Datasets Used in Course: 'Statistical Learning and Data Mining With R'

John Maindonald

December 16, 2010

The following provides guidance in gaining familiarity with datasets that are used in the examples in the notes. This is an incomplete list, and I will add to it in the next week or two.

The first obvious step, in each case, is to look through the help page for the dataset. The `str()` function will give summary information about the dataset. After that, you might like to try the plots that are suggested. This will help in gaining familiarity with code that can be used for producing graphs in R.

Type the code that follows the '> prompt.

`worldRecords`: ***DAAG***

Enter `help(worldRecords)` to view the help page for this dataset.

Hereafter, it will be taken for granted that you know to look at the help page.

```
> library(DAAG)
>   # NB: Datasets in the DAAG package are available once the package
>   # has been attached.
>   # Other packages, e.g., Ecdat, may require use of data() to make
>   # a dataset available.
> ## Show summary information about the data
> str(worldRecords)
```

```
'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 1 1.5 ...
 $ roadORtrack: Factor w/ 2 levels "road","track": 2 2 2 2 2 2 2 2 2 2 ...
 $ Place      : chr  "Athens" "Cassino" "Atlanta" "Pretoria" ...
 $ Time       : num  0.163 0.247 0.322 0.514 0.72 ...
 $ Date       :Class 'Date'  num [1:40] 12948 4889 9709 11040 10829 ...
```

```
> ## Plot data
> plot(Time ~ Distance, data=worldRecords)
```

`cricketer`: ***DAAG***

```
> library(DAAG)  ## Not needed, if you typed library(DAAG) earlier on
>   # NB: Datasets in the DAAG package are available once the package
>   # has been attached.
>   # Other packages, e.g., Ecdat, may require use of data() to make
>   # a dataset available.
> ## Show summary information about the data
> str(worldRecords)
```

```
'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 1 1.5 ...
 $ roadORtrack: Factor w/ 2 levels "road","track": 2 2 2 2 2 2 2 2 2 2 ...
 $ Place      : chr  "Athens" "Cassino" "Atlanta" "Pretoria" ...
 $ Time       : num  0.163 0.247 0.322 0.514 0.72 ...
 $ Date       :Class 'Date'  num [1:40] 12948 4889 9709 11040 10829 ...
```

fgl: ***MASS***

```
> library(MASS)
>   # NB: Datasets in the MASS package are available once the package
>   # has been attached.
> ## Show summary information about the data
> str(fgl)
> ## Show scatterplot matrix
> plot(fgl)
```

Here is a more informative type of scatterplot matrix:

```
> library(car)
```

Try also a plot that uses separate colours and characters for different groups in the data.

```
> scatterplot.matrix(~ . | type, smooth=TRUE, reg.line=NA, data=fgl)
```

The default colour palette is not however very satisfactory. A better choice may be:

```
> library(RColorBrewer)
> scatterplot.matrix(~ . | type, smooth=TRUE, reg.line=NA, data=fgl,
                     col=brewer.pal(n=7, name="Set1"))
```

A note on scatterplot matrices

A scatterplot matrix, which plots every column against every other column and shows the result in the layout used for correlation matrices, is useful for an initial look at the data. The scatterplot matrix is a graphical counterpart of the correlation matrix.

For identifying the axes for each panel

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

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.

nihills: ***DAAG***

This dataset has record times for Northern Ireland mountain races, for males and females separately.

```
> ## Check the contents of the various columns
> str(nihills)
```

```
'data.frame':        23 obs. of  5 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 ...
 $ gradient: num  232 264 205 485 240 ...
```

```
> ## Scatterplot matrix -- Plot each column against each other column
> plot(nihills)
```

Sugar yield data

	weight	trt
1	82.00	Control
2	97.80	Control
3	69.90	Control
4	58.30	A
. . .		

Table 1: The table has the first few lines of the data frame `sugar`.

`roller`: ***DAAG***

The data has lawn depression for various weights of lawn roller. Type `help(roller)` to see the help page for this dataset.

Here, code is shown without output.

```
> library(DAAG)
> ## Show summary information about the data
> str(roller)
> ## Plot depression against weight
> plot(depression ~ weight, data=roller)
```

`sugar`: ***DAAG*** package

The `sugar` data frame (*DAAG* package) compares the amount of sugar obtained from an unmodified wild type plant with the amounts from three different types of genetically modified plants. Table 1 shows the first few lines of data.

