2

Dataframes

Learning how to handle your data, how to enter it into the computer, and how to read the data into R are amongst the most important topics you will need to master. R handles data in objects known as dataframes. A dataframe is an object with rows and columns (a bit like a two-dimensional matrix). The rows contain different observations from your study, or measurements from your experiment. The columns contain the values of different variables. The values in the body of the dataframe can be numbers (as they would be in as matrix), but they could also be text (e.g. the names of factor levels for categorical variables, like ‘male’ or ‘female’ in a variable called ‘gender’), they could be calendar dates (like 23/5/04), or they could be logical variables (like ‘true’ or ‘false’). Here is a spreadsheet in the form of a dataframe with seven variables, the left-most of which comprises the row names, and other variables are numeric (area, slope, soil pH and worm density), categorical (field name and vegetation) or logical (damp is either true = T or false = F).

Field name	Area	Slope	Vegetation	Soil pH	Damp	Worm density
Nash’s Field	3.6	11	Grassland	4.1	F	4
Silwood Bottom	5.1	2	Arable	5.2	F	7
Nursery Field	2.8	3	Grassland	4.3	F	2
Rush Meadow	2.4	5	Meadow	4.9	T	5
Gunness’ Thicket	3.8	0	Scrub	4.2	F	6
Oak Mead	3.1	2	Grassland	3.9	F	2
Church Field	3.5	3	Grassland	4.2	F	3
Ashurst	2.1	0	Arable	4.8	F	4
The Orchard	1.9	0	Orchard	5.7	F	9
Rookery Slope	1.5	4	Grassland	5	T	7
Garden Wood	2.9	10	Scrub	5.2	F	8
North Gravel	3.3	1	Grassland	4.1	F	1
South Gravel	3.7	2	Grassland	4	F	2
Observatory Ridge	1.8	6	Grassland	3.8	F	0
Pond Field	4.1	0	Meadow	5	T	6
Water Meadow	3.9	0	Meadow	4.9	T	8

Statistics: An Introduction using R M. J. Crawley
© 2005 John Wiley & Sons, Ltd ISBNs: 0-470-02298-1 (PBK); 0-470-02297-3 (PPC)

Field name	Area	Slope	Vegetation	Soil pH	Damp	Worm density
Cheapside	2.2	8	Scrub	4.7	T	4
Pound Hill	4.4	2	Arable	4.5	F	5
Gravel Pit	2.9	1	Grassland	3.5	F	1
Farm Wood	0.8	10	Scrub	5.1	T	3

Perhaps the most important thing about analysing your own data properly is getting your dataframe absolutely right. The expectation is that you will have used a spreadsheet like Excel to enter and edit the data, and that you will have used plots to check for errors. The thing that takes some practice is learning exactly how to put your numbers into the spreadsheet. There are countless ways of doing it wrong, but only one way of doing it right – and this way is not the way that most people find intuitively to be the most obvious.

The key thing is this: **all the values of the same variable must go in the same column**. It does not sound like much, but this is what people tend to get wrong. If you had an experiment with three treatments (control, pre-heated and pre-chilled), and four measurements per treatment, it might seem like a good idea to create the spreadsheet like this:

Control	Pre-heated	Pre-chilled
6.1	6.3	7.1
5.9	6.2	8.2
5.8	5.8	7.3
5.4	6.3	6.9

However, this is not a dataframe, because values of the response variable appear in three different columns, rather than all in the same column. The correct way to enter these data is to have two columns: one for the response variable and one for the levels of the experimental factor (control, pre-heated and pre-chilled). Here are the same data, entered correctly as a dataframe

Response	Treatment
6.1	Control
5.9	Control
5.8	Control
5.4	Control
6.3	Pre-heated
6.2	Pre-heated
5.8	Pre-heated
6.3	Pre-heated
7.1	Pre-chilled
8.2	Pre-chilled
7.3	Pre-chilled
6.9	Pre-chilled

A good way to practice this layout is to use the Excel function called Pivot Table (found under the Data tab on the main menu bar) on your own data; it requires your spreadsheet to be in the form of a dataframe, with each of the explanatory variables in its own column.

