: For the projects, and the labs, it is best to use an editor so you don't have to retype the commands. You can always cut-and-paste into the R command window. A better alternative is to write a series of commands in your editor of choice, and save the file as "file.r". You can execute the commands in the file by writing
`source(''file.r'')`
at the prompt. The extension `".r"` is just what I use - you can use any extension you want, but it usually a good idea to use a particular file extension for code and another for e.g. report files. I use "emacs", but you can use wordpad, notepad or any other editor of choice. R has its own editor as well, which you can access under the "files" menu.
PLOTS AND GRAPHICS: In R you can open an empty graphics window with the command `x11()`. Under windows you don't have to open the graphics window, it will open automatically when you plot something. If you want to display multiple plots in the graphics window, use commands `par(mfrow=c(m,n))` or `split.screen(c(m,n))` to divide into m by n small graphics displays. Read the help files on these commands.
ENDING THE SESSION: You quit R by typing q() at the prompt, or go to "exit" under the file menu. You can elect to save the workspace in which case all functions you've created during the lab will be available when you start a new session. The results and functions are stored in a file called .RData. To review the commands you issued in a session look at the file .Rhistory. When you start another session you can launch R from the directory of choice, or change to

this directory once you've got R running. In R, go to the file menu and load workspace. You can browse to the directory of choice and mark the .RData file you wish to load. Note, you can save workspaces this way too. Note also that you can load multiple workspaces to combine work from different directories and sessions.

2 Data Manipulation

Let's start by revisiting the data from lecture: the animal sleep data. The data is available at the class home page under "labs". The file name is `sleeptab`, which you can save as a tab-del or text file. Check if an extension to the file name, like .txt, is added when you save.
There are several variables: species name, body (bwt) and brain weight (brwt) (in kilograms and grams respectively), hours of sleeping *not* dreaming, hours dreaming, hours of sleep total, maximum lifespan, gestation, predindex (an index denoting if the animal is a predator (low)or prey (high)), sleepexp (sleep exposure index, high=exposed, low=not exposed), a combination of the latter two into a "danger index", low=not in danger from other animals, high=in danger from other animals.
Load this data set into R by calling the function `read.table`. If you open the data in an editor first (e.g. `emacs sleeptab`) you can see that the first row of the data table consists of variable names. To take this into account when loading the data use the option `header=T`.At the R prompt call

```
sleeptab<-read.table(''sleeptab'',header=T)
names(sleeptab)
```

(note; maybe "sleeptab.txt" above if you saved as a text file). The last function call gives you the name of the data matrix. You can see that I used various abbreviations for the variables. Note that missing values are denoted by -999. (Before continuing, review the help files for `read.table, read, scan`).

3 Plotting

3.1 Saving a plot

To save a plot, activate the graphics window by clicking on the plot itself. Under the file menu you can now find the option to save the figure in various formats, e.g. pdf, postscript or jpg.

To access a variable in a data frame like "sleeptab" you use the dollar sign $.

```
bwt<-sleeptab$bwt
brwt<-sleeptab$brwt
par(mfrow=c(2,1))
hist(bwt,xlab="Body weight, kg (log)")
hist(brwt,xlab="Brain weight, g (log)")
```

The command `x11()` opens a graphics window that you can split into multiple rows and columns using the `par(mfrow=c(m,n))` command. Here I split the window into 2 rows, 1 column. The histograms of body weight and brain weight look very skewed. Why is that?

Try transforming the data to (i) compress the scale of the data and (ii) symmetrize it. Taking logs is a standard approach.

```
par(mfrow=c(2,1))
hist(log(bwt),xlab="Body weight, kg")
hist(log(brwt),xlab="Brain weight, g")
```

The log transform symmetrized the data somewhat, in fact it looks almost normal. You can check normality using a QQplot.

```
par(mfrow=c(1,1))
qqnorm(log(brwt),main="QQplot of log brain weight")
qqline(log(brwt))
```

Comment on the appearance of the QQplot (long tailed/short tailed compared to normal).
Simulate some normal data and check the QQplot. Read the help file on `rnorm`. Simulate three data sets, with 50, 100 and 500 observations each. Plot a histogram and a QQplot for each and comment. Now try a t-distribution with three degrees of freedom (`rt`). What do the QQplots indicate now?

