UNIT-II

WORKING WITH OBJECTS

Data objects are the fundamental items that you work with in R. Carrying out analyses on your data and making sense of the results are the primary reasons

Manipulating Objects

There are many ways to manipulate your data in R.

Manipulating Vectors

Vectors are essentially the building blocks of more complicated items and these are fundamental objects. You can manipulate these in many ways. The main ways of manipulation of vectors are

1. Selecting and displaying certain parts
2. Sorting and rearranging
3. Returning logical values

Selecting and Displaying Parts of a Vector

Being able to select and display parts of a vector can be important for many reasons.
Ex: if you have a large sample of data you may want to see which items are larger than a certain value.
if you may want to extract a series of values as a subsample in an analysis.
Here is a simple vector of numbers that form a sample:

```
data1=c(3,5,7,5,3,2,6,8,5,6,9)
```

```
Ex:> data1
[1] 3 5 7 5 3 2 6 8 5 6 9
```

Command:
length(data1)   This tells you how many elements there are in the vector, including NA items
Ex: data1[(length(data1)-5):length(data1)]   This shows the last five elements of the vector.

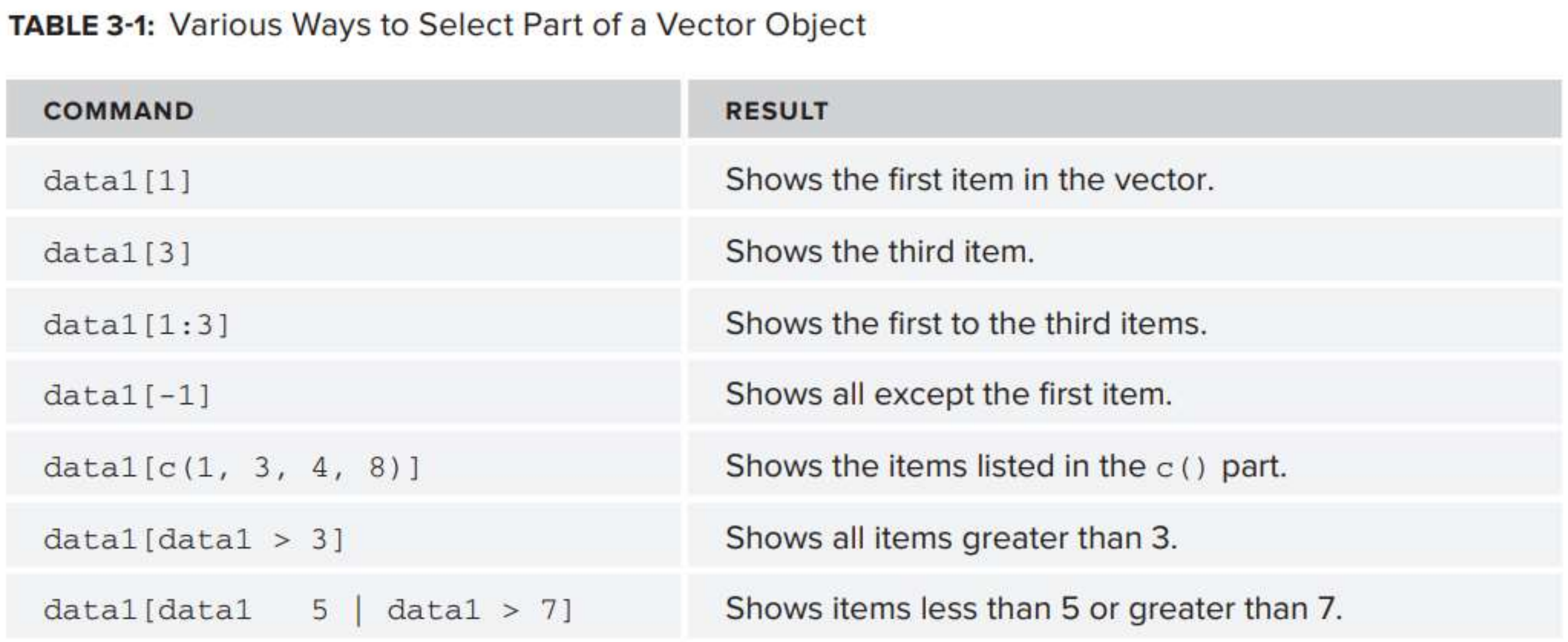


TABLE 3-1: Various Ways to Select Part of a Vector Object

COMMAND	RESULT
`data1[1]`	Shows the first item in the vector.
`data1[3]`	Shows the third item.
`data1[1:3]`	Shows the first to the third items.
`data1[-1]`	Shows all except the first item.
`data1[c(1, 3, 4, 8)]`	Shows the items listed in the `c()` part.
`data1[data1 > 3]`	Shows all items greater than 3.
`data1[data1   5 | data1 > 7]`	Shows items less than 5 or greater than 7.

You can use other operations.   The max() is used to get the largest value in the vector.

```
> data1
 [1] 3 5 7 5 3 2 6 8 5 6 9
> max(data1)
[1] 9
> which(data1 == max(data1))
[1] 11
```

The first command, max(), gives the actual value that is the largest numerical value in the vector (in this case 9). The second command asks which of the elements is the largest. The value obtained is 11, meaning that the eleventh item in the vector is the largest. Notice that double equal signs are used here.

Another useful command is one that generates sequences, seq(). You can use this to extract values from a vector at regular intervals.

The general form of the seq() command is   seq(start, end, interval)

```
> data1[seq(1, length(data1), 2)]
[1] 3 7 3 6 5 9
```

This would pick out a sequence beginning with the first and ending with the last and with an interval of two. In other words, you select the first, third, fifth, and so on. You use the length() part to ensure that you stop once you get to the end

These commands will work on character vectors as well as numeric.

```
> data5
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
> data5[-1:-6]
[1] "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
> which(data5 == max(data5))
[1] 9
```

In this example R has selected the ninth item, which is “Sep”. The items are sorted alphabetically, so the biggest is determined by that order as opposed to a numerical value.

Sorting and Rearranging a Vector

You can rearrange the items in a vector to be in one order or another using the sort() command.The default is to use ascending order and to leave out any NA items,

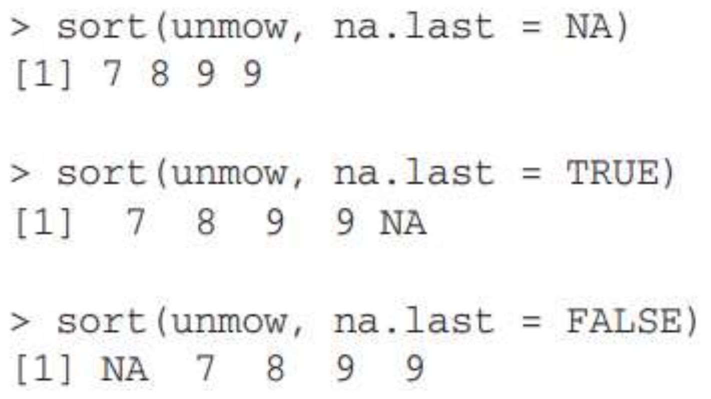
```
> unmow
[1] 8 9 7 9 NA
> sort(unmow)
[1] 7 8 9 9
```

You can rearrange the items in a vector to be in one order or another using sort() command
The default is to use ascending order and to leave out any NA items

```
> unmow
[1] 8 9 7 9 NA
> sort(unmow)
[1] 7 8 9 9
> sort(unmow, decreasing = TRUE)  [You can alter the order using decreasing=TRUE]
[1] 9 9 8 7
```

You can change the way NA items are dealt with using the na.last = instruction.
 3 options:

1. NA ----- meaning is dropped
2. TRUE------ means   NA items are placed at last.
3. FALSE------ means NA items are placed at first.



```
> sort(unmow, na.last = NA)
[1] 7 8 9 9

> sort(unmow, na.last = TRUE)
[1]  7  8  9  9 NA

> sort(unmow, na.last = FALSE)
[1] NA  7  8  9  9
```

You can get an index using the order() command. This uses the same instructions as the sort() command, but tells you the position of each item along the vector:

```
> unmow
[1] 8 9 7 9 NA
> order(unmow)
[1] 3 1 2 4 5                    [Sorting order=7 8 9 9 NA: 3 1 2 4 5]
> order(unmow, na.last = NA)
[1] 3 1 2 4                      [Sorting order=7 8 9 9 NA: 3 1 2 4 ]
> order(unmow, na.last = FALSE)
[1] 5 3 1 2 4                    [Sorting order=7 8 9 9 NA: 5 3 1 2 4 ]
```

The rank() command sorts your data in a slightly different way than the order() command: it handles tied values.

```
> unmow
[1] 8 9 7 9 NA
> order(unmow)
[1] 3 1 2 4 5
> rank(unmow)
[1] 2.0 3.5 1.0 3.5 5.0
```

The default instructions in the preceding example set na.last = TRUE, which is slightly different than the sort() command. You can also see here that the order() command reported that the third item in the vector is the first value when ordered numerically.