Once you have made your dataframe in Excel and corrected all the inevitable data-entry and spelling errors, then you need to save the dataframe in a file format that can be read by R. Much the simplest way is to save all your dataframes from Excel as tab-delimited text files: File/Save As/... then from the 'Save as type' options choose 'Text (Tab delimited)'. There is no need to add a suffix, because Excel will automatically add '.txt' to your file name. This file can then be read into R directly as a dataframe, using the read.table function.

It is important to note that read.table would fail if there were any spaces in any of the variable names in row 1 of the dataframe (the header row) like Field name, Soil pH or Worm density, or between any of the words within the same factor level (as in many of the Field names). We should replace all these spaces by dots '.' before saving the dataframe in Excel (use Edit/Replace with " " replaced by "."). Now the dataframe can be read into R. There are three things to remember:

- the whole path and file name needs to be enclosed in double quotes: "c:\\abc.txt",
- header = T says that the first row contains the variable names,
- always use double backslash \\ rather than \ in the file path definition.

Think of a name for the data frame (say 'worms' in this case). Now use the **gets arrow** < – which is a composite symbol made up of the two characters < (less than) and – (minus) like this

```
worms < -read.table("c:\\temp\\worms.txt",header = T,row.names = 1)
```

Once the file has been imported to R we want to do two things:

- use attach to make the variables accessible by name within the R session, and
- use names to get a list of the variable names.

Typically, the two commands are issued in sequence, whenever a new dataframe is imported from file:

```
attach(worms)
names(worms)
```

```
[ 1]  Field.Name"  "Area"  "Slope"         "Vegetation"
[ 5]  "Soil.pH"    "Damp"  "Worm.density"
```

To see the contents of the dataframe, just type its name worms

```
                   Area  Slope  Vegetation  Soil.pH   Damp  Worm.density
Nash's.Field        3.6     11   Grassland      4.1  FALSE             4
Silwood.Bottom      5.1      2      Arable      5.2  FALSE             7
Nursery.Field       2.8      3   Grassland      4.3  FALSE             2
Rush.Meadow         2.4      5      Meadow      4.9   TRUE             5
Gunness'.Thicket    3.8      0       Scrub      4.2  FALSE             6
Oak.Mead            3.1      2   Grassland      3.9  FALSE             2
Church.Field        3.5      3   Grassland      4.2  FALSE             3
Ashurst             2.1      0      Arable      4.8  FALSE             4
The.Orchard         1.9      0     Orchard      5.7  FALSE             9
Rookery.Slope       1.5      4   Grassland      5.0   TRUE             7
Garden.Wood         2.9     10       Scrub      5.2  FALSE             8
North.Gravel        3.3      1   Grassland      4.1  FALSE             1
South.Gravel        3.7      2   Grassland      4.0  FALSE             2
Observatory.Ridge   1.8      6   Grassland      3.8  FALSE             0
Pond.Field          4.1      0      Meadow      5.0   TRUE             6
Water.Meadow        3.9      0      Meadow      4.9   TRUE             8
Cheapside           2.2      8       Scrub      4.7   TRUE             4
Pound.Hill          4.4      2      Arable      4.5  FALSE             5
Gravel.Pit          2.9      1   Grassland      3.5  FALSE             1
Farm.Wood           0.8     10       Scrub      5.1   TRUE             3
```

If, as here, the rows have unique names as part of the dataframe, we suppress R's natural inclination to produce its own row numbers, by telling R the number of the column containing the row names (1 in this case) as part of the read.table function (see above). Notice, also, that R has expanded our abbreviated T and F into `TRUE` and `FALSE`. The object called worms now has all the attributes of a dataframe. For example, you can summarize it, using summary:

summary(worms)

```
     Area            Slope        Vegetation        Soil.pH         Damp
Min.   :0.800   Min.   : 0.00   Arable    :3   Min.   :3.500   Mode :logical
1st Qu.:2.175   1st Qu.: 0.75   Grassland :9   1st Qu.:4.100   FALSE:14
Median :3.000   Median : 2.00   Meadow    :3   Median :4.600   TRUE :6
Mean   :2.990   Mean   : 3.50   Orchard   :1   Mean   :4.555
3rd Qu.:3.725   3rd Qu.: 5.25   Scrub     :4   3rd Qu.:5.000
Max.   :5.100   Max.   :11.00                  Max.   :5.700

 Worm.density
Min.   :0.00
1st Qu.:2.00
Median :4.00
Mean   :4.35
3rd Qu.:6.25
Max.   :9.00
```

Values of continuous variables are summarized under six headings: one parametric (the arithmetic mean) and five non-parametric (maximum, minimum, median, 25 percentile – the first quartile – and 75 percentile – the third quartile). Levels of categorical variables are counted. Note that the field names are not summarized, because they have been declared to be the row names, and hence all the names have to be unique.