4 Summarizing data

You should be aware of several numerical summaries: mean, median, trimmed mean, etc. These are examples of location estimates. To summarize the spread in the data we use the standard devation (SD), or IQR (interquartile range), MAD (median absolute deviation). We will return to robust statistics later in the class, but let's try things out using the brain weight data. We will contaminate the data, one observation only, and see how this affects the data summaries. Let's first copy the brain weight data.

```
brwt2<-brwt
```

Now contaminate the first observation by setting it to 100 times the maximum value of the brain weight data. In order to do this you need to understand how to manipulate vectors in R. A vector element can be called with square brackets and renamed by the assign `<-` operator.

```
brwt2[1]<-max(brwt)*100
```

(Before you go on; figure out how you would assign new values to the first 3 components of the vector, how about all the odd components? `[1:3]`, `[seq(1,length(brwt2),by=2)]`)
Now compute the mean and SD of the two brain weight data sets (brwt, brwt2) using the functions `mean` and `sd`. The median is a more robust location summary, as are the `mad` and `IQR` of the spread. Read the help files for these functions, apply them and discuss the results.

5 Regression

Let's try our hand at linear modelling. Plot the brain weight on body weight and comment on the results:

```
par(mfrow=c(1,1))
plot(log(bwt),log(brwt),xlab="Body weight (log)",ylab="Brain weight (log)")
```

Looks nice and linear. Let's try modelling this data set using least squares (or normal error likelihood, more later in class). We use the function `lm` for linear models. Read the help file on `lm`.

```
mod1<-lm(log(brwt)~ log(bwt))
summary(mod1)
lines(log(bwt),mod1$fitted)
```
What do you think of the fit of the model? The line is the fitted values, which come pretty close to the observed data (open circles). A good diagnostic tool is the residual plot:
```
par(mfrow=c(2,1))
plot(log(bwt),mod1$res,xlab="Body weight(log)",ylab="Residuals",main="Diagnostic
plots")
plot(mod1$fitted,mod1$res,xlab="Brain weight(log)",ylab="Residuals")
```
You want to plot the residuals against the fitted values and predictors to see if all the dependency structure has been accounted for.
Single observations can also influence the fitting of a linear model. Which observations play a major role? Well, observations toward the edges of the predictor variables have a lot of influence on the slope of the regression line, as do extreme observations. We want to identify outliers in the data set; i.e. observations that don't fit the model well. That's relatively easy: the residual plots should tell us this directly. However, other troublesome observations may pull the regression line toward them, and not show up as large residuals. Such observations may be identified by the "leverage".
```
lmi<-lm.influence(mod1)
par(mfrow=c(3,1))
plot(log(bwt),lmi$hat,xlab="Body weight (log)", ylab="Leverage")
plot(log(bwt),lmi$coeff[,2],xlab="Body weight (log)", ylab="change in slope")
plot(log(bwt),lmi$sigma,xlab="Body weight (log)", ylab="Residual MSE when obs
is dropped")
```
Above are three influence measures: the leverage, the change in the slope due to the removal of one observation plotted against that observation, and the change in residual sum of squares due to the removal of one observation. You can try contaminating the data in various way like we did in class to check how the residual and influence plots change.
It's often informative to identify outliers by name. Plot brain weight on body weight again.
```
plot(log(bwt),log(brwt))
identify(log(bwt),log(brwt),sleeptab[,1])
```
Use the left mouse button to identify points of interest and the right to quit. See any interesting observations? Which animals stand out?

6 Boxplots

Boxplots are very useful for (i) comparing variables, (ii) summarizing data that are a mix of continuous and categorical.
```
dream<-sleeptab$dream
totsleep<-sleeptab$totsleep
danger<-sleeptab$dang
propdream<-dream/totsleep
propdream[dream==-999 | totsleep==-999]<--999
par(mfrow=c(1,1))
boxplot(propdream[propdream!=-999]~ danger[propdream!=-999],xlab="Danger
```

`index",ylab="Prop of sleep hrs dream")`
Here I've plotted a boxplot of the proportion of hours an animal dreams out of the total number of hours it sleeps, categorized by the danger index. A regular boxplot that doesn't split the data this way is obtained by simply calling `boxplot(propdream[propdream!=-999])`.
Note, `[propdream!=-999]` is a vector command that identifies the part of the vector propdream that doesn't have an entry equal to -999 (the `!=` command).
Read the help file on `boxplot`.