The code used to fit the model is:

```
> library(DAAG)        # sugar is in DAAG package
> ## Examine data
> sugar
```

```
   weight     trt
1    82.0 Control
2    97.8 Control
3    69.9 Control
4    58.3       A
5    67.9       A
6    59.3       A
7    68.1       B
8    70.8       B
9    63.6       B
10   50.7       C
11   47.1       C
12   48.9       C
```

```
> ## Summary information about data
> str(sugar)
```

```
'data.frame':        12 obs. of  2 variables:
 $ weight: num  82 97.8 69.9 58.3 67.9 59.3 68.1 70.8 63.6 50.7 ...
 $ trt   : Factor w/ 4 levels "Control","A",..: 1 1 1 2 2 2 3 3 3 4 ...
```

`cuckoos`: ***DAAG*** package

Type `help(cuckoos)` to see the help page for this dataset. A good plot for these data is:

```
> ## Get details of data
> str(cuckoos)

'data.frame':        120 obs. of  4 variables:
 $ length : num  21.7 22.6 20.9 21.6 22.2 22.5 22.2 24.3 22.3 22.6 ...
 $ breadth: num  16.1 17 16.2 16.2 16.9 16.9 17.3 16.8 16.8 17 ...
 $ species: Factor w/ 6 levels "hedge.sparrow",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ id     : num  21 22 23 24 25 26 27 28 29 30 ...

> ## Plot data
> dotplot(species ~ length+breadth, data=cuckoos, outer=TRUE,
          scale=list(x=list(relation="free")))
```

The `length+breadth` part of the formula results in separate plots (the argument `outer=TRUE` ensures plots in separate panels) for each of `length` and `breadth`.

A note on factors: The names for the different values that a factor can take are the "levels".

```
> levels(cuckoos$species)     # column 'species' from the data frame 'cuckoos'

[1] "hedge.sparrow" "meadow.pipit"  "pied.wagtail"  "robin"
[5] "tree.pipit"    "wren"
```

Internally, factors are stored as integer values. The column `species` of the data frame `cuckoos` is a factor that has 6 levels. A lookup table is used to associate levels with these integer values.

`Electricity`: *Ecdat* package

In the *Ecdat* package, datasets do not automatically become available when you use `library(Ecdat)` to attach the package. Hence the use of `data(Electricity)` in the code that follows:

```
> library(Ecdat)
> data(Electricity)     # For datsets in the 'Ecdat' package, use
>                       # data() as required to make datasets available.
> ## Get details of columns in the data frame
> str(Electricity)

'data.frame':        158 obs. of  8 variables:
 $ cost: num  0.213 3.043 9.406 0.761 2.259 ...
 $ q   : num  8 869 1412 65 295 ...
 $ pl  : num  6869 8373 7961 8972 8218 ...
 $ sl  : num  0.3291 0.103 0.0891 0.2802 0.1772 ...
 $ pk  : num  64.9 68.2 40.7 41.2 71.9 ...
 $ sk  : num  0.42 0.291 0.157 0.128 0.162 ...
 $ pf  : num  18 21.1 41.5 28.5 39.2 ...
 $ sf  : num  0.251 0.606 0.754 0.592 0.661 ...

> ## Examine scatterplot matrix
> plot(Electricity)
```

An alternative that gives more information is:

```
> library(car)
> scatterplot.matrix(Electricity, smooth=TRUE, reg.line=NA,
                     col=c("black","gray40"))
```

Be sure to look at the help page for `Electricity` (`help(Electricity)`) to get details of the variables.

`Crime`: *Ecdat* package

```
> library(Ecdat)
> data(Crime)
> str(Crime)
```

You can try

```
> plot(Crime)
```

Because however there are so many columns, this may not be satisfactory. Density plots for the columns that have continuous variables are however perfectly feasible:

```
> library(lattice)
> contnums <- (1:ncol(crime))[-c(1:2,11:12)]
> formCont <- formula(paste("~", paste(names(Crime)[contnums], collapse="+")))
> densityplot(formCont, data=Crime, outer=TRUE,
              scales=list(x=list(relation="free"), y=list(relation="free")))
```

`Wages`: *Ecdat* package

Here, code is shown without output.

```
> library(Ecdat)
> data(Wages)
> str(Wages)
> library(lattice)
> splom(Wages[, c(1,2,10,12)], alpha=0.4)
```

Use `splom()` (*lattice*) rather than `plot()` because this makes it easier to adjust the transparency; the argument `alpha` does this. Set `alpha` to be any value between 0 (full transparancy) and 1 (totally opaque).

`bronchit`: *SMIR* package

Again, code is shown without output.

```
> library(SMIR); data(bronchit)
> data(bronchit)
> str(bronchit)
> library(lattice)
> xyplot(poll ~ cig, groups=r, auto.key=list(columns=2),
         xlab="# cigarettes per day", ylab="Pollution",
         data=bronchit)
```

`nassCDS`: *DAAG* package

Code is shown without output.

```
> library(DAAG)
> str(nassCDS)
```

`Fair`: : *Ecdat* package

Code is shown without output.

```
> library(Ecdat)
> data(Fair)
> str(Fair)
```

`diabetes`: : *mclust* package

Code is shown without output.

```
> library(mclust)
> data(diabetes)
> str(diabetes)
> scatterplot.matrix(~ glucose +insulin+sspg | class, smooth=TRUE,
                     reg.line=NA, data=diabetes,
                     col=brewer.pal(n=4, name="Set1"))
```

`germandata`: : *nws* package

Code is shown without output.

```
> library(nws)
> data(germandata)
> str(germandata)
> sapply(germandata, range)  # Check range of values in each column
> scatterplot.matrix(~ X6 + X12 + jitter(X5) + jitter(X5.1) + X67 | X1.2,
                     smooth=TRUE, reg.line=NA, data=germandata,
                     col=brewer.pal(n=4, name="Set1"))

> options(continue="+ ")
```