Selecting Parts of a Dataframe: Subscripts

We often want to extract part of a dataframe. This is a very general procedure in R, accomplished using what are called subscripts. You can think of subscripts as addresses within a vector, a matrix or a dataframe. Subscripts in R appear within square brackets, thus y[7] is the seventh element of the vector called *y* and z[2,6] is the second row of the sixth column of a two-dimensional matrix called *z*. This is in contrast to arguments to functions in R, which appear in round brackets (4,7).

We might want to select all the rows of a dataframe for certain specified columns. Or we might want to select all the columns for certain specified rows of the dataframe. The convention in R is that when we do not specify any subscript, then all the rows, or all the columns is assumed. This syntax is difficult to understand on first acquaintance, but [,'blank then comma' means 'all the rows' and,] 'comma then blank' means all the columns. For instance, to select the first column of the dataframe, use subscript [,1]. Thus, to select all the rows of the first three columns of worms, we write:

```
worms[,1:3]
```

```
                       Area  Slope   Vegetation
Nash's.Field            3.6     11    Grassland
Silwood.Bottom          5.1      2       Arable
Nursery.Field           2.8      3    Grassland
Rush.Meadow             2.4      5       Meadow
Gunness'.Thicket        3.8      0        Scrub
Oak.Mead                3.1      2    Grassland
Church.Field            3.5      3    Grassland
Ashurst                 2.1      0       Arable
The.Orchard             1.9      0      Orchard
Rookery.Slope           1.5      4    Grassland
Garden.Wood             2.9     10        Scrub
North.Gravel            3.3      1    Grassland
South.Gravel            3.7      2    Grassland
Observatory.Ridge       1.8      6    Grassland
Pond.Field              4.1      0       Meadow
Water.Meadow            3.9      0       Meadow
Cheapside               2.2      8        Scrub
Pound.Hill              4.4      2       Arable
Gravel.Pit              2.9      1    Grassland
Farm.Wood               0.8     10        Scrub
```

To select just the middle 11 rows for all the columns of the dataframe, use subscript [5:15,] like this:

```
worms[5:15,]
```

```
                    Area  Slope  Vegetation  Soil.pH   Damp  Worm.density
Gunness'.Thicket     3.8      0       Scrub      4.2  FALSE             6
Oak.Mead             3.1      2   Grassland      3.9  FALSE             2
Church.Field         3.5      3   Grassland      4.2  FALSE             3
Ashurst              2.1      0     Arable       4.8  FALSE             4
The.Orchard          1.9      0    Orchard       5.7  FALSE             9
Rookery.Slope        1.5      4   Grassland      5.0   TRUE             7
Garden.Wood          2.9     10       Scrub      5.2  FALSE             8
North.Gravel         3.3      1   Grassland      4.1  FALSE             1
South.Gravel         3.7      2   Grassland      4.0  FALSE             2
Observatory.Ridge    1.8      6   Grassland      3.8  FALSE             0
Pond.Field           4.1      0      Meadow      5.0   TRUE             6
```

It is often useful to select certain rows, based on **logical tests** on the values of one or more variables. Here is the code to select only those rows which have Area $>$ 3 and Slope $<$ 3 using 'comma then blank' like this:

```
worms[Area > 3 & Slope < 3,]
```

```
                    Area  Slope  Vegetation  Soil.pH   Damp  Worm.density
Silwood.Bottom       5.1      2      Arable      5.2  FALSE             7
Gunness'.Thicket     3.8      0       Scrub      4.2  FALSE             6
Oak.Mead             3.1      2   Grassland      3.9  FALSE             2
North.Gravel         3.3      1   Grassland      4.1  FALSE             1
South.Gravel         3.7      2   Grassland      4.0  FALSE             2
Pond.Field           4.1      0      Meadow      5.0   TRUE             6
Water.Meadow         3.9      0      Meadow      4.9   TRUE             8
Pound.Hill           4.4      2      Arable      4.5  FALSE             5
```