The rank() command sorts your data in a slightly different way than the order() command: it handles tied values. Compare the two commands in the following example

```
> unmow
[1] 8 9 7 9 NA
> order(unmow)
[1] 3 1 2 4 5
> rank(unmow)
[1] 2.0 3.5 1.0 3.5 5.0
```

When using the order() command you see that the two values of 9 in the original vector are given a different value (2 and 4). There is no reason why they could not have been ordered the other way around (that is, 4 and 2); the default is to take them as they come. When you use the rank() command you get a different result; the two 9 values are the third and fourth largest and by default they get a shared rank of 3.5. The NA item is placed at the end because by default the na.last = TRUE is the default setting.

```
> unmow
[1] 8 9 7 9 NA
> rank(unmow, ties.method = 'first')
[1] 2 3 1 4 5
> rank(unmow, ties.method = 'average')
[1] 2.0 3.5 1.0 3.5 5.0
> rank(unmow, ties.method = 'max')
[1] 2 4 1 4 5
> rank(unmow, ties.method = 'random', na.last = 'keep')
[1] 2 3 1 4 NA
```

You can alter the way tied values are handled using the ties. Method = “method” instruction; by default this is set to “average” and this is the method used in all non-parametric statistical routines. You have other options and can set “first”, “random”, “max”, or “min” as alternatives to the default “average”:

```
> dat.na
[1] 2 5 4 NA 7 3 9 NA 12
> rank(dat.na, na.last = 'keep')
[1] 1 4 3 NA 5 2 6 NA 7
```

In the last example you can see a different option for the na.last = instruction; you can choose to keep any NA items intact

The rank() command is especially useful for non-parametric statistical testing, which relies heavily on the original data being converted to ranks.

Returning Logical Values from a Vector which() command was used to tell which items in a vector met some criterion.

```
> data1
 [1] 3 5 7 5 3 2 6 8 5 6 9
> which(data1 == 6)
[1] 7 10
```

If you omit the which() command and use the == directly you get a different sort of answer

```
> data1 == 6
 [1] FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE TRUE FALSE
```

You can use other mathematical operators, especially the greater than or less than symbols

```
> data2 >5
 [1] FALSE FALSE TRUE FALSE FALSE FALSE TRUE TRUE FALSE TRUE TRUE FALSE FALSE
[14] TRUE FALSE FALSE
> data2 <5
 [1] TRUE FALSE FALSE FALSE TRUE TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE
[14] FALSE TRUE TRUE
```

You can also combine items using various logical operators

```
> data2 >5 & data2 <8
 [1] FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE FALSE
[14] TRUE FALSE FALSE
> data8
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
> data8 == 'Feb'| data8 == 'Apr'
 [1] FALSE TRUE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

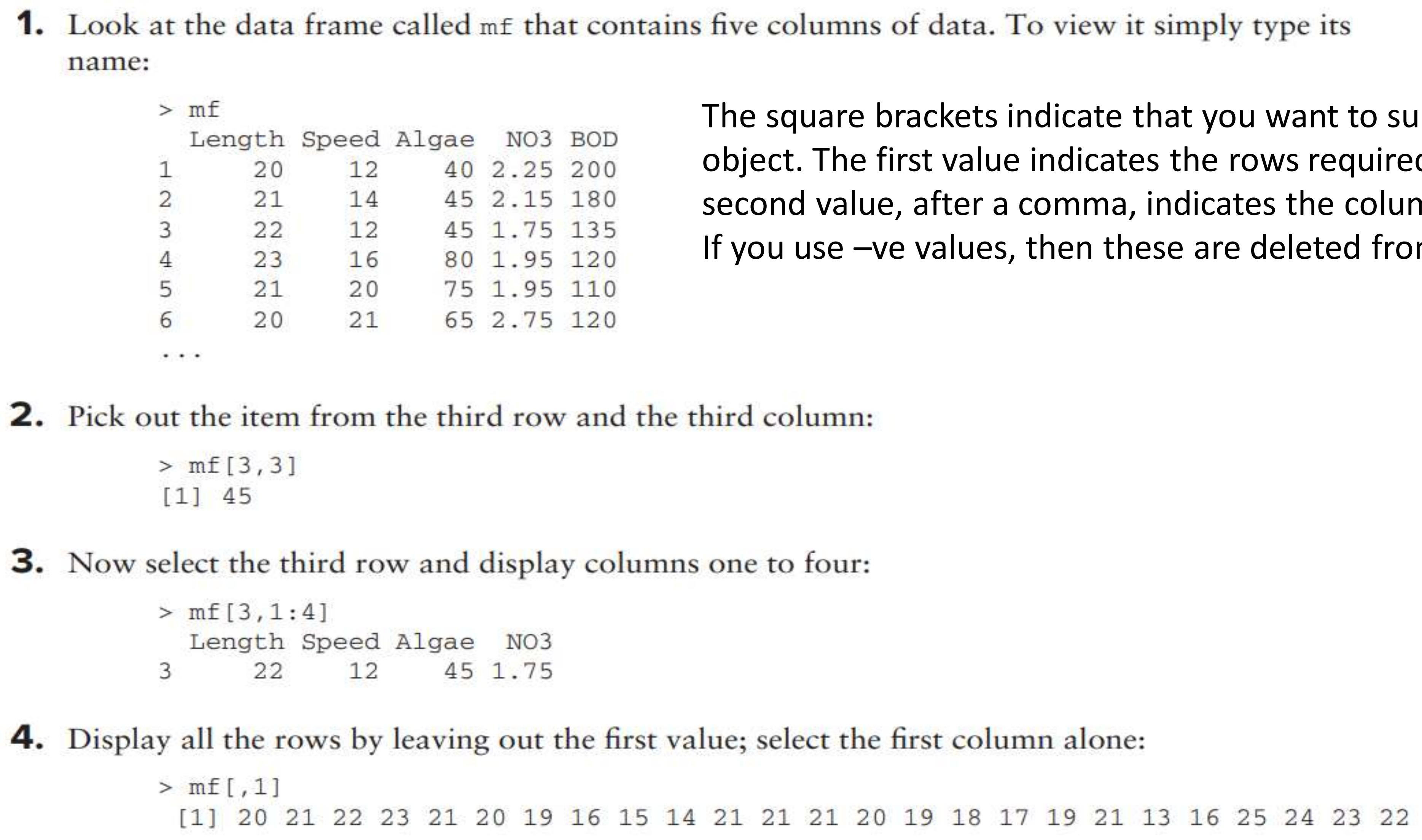
Manipulating Matrix and Data Frames

a matrix or a data frame you have a two-dimensional object. Complex objects tend to be used for many statistical operations simply because that is the nature of statistics. Most of the data will be in complex forms so it is essential to deal the complex data and manipulate the data.

Selecting and Displaying Parts of a Matrix or Data Frame

Use the mf data from the Beginning.RData file for this , selecting out various parts using the square bracket syntax

Select Parts of a Data Frame Object **load("C:\\Users\\ASUS\\Downloads\\Beginning.RData")**



1. Look at the data frame called `mf` that contains five columns of data. To view it simply type its name:

```
> mf
  Length Speed Algae  NO3 BOD
1     20    12    40 2.25 200
2     21    14    45 2.15 180
3     22    12    45 1.75 135
4     23    16    80 1.95 120
5     21    20    75 1.95 110
6     20    21    65 2.75 120
...
```

2. Pick out the item from the third row and the third column:

```
> mf[3,3]
[1] 45
```

3. Now select the third row and display columns one to four:

```
> mf[3,1:4]
  Length Speed Algae  NO3
3     22    12    45 1.75
```

4. Display all the rows by leaving out the first value; select the first column alone:

```
> mf[,1]
 [1] 20 21 22 23 21 20 19 16 15 14 21 21 21 20 19 18 17 19 21 13 16 25 24 23 22
```

The square brackets indicate that you want to subset the data object. The first value indicates the rows required and the second value, after a comma, indicates the columns required. If you use –ve values, then these are deleted from the display

5. Specify several rows but leave out a value at the end to display all columns:

```
> mf[c(1,3,5,7),]
  Length Speed Algae  NO3 BOD
1     20    12    40 2.25 200
3     22    12    45 1.75 135
5     21    20    75 1.95 110
7     19    17    65 1.85  95
```

6. Now specify several rows but use a `-4` to indicate that you want to display all columns except the fourth:

```
> mf[c(1,3,5,7),-4]
```

```
  Length Speed Algae BOD
1     20    12    40 200
3     22    12    45 135
5     21    20    75 110
7     19    17    65  95
```

you can use a variety of methods to select the items you require, seq() or c() commands.

Named columns (or rows) can be displayed by giving the names (in quotes) instead of a plain number

7. Because the columns are named you can select one by using its name rather than a simple value:

```
> mf[c(1,3,5,7), 'Algae']
[1] 40 45 75 65
```

8. Try giving a single value in the square brackets:

```
> mf[3]
  Algae
1    40
2    45
3    45
4    80
5    75
6    65
7    65
...
```

Select Parts of a Matrix Data Object

Use the bird data from the Beginning.RData file for this,selecting out various parts using the square bracket syntax.

1. Look at the matrix object called bird; simply type its name:

```
> bird
              Garden Hedgerow Parkland Pasture Woodland
Blackbird         47       10       40       2        2
Chaffinch         19        3        5       0        2
Great Tit         50        0       10       7        0
House Sparrow     46       16        8       4        0
Robin              9        3        0       0        2
Song Thrush        4        0        6       0        0
```

2. You can see that there are five columns and six rows; the rows are labeled rather than having a numeric index. At first glance this appears like a data frame, so use the `str()` command to look more closely:

```
> str(bird)
 int [1:6, 1:5] 47 19 50 46 9 4 10 3 0 16 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:6] "Blackbird" "Chaffinch " "Great Tit" "House Sparrow " ...
  ..$ : chr [1:5] "Garden" "Hedgerow" "Parkland" "Pasture" ...
```

3. Use the `class()` command to see that this is a matrix:

```
> class(bird)
[1] "matrix"
```

4. Select the second row and all the columns:

```
> bird[2,]
 Garden Hedgerow Parkland  Pasture Woodland
     19        3        5        0        2
```

5. Now select all rows but only the fourth column:

```
> bird[,4]
     Blackbird     Chaffinch     Great Tit     House Sparrow     Robin     Song Thrush
             2             0             7                 4         0               0
```

6. Use named rows rather than a simple number and choose all columns:

```
> bird[c('Robin', 'Blackbird'),]
          Garden Hedgerow Parkland Pasture Woodland
Robin          9        3        0       0        2
Blackbird     47       10       40       2        2
```

7. Now select a single row and column:

```
> bird[3,1]
[1] 50
```

The str() command shows you details of the structure of an object.
The class() command   shows you what kind of object you are dealing

8. Now specify a single value:

```
> bird[4]
[1] 46
```

The [row, column] syntax works. You can type in a named row or column as long as you make sure the name is in quotes.

You can also use –ve values to indicate rows or columns that you do not want, only works if you use numbers

Sorting and Rearranging a Matrix or Data Frame You can use the sort(), order(), and rank() commands on your matrix. If you specify simply the matrix object you get results along the following lines:

```
> sort(bird)
 [1]  0  0  0  0  0  0  0  0  0  2  2  2  2  3  3  4  4  5  6  7  8  9 10 10 16 19
[27] 40 46 47 50

> order(bird)
 [1]  9 12 17 20 23 24 27 28 30 19 25 26 29  8 11  6 22 14 18 21 16  5  7 15 10  2
[27] 13  4  1  3

> rank(bird)
 [1] 29.0 26.0 30.0 28.0 22.0 16.5 23.5 14.5  5.0 25.0 14.5  5.0 27.0 18.0 23.5
[16] 21.0  5.0 19.0 11.5  5.0 20.0 16.5  5.0  5.0 11.5 11.5  5.0  5.0 11.5  5.0
```

If you want to sort, order, or rank rows or columns, you must specify them explicitly:

```
> sort(bird[,1])
  Song Thrush       Robin     Chaffinch    House Sparrow     Blackbird     Great Tit
            4           9            19               46            47            50
```

the first column is used as the target to sort or order

```
> order(bird[,1])
[1] 6 5 2 4 1 3
```

The order() command allows you to give additional vectors; these are used as tie-breakers to help resolve the order of items in the
first vector

```
> order(bird[,5])
[1] 3 4 6 1 2 5

> order(bird[,5], bird[,1])
[1] 6 4 3 5 2 1
```

In the first case the result obtained is the order of the fifth column. There are a number of tied values and the result takes these in the order it finds them.
In the second case the first column is used to influence the order and the final result changes. Here a second column was added to the command

```
> grass2
  mow unmow
1  12     8
2  15     9
3  17     7
4  11     9
5  15    NA
```

If you have a data frame, You cannot perform a sort() command on an entire data frame, even if it is composed entirely of the same kind of data (that is, all numeric or all character)

You simply get an error; the command needs to operate on a single vector.
You can pick out a part of your data frame to run the sort() command

```
> sort(grass2)
Error in  [.data.frame (x, order(x, na.last = na.last, decreasing = decreasing)) :
  undefined columns selected
```

In the first example the first row of the data frame is sorted; the subsequent examples all sort the first column but use differing syntax.

```
> sort(grass2[,1])
[1] 11 12 15 15 17
```

The $, for example, is used to obtain a single vector from a list or a data frame.

```
> sort(grass2[,'mow'])
[1] 11 12 15 15 17
```

```
> sort(grass2$mow)
[1] 11 12 15 15 17
```

This syntax can also be used in the order() command, which works pretty much the same way as it did for the matrix object:

```
> order(grass2)
 [1]  8  6  7  9  4  1  2  5  3 10
```

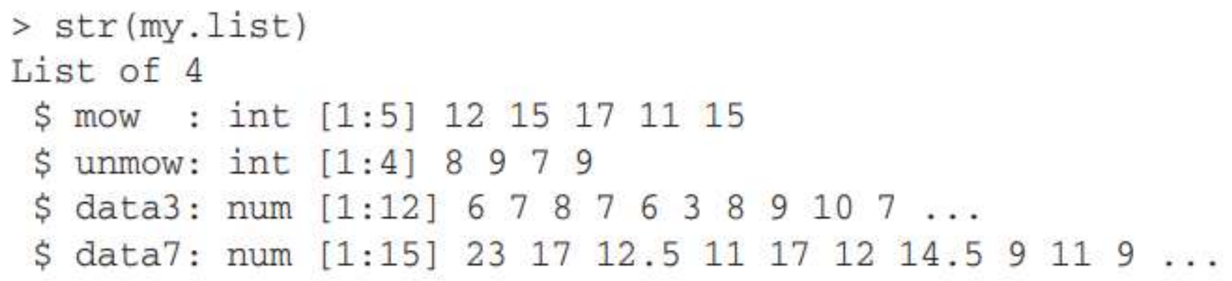
The two cases illustrate two ways to use the syntax to order the first column, using the second column as a tie-breaker
with() This temporarily “opens up” the data frame and allows you to utilize the contents of the specified object.

```
> order(grass2$mow, grass2[,2])
[1] 4 1 2 5 3
```

```
> with(grass2, order(mow,unmow))
[1] 4 1 2 5 3
```

Manipulating Lists

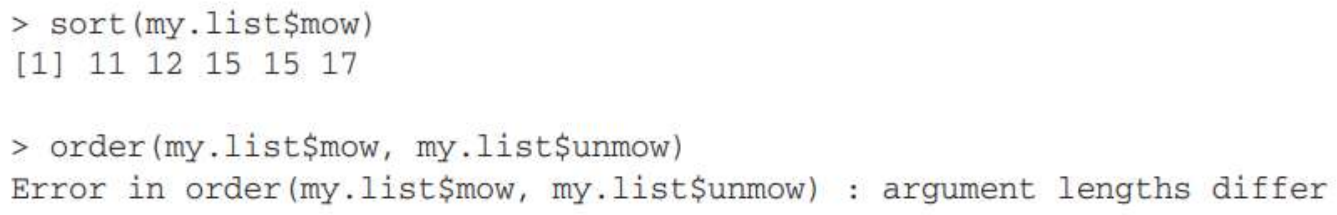
When you have a list the square brackets give a different result compared to other data objects you have met.
check the structure using the str() command:



```
> str(my.list)
List of 4
 $ mow  : int [1:5] 12 15 17 11 15
 $ unmow: int [1:4] 8 9 7 9
 $ data3: num [1:12] 6 7 8 7 6 3 8 9 10 7 ...
 $ data7: num [1:15] 23 17 12.5 11 17 12 14.5 9 11 9 ...
```

Here there are four elements in the list; each has a name preceded by a dollar sign. The list is a one-dimensional object, so you can use only a single value in the square brackets
> my.list[1]
$mow
[1] 12 15 17 11 15
You will need to use a slightly different convention to extract the elements; you must use the $ in the name.

This is the only way you can utilize the sort(), order(), or rank() commands on list items. In the first case the mow vector is extracted from the list and a sort() command is performed.
In the next case the order() command is used.
This fails because the two items are of different lengths;
you therefore cannot use the second item as a tie-breaker.

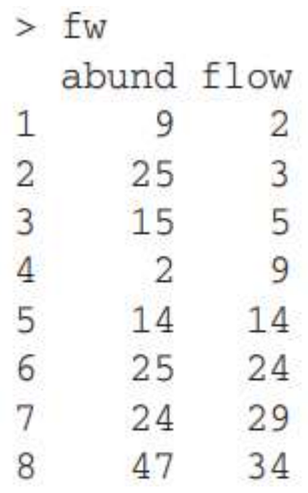


```
> sort(my.list$mow)
[1] 11 12 15 15 17

> order(my.list$mow, my.list$unmow)
Error in order(my.list$mow, my.list$unmow) : argument lengths differ
```

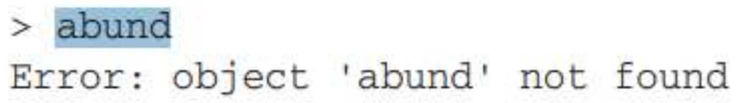
use the order() command on any of the separate items in the list:
> order(my.list$unmow)
[1] 3 1 2 4

Viewing Objects within Objects
When you create a list, matrix, or data frame you are bundling together several items. these do not appear when you use ls() command.

Looking Inside Complicated Data Objects
looked at some data called fw. The data were presented as a data frame with two columns



```
> fw
  abund flow
1     9    2
2    25    3
3    15    5
4     2    9
5    14   14
6    25   24
7    24   29
8    47   34
```



```
> abund
Error: object 'abund' not found
```

The problem is that the abund and flow data are contained within the fw object
use the $ to help penetrate the data and extract parts you want to view
> fw$abund
[1] 9 25 15 2 14 25 24 47
use the square brackets as before to select out parts of the item:
> fw$abund[1:4]
[1] 9 25 15 2
Even in a list also ,use the $ to extract the parts of them.
> my.list$mow
[1] 12 15 17 11 15
You can now subset this using the square brackets:
> my.list$mow[1:4]
[1] 12 15 17 11
For a matrix ,you can get a error like this
> bird$Garden
Error in bird$Garden : $ operator is invalid for atomic vectors

This is because a matrix is essentially a single data item that has been displayed in rows and columns.

Matrix objects: use the column name in the square brackets

```
> bird[,'Garden']
 Blackbird Chaffinch Great Tit House Sparrow Robin Song Thrush
 47             19          50                      46            9              4
```

Because there are also row names in your matrix you can use these names:

```
> bird['Robin',]
 Garden Hedgerow Parkland Pasture Woodland
    9           3                 0            0            2
```

use the names of data frames

```
> mf[,'NO3']
 [1] 2.25 2.15 1.75 1.95 1.95 2.75 1.85 1.75 1.95 2.35 2.35 2.35 2.05 1.85 1.75
[16] 1.45 1.35 2.05 1.25 1.05 2.55 2.85 2.95 2.85 1.75
```

Opening Complicated Data Objects

attach() command ----This allows the columns of a data frame and the elements of a list to be viewed without the need to use the $ sign

/////e attach() command is useful because it can help reduce typing

attach(my.list)

attach(mf)

If you try to do this for a matrix you get an error like so:

```
> attach(bird)
Error in attach(bird) :
  'attach' only works for lists, data frames and environments
```

data object with the same name as one that you retrieve using attach()

The mf data contains a column headed BOD.
 It so happens that there is a data item called BOD already in the datasets package

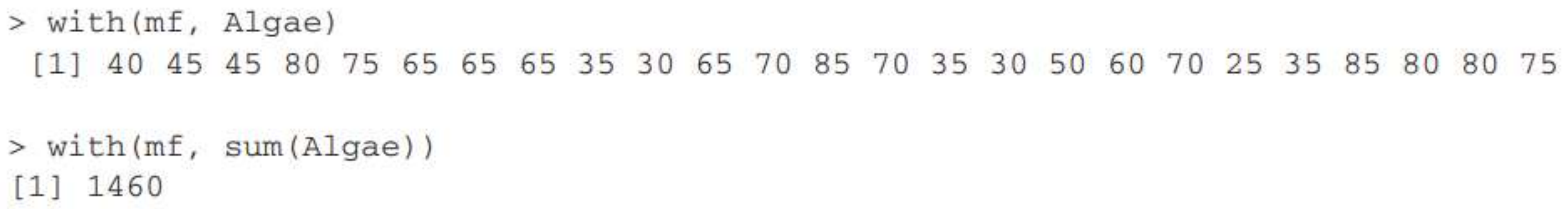
```
> attach(mf)
The following object(s) are masked from 'package:datasets':

    BOD
```

use a with() command that attaches the data only transiently. The basic form of the command is:

with(object, ...)

> Algae

Error: object 'Algae’ not found



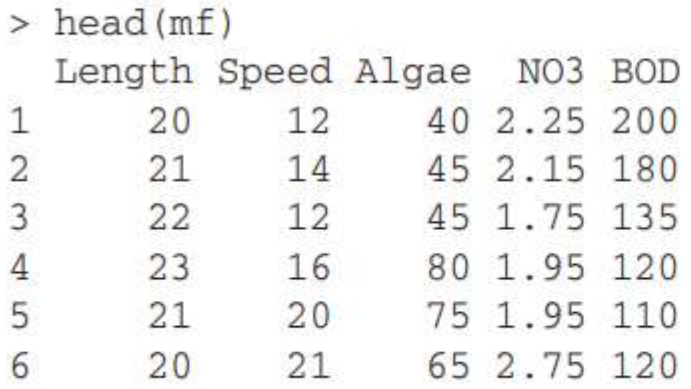
```
> with(mf, Algae)
 [1] 40 45 45 80 75 65 65 65 35 30 65 70 85 70 35 30 50 60 70 25 35 85 80 80 75

> with(mf, sum(Algae))
[1] 1460
```

Second example -- to view the vector of data. Third example ---which simply adds up the values and gives a final total.

Quick Looks at Complicated Data Objects

You might elect to show just the first few lines of a data object; which you can do using the **head()** command. This shows the top of your data object and by default shows the first six rows

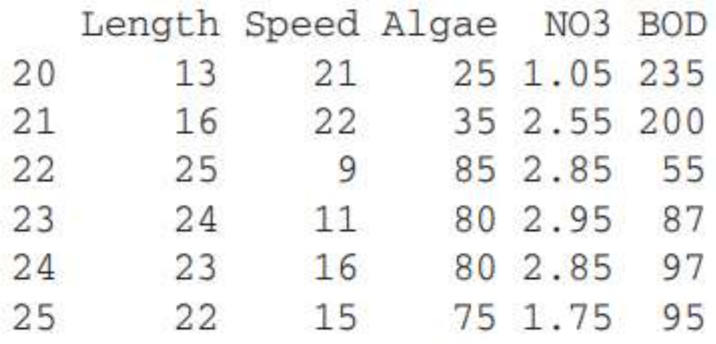
> head(mf)



```
> head(mf)
  Length Speed Algae  NO3 BOD
1     20    12    40 2.25 200
2     21    14    45 2.15 180
3     22    12    45 1.75 135
4     23    16    80 1.95 120
5     21    20    75 1.95 110
6     20    21    65 2.75 120
```

You can also display the bottom of the data using the tail() command

> tail(mf)

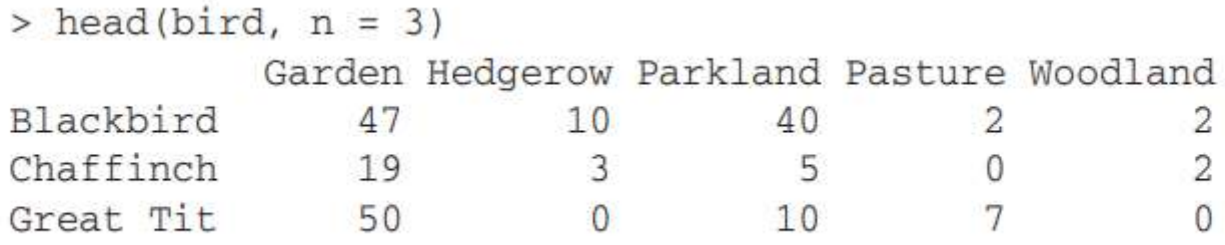


```
   Length Speed Algae  NO3 BOD
20     13    21    25 1.05 235
21     16    22    35 2.55 200
22     25     9    85 2.85  55
23     24    11    80 2.95  87
24     23    16    80 2.85  97
25     22    15    75 1.75  95
```

This is potentially more useful because you can also see how many rows there are in total

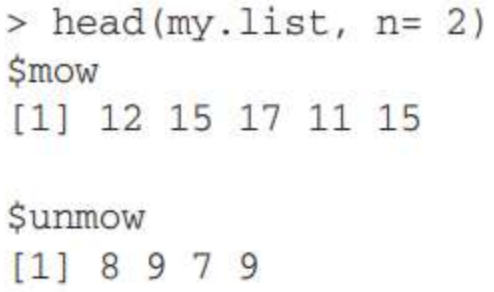
You can elect to show a different number of rows using the n = 3 instruction like

> head(bird, n = 3)



```
> head(bird, n = 3)
           Garden Hedgerow Parkland Pasture Woodland
Blackbird      47       10       40       2        2
Chaffinch      19        3        5       0        2
Great Tit      50        0       10       7        0
```

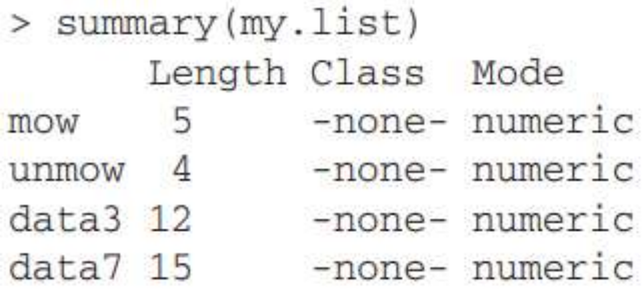
In this case the data are in a matrix.



```
> head(my.list, n= 2)
$mow
[1] 12 15 17 11 15

$unmow
[1] 8 9 7 9
```

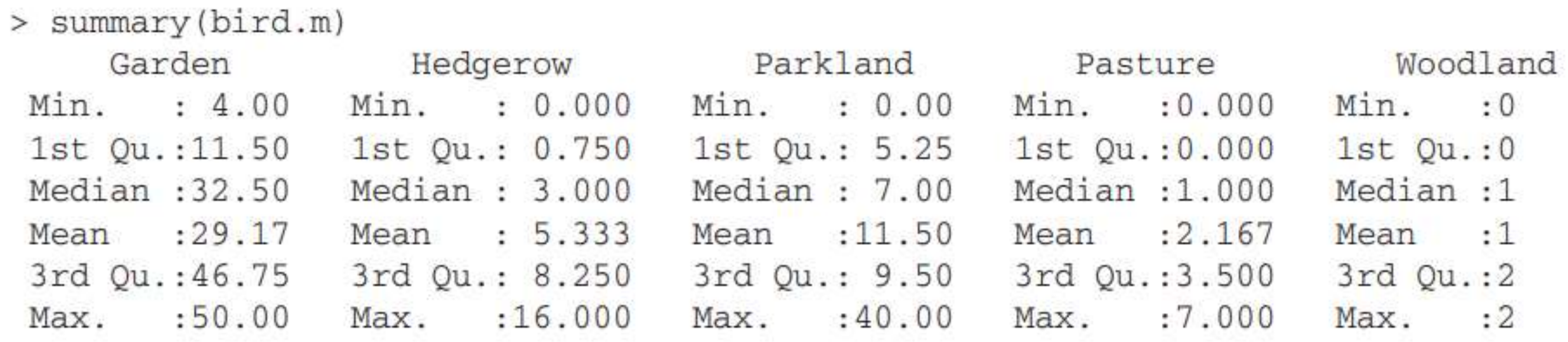
> head(my.list, n= 2)
$mow
[1] 12 15 17 11 15
$unmow
[1] 8 9 7 9

another way to get information about an object. use the summary() command, gives some simple statistics about the numeric data called species. The cut variable is a factor and the command shows the different levels as well as the number of observations in each.
If the object you are examining is a, **list** you are presented with a slightly simpler summary output:



```
> summary(my.list)
      Length Class  Mode
mow    5     -none- numeric
unmow  4     -none- numeric
data3 12     -none- numeric
data7 15     -none- numeric
```

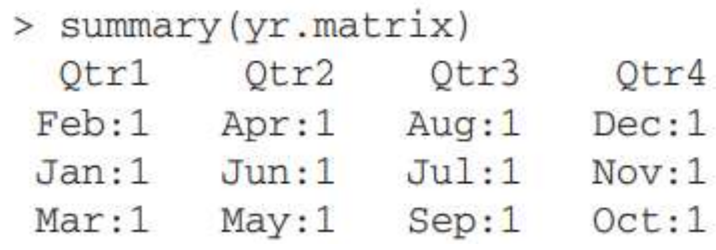
The summary command also works on a matrix. Each of the named columns is summarized using basic statistics



```
> summary(bird.m)
     Garden          Hedgerow         Parkland        Pasture          Woodland
 Min.   : 4.00   Min.   : 0.000   Min.   : 0.00   Min.   :0.000   Min.   :0
 1st Qu.:11.50   1st Qu.: 0.750   1st Qu.: 5.25   1st Qu.:0.000   1st Qu.:0
 Median :32.50   Median : 3.000   Median : 7.00   Median :1.000   Median :1
 Mean   :29.17   Mean   : 5.333   Mean   :11.50   Mean   :2.167   Mean   :1
 3rd Qu.:46.75   3rd Qu.: 8.250   3rd Qu.: 9.50   3rd Qu.:3.500   3rd Qu.:2
 Max.   :50.00   Max.   :16.000   Max.   :40.00   Max.   :7.000   Max.   :2
```

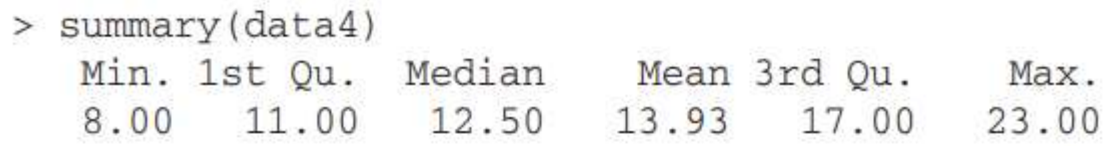
the matrix contains months of the year (that is, it contains character data):
When you use the summary() command you see each element listed along with how many there were in each “category”; in this case the values are all 1 because there is only one of each month

```
> yr.matrix
 Qtr1 Qtr2 Qtr3 Qtr4
row1 "Jan" "Apr" "Jul" "Oct"
row2 "Feb" "May" "Aug" "Nov"
row3 "Mar" "Jun" "Sep" "Dec“
```

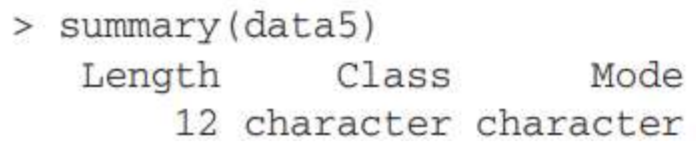


```
> summary(yr.matrix)
   Qtr1     Qtr2     Qtr3     Qtr4
 Feb:1    Apr:1    Aug:1    Dec:1
 Jan:1    Jun:1    Jul:1    Nov:1
 Mar:1    May:1    Sep:1    Oct:1
```

If the data are a simple vector, the output you get depends on the type of item. The following examples show the results for numeric data followed by character data



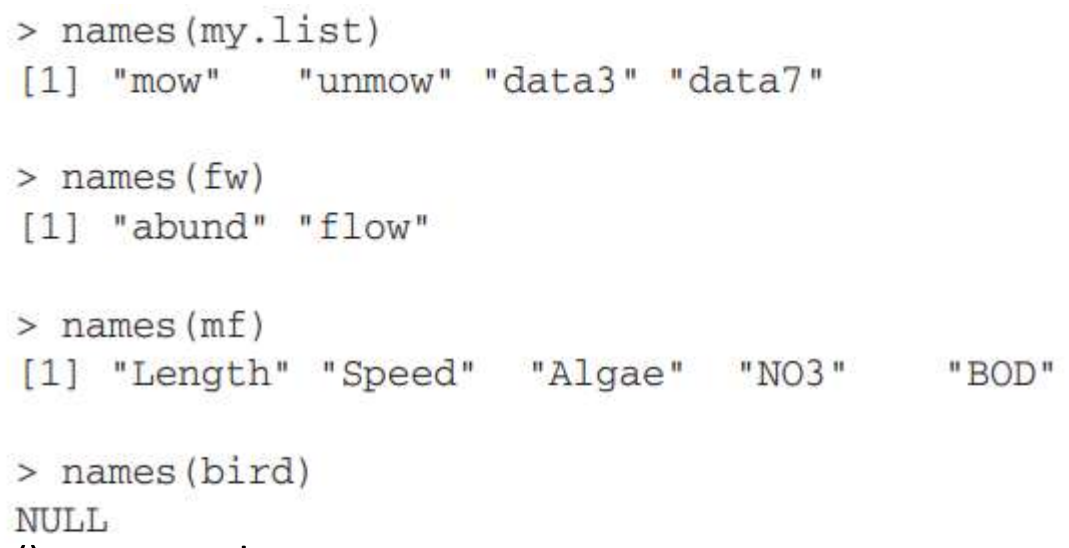
```
> summary(data4)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   8.00   11.00   12.50   13.93   17.00   23.00
```



```
> summary(data5)
   Length     Class      Mode
       12 character character
```

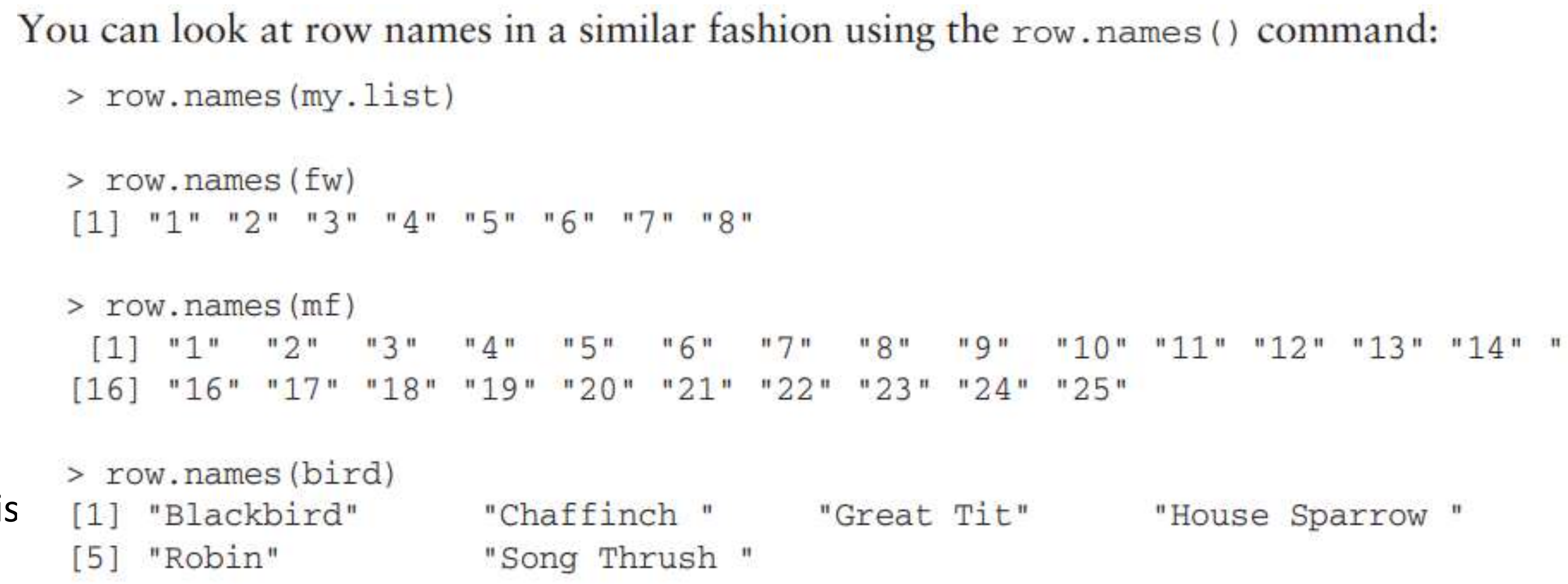
Viewing and Setting Names

You may want to see just the column (or row) names contained within an object



```
> names(my.list)
[1] "mow"    "unmow" "data3" "data7"

> names(fw)
[1] "abund" "flow"

> names(mf)
[1] "Length" "Speed"  "Algae"  "NO3"    "BOD"

> names(bird)
NULL
```

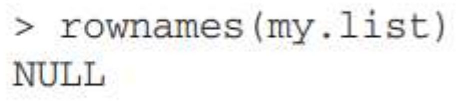
You can look at row names in a similar fashion using the row.names() command



You can look at row names in a similar fashion using the `row.names()` command:

```
> row.names(my.list)

> row.names(fw)
[1] "1" "2" "3" "4" "5" "6" "7" "8"

> row.names(mf)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "
[16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25"

> row.names(bird)
[1] "Blackbird"       "Chaffinch "      "Great Tit"       "House Sparrow "
[5] "Robin"           "Song Thrush "
```

Modify the command and use like this



```
> rownames(my.list)
NULL
```

You can also use colnames() in a similar way:

```
> colnames(mf)
[1] "Length" "Speed"  "Algae"  "NO3"    "BOD"

> colnames(my.list)
NULL

> colnames(bird)
[1] "Garden"   "Hedgerow" "Parkland" "Pasture"  "Woodland"
```

dimnames(); this looks at both the row and column names at the same time:

```
> dimnames(my.list)
NULL

> dimnames(mf)
[[1]]
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
"15"
[16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25"

[[2]]
[1] "Length" "Speed"  "Algae"  "NO3"    "BOD"
```

The command shows the row names first and then the column names,

```
> dimnames(bird)
[[1]]
[1] "Blackbird"     "Chaffinch"     "Great Tit"     "House Sparrow"
[5] "Robin"         "Song Thrush"

[[2]]
[1] "Garden"   "Hedgerow" "Parkland" "Pasture"  "Woodland"
```

Note: The command does not work on a list

Use these commands to see current names
 if you can get the names using the names() command, you can also set them

```
> names(mf)
[1] "Length" "Speed"  "Algae"  "NO3"    "BOD"
```

```
names(mf) = c('len','sp', 'alg', 'no3', 'bod')
```

```
> names(mf)
[1] "len" "sp"  "alg" "no3" "bod"
```

do the same thing using the colnames() or rownames() commands. colnames() and rownames() will work on a matrix

```
> sites = c('Taw', 'Torridge', 'Ouse', 'Exe', 'Lyn', 'Brook', 'Ditch', 'Fal')
> rownames(fw) = sites
> fw
         abund flow
Taw          9    2
Torridge    25    3
Ouse        15    5
Exe          2    9
Lyn         14   14
Brook       25   24
Ditch       24   29
Fal         47   34
```

You can also use the dimnames() command to set both row and column names simultaneously

```
> species = c('Bbird', 'C.Finch', 'Gt.Tit', 'Sparrow', 'Robin', 'Thrush')
> habitats = c('Gdn', 'Hedge', 'Park', 'Field', 'Wood')
> dimnames(bird) = list(species, habitats)
> bird
        Gdn Hedge Park Field Wood
Bbird    47    10   40     2    2
C.Finch  19     3    5     0    2
Gt.Tit   50     0   10     7    0
Sparrow  46    16    8     4    0
Robin     9     3    0     0    2
Thrush    4     0    6     0    0
```

dimnames(our.object) = list(rows, columns)

```
> dimnames(bird) = list(c('Bbird', 'C.Finch', 'Gt.Tit', 'Sparrow', 'Robin',
'Thrush'), c('Gdn', 'Hedge', 'Park', 'Field', 'Wood'))
```

It is possible to type the names in one single command by specifying them explicitly.
You can reset names using NULL. You simply use this instead of a named object or a c() list of labels:

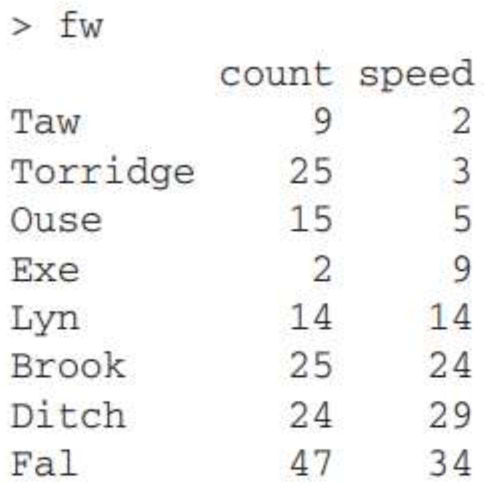
```
>dimnames(bird) = list(NULL, habitats)
>colnames(bird) = NULL
>names(fw) = NULL
```

TABLE 3-2: Commands to View and Set Names for Data Objects

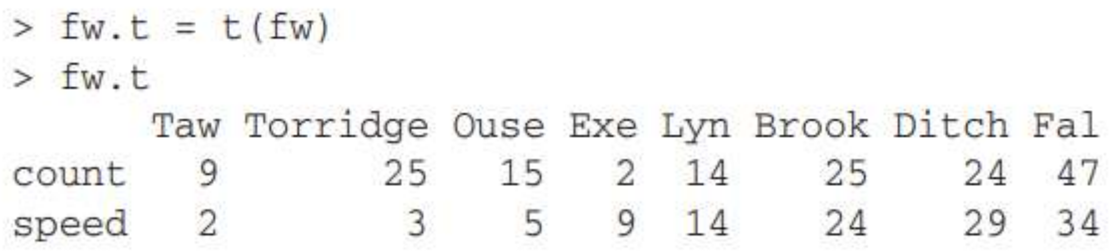
COMMAND	APPROPRIATE OBJECTS
`names()`	Works on list, matrix, and data frame
`row.names()`	Works on matrix and data frame
`rownames()`	Works on matrix and data frame
`colnames()`	Works on matrix and data frame
`dimnames(row, col)`	Will get and set names for matrix and data frame but `NULL` only works for matrix

Rotating Data Tables

you can easily rotate a frame or a matrix so that the rows become the columns and the columns become the rows
To do this you use the t() command. The final object is transposed so that there now are two rows rather than two columns

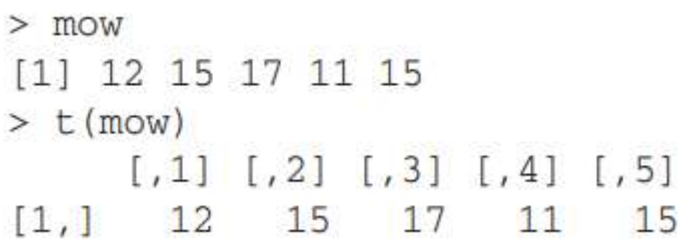


```
> fw
         count speed
Taw          9     2
Torridge    25     3
Ouse        15     5
Exe          2     9
Lyn         14    14
Brook       25    24
Ditch       24    29
Fal         47    34
```



```
> fw.t = t(fw)
> fw.t
      Taw Torridge Ouse Exe Lyn Brook Ditch Fal
count   9       25   15   2  14    25    24  47
speed   2        3    5   9  14    24    29  34
```

if you try the same t() command on a simple vector: now it is a matrix



```
> mow
[1] 12 15 17 11 15
> t(mow)
     [,1] [,2] [,3] [,4] [,5]
[1,]   12   15   17   11   15
```

Constructing Data Ob jects

Making Lists:

simplest complicated object is the list and this allows you to link together several vector items of varying type or size. Lists are useful because you can tie together more or less anything to make a single object that can be used to keep project items together. you use the list() command. there are five vectors; the first four are numeric and the last one is comprised of characters:To make these objects into a simple list

```
> mow; unmow; data3; data7; data8
[1] 12 15 17 11 15
[1] 8 9 7 9
[1]  6  7  8  7  6  3  8  9 10  7  6  9
[1] 23.0 17.0 12.5 11.0 17.0 12.0 14.5  9.0 11.0  9.0 12.5 14.5 17.0  8.0 21.0
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
> grass.list = list(mow, unmow, data3, data7, data8)
> grass.list
[[1]]
[1] 12 15 17 11 15

[[2]]
[1] 8 9 7 9

[[3]]
 [1]  6  7  8  7  6  3  8  9 10  7  6  9

[[4]]
 [1] 23.0 17.0 12.5 11.0 17.0 12.0 14.5  9.0 11.0  9.0 12.5 14.5 17.0  8.0 21.0

[[5]]
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

Lists are the simplest of the complicated objects that you can make and are useful to allow you to keep together disparate objects as one.

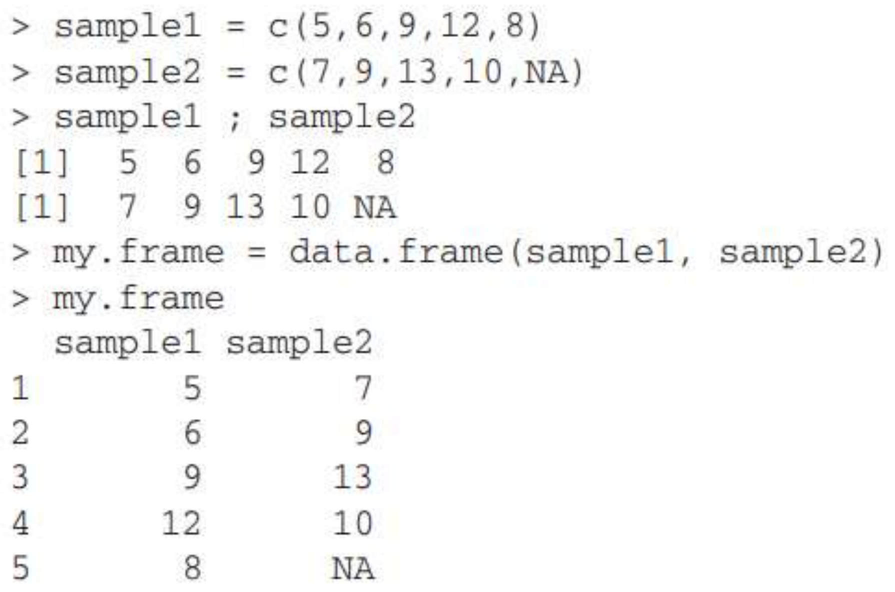
The list contains no names.so use the names() commands.

```
> names(grass.list) = c('mow', 'unmow', 'data3', 'data7', 'months')
```

```
> my.names = c('mow', 'unmow', 'data3', 'data7', 'months')
> names(grass.list) = my.names
```

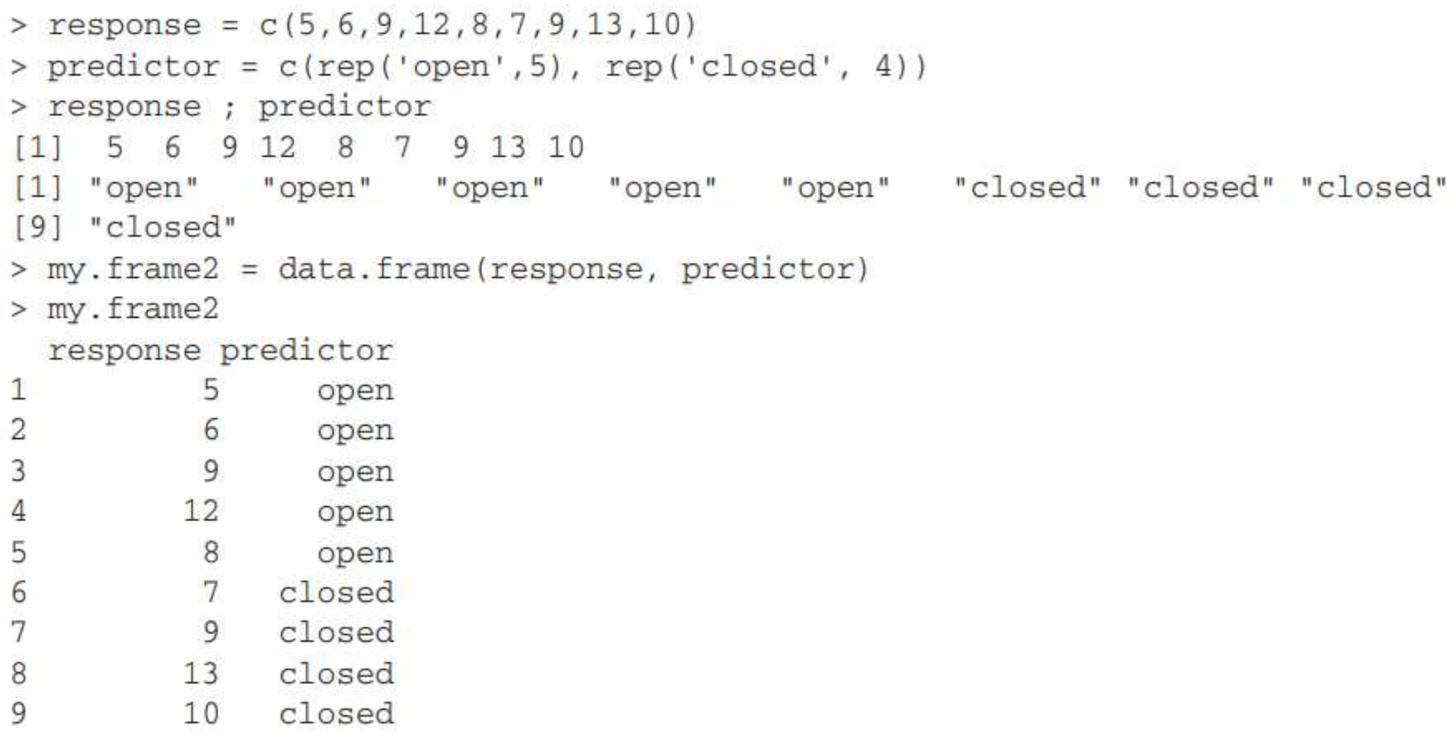
Making Data Frames

A data frame is a collection of columns of data, might have several columns of numbers and several of characters.
use the data.frame() command:
my.frame = data.frame(item1, item2, item3)
In the parentheses you give the names of the objects that you want to use as the columns, separated with commas
If you have vectors of unequal size, you must pad out the short ones yourself.



```
> sample1 = c(5,6,9,12,8)
> sample2 = c(7,9,13,10,NA)
> sample1 ; sample2
[1]  5  6  9 12  8
[1]  7  9 13 10 NA
> my.frame = data.frame(sample1, sample2)
> my.frame
  sample1 sample2
1       5       7
2       6       9
3       9      13
4      12      10
5       8      NA
```

If you had more than a few NA items, the typing could become quite tedious. You can use a command called rep() to repeat items multiple times.
rep(item, times)

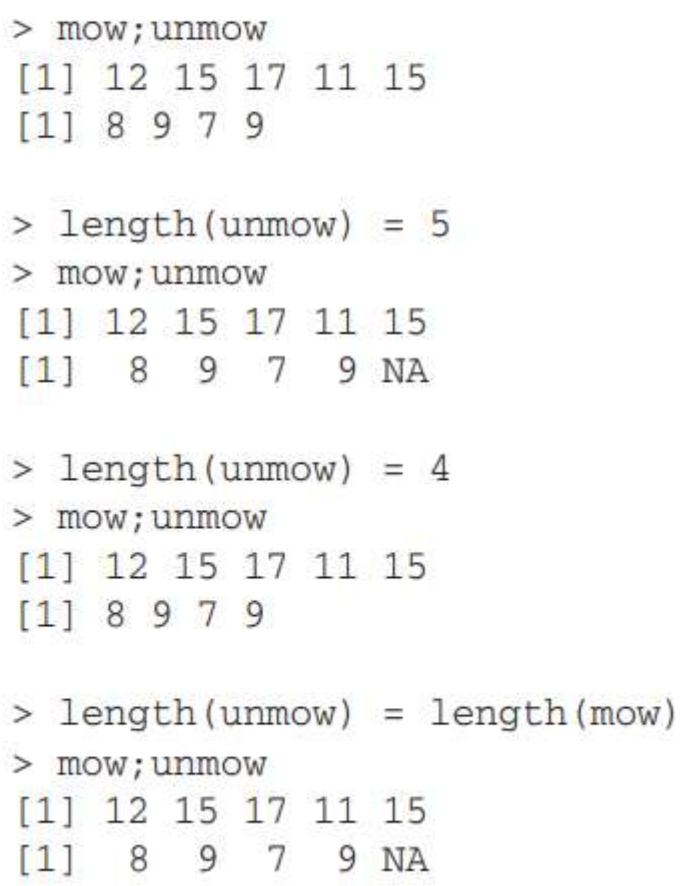


```
> response = c(5,6,9,12,8,7,9,13,10)
> predictor = c(rep('open',5), rep('closed', 4))
> response ; predictor
[1]  5  6  9 12  8  7  9 13 10
[1] "open"   "open"   "open"   "open"   "open"   "closed" "closed" "closed"
[9] "closed"
> my.frame2 = data.frame(response, predictor)
> my.frame2
  response predictor
1        5      open
2        6      open
3        9      open
4       12      open
5        8      open
6        7    closed
7        9    closed
8       13    closed
9       10    closed
```

use the length() command to extend or trim a vector
If you set the length() of the vector to a value greater than its current length, NA items are added at the end

If you set the length() to a value smaller than its current length, values at the end are lost.

length of the longest vector was used to set the length of the shorter ones,
This saves time, typing, and errors
length(short.vector) = length(long.vector)



```
> mow;unmow
[1] 12 15 17 11 15
[1] 8 9 7 9

> length(unmow) = 5
> mow;unmow
[1] 12 15 17 11 15
[1]  8  9  7  9 NA

> length(unmow) = 4
> mow;unmow
[1] 12 15 17 11 15
[1] 8 9 7 9

> length(unmow) = length(mow)
> mow;unmow
[1] 12 15 17 11 15
[1]  8  9  7  9 NA
```

Making Matrix Objects

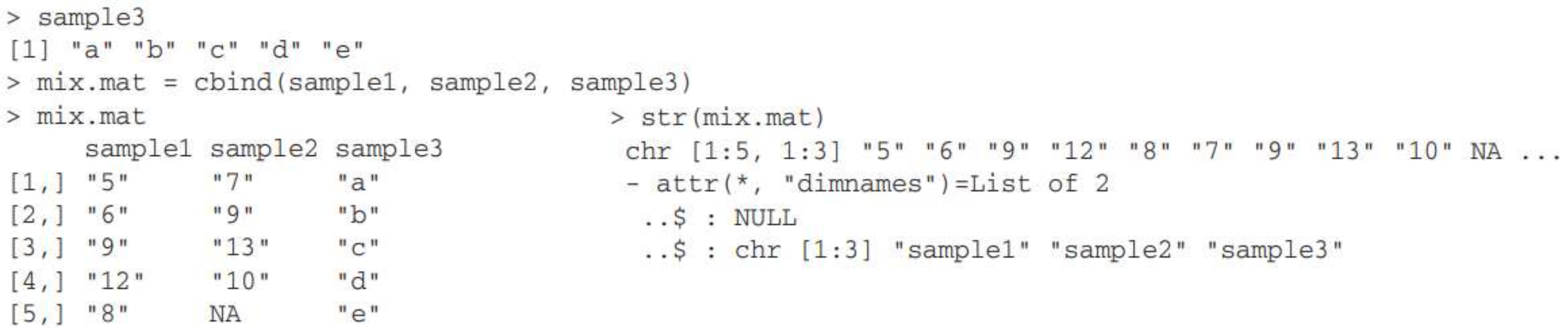
A matrix is also a rectangular object, and the columns of a matrix are also of equal length.

```
> sample1 ; sample2
[1] 5 6 9 12 8
[1] 7 9 13 10 NA
> cmat = cbind(sample1, sample2)
> cmat
 sample1 sample2
[1,] 5 7
[2,] 6 9
[3,] 9 13
[4,] 12 10
[5,] 8 NA
```

Use the cbind() command, If you have vectors of data that you want to form the columns of your matrix

You may also want to use the samples as rows, and to do that you use the rbind() command in a similar fashion
rmat = rbind(sample1, sample2)
> rmat [,1] [,2] [,3] [,4] [,5]
sample1 5 6 9 12 8
sample2 7 9 13 10 NA
Now the rows are named from the names of the vectors that were used to create the rows of the matrix

If you try to create a matrix using a mixture, the result is that all the items are converted to characters.