Sorting

You can sort the rows or the columns of the dataframe in any way you choose, but you need to state which columns you want to be sorted (typically you will want all of them sorted, i.e. columns 1:6 in the case of worms). Suppose we want the rows of the whole dataframe sorted by Area (this is the variable in column number one [,1]):

```
worms[order(worms[,1]),1:6]
```

```
                    Area  Slope  Vegetation  Soil.pH   Damp  Worm.density
Farm.Wood            0.8     10       Scrub      5.1   TRUE             3
Rookery.Slope        1.5      4   Grassland      5.0   TRUE             7
Observatory.Ridge    1.8      6   Grassland      3.8  FALSE             0
```

```
The.Orchard         1.9     0     Orchard     5.7   FALSE          9
Ashurst             2.1     0      Arable     4.8   FALSE          4
Cheapside           2.2     8       Scrub     4.7    TRUE          4
Rush.Meadow         2.4     5      Meadow     4.9    TRUE          5
Nursery.Field       2.8     3   Grassland     4.3   FALSE          2
Garden.Wood         2.9    10       Scrub     5.2   FALSE          8
Gravel.Pit          2.9     1   Grassland     3.5   FALSE          1
Oak.Mead            3.1     2   Grassland     3.9   FALSE          2
North.Gravel        3.3     1   Grassland     4.1   FALSE          1
Church.Field        3.5     3   Grassland     4.2   FALSE          3
Nash's.Field        3.6    11   Grassland     4.1   FALSE          4
South.Gravel        3.7     2   Grassland     4.0   FALSE          2
Gunness'.Thicket    3.8     0       Scrub     4.2   FALSE          6
Water.Meadow        3.9     0      Meadow     4.9    TRUE          8
Pond.Field          4.1     0      Meadow     5.0    TRUE          6
Pound.Hill          4.4     2      Arable     4.5   FALSE          5
Silwood.Bottom      5.1     2      Arable     5.2   FALSE          7
```

Alternatively, the dataframe can be sorted in descending order by Soil pH, with only Soil pH and Worm density as output:

```
worms[rev(order(worms[,4])),c(4,6)]
```

```
                   Soil.pH   Worm.density
The.Orchard            5.7              9
Garden.Wood            5.2              8
Silwood.Bottom         5.2              7
Farm.Wood              5.1              3
Pond.Field             5.0              6
Rookery.Slope          5.0              7
Water.Meadow           4.9              8
Rush.Meadow            4.9              5
Ashurst                4.8              4
Cheapside              4.7              4
Pound.Hill             4.5              5
Nursery.Field          4.3              2
Church.Field           4.2              3
Gunness'.Thicket       4.2              6
North.Gravel           4.1              1
Nash's.Field           4.1              4
South.Gravel           4.0              2
Oak.Mead               3.9              2
Observatory.Ridge      3.8              0
Gravel.Pit             3.5              1
```

Saving Your Work

At any stage, you can highlight material on the screen, then use copy (Ctrl C) and paste (Ctrl V) to save it to a Word document. Note that to keep tabular material properly aligned in the Word document you will need to use a font like `Courier New` that has absolute (rather than proportional) spacing. Graphs that you want to keep should be saved as you go along (using File: Save As when the graphics window is highlighted, to chose an appropriate format), or copied and pasted into a Word document.

You can review the command lines entered during a session with

```
history(Inf)
```

and you can copy from this and paste into the command line to save re-typing. You can save the history of command lines to a text file like this

```
savehistory("c:\\temp\\today.txt")
```

and read it back into R with `loadhistory("c:\\temp\\today.txt")`. The session as a whole can be saved as a binary file with

```
save(list=ls(), file="c:\\temp\\all.Rdata")
```

and retrieved using `load("c:\\temp\\all.Rdata")`

Tidying Up

At the end of a session in R, it is good practice to remove (`rm`) any variables names you have created (using, say, `x <- 5.6`) and to `detach` any dataframes you have attached earlier in the session. That way, variables with the same names but different properties will not get in each other's way in subsequent work:

```
rm(x,y,z)
detach(worms)
```

This command does not make the dataframe called worms disappear; it just means that the variables within worms, like Slope and Area, are no longer accessible directly by name. To get rid of everything, including all the dataframes, type

```
rm(list=ls())
```

but be absolutely sure that you want to be as Draconian as this before you execute the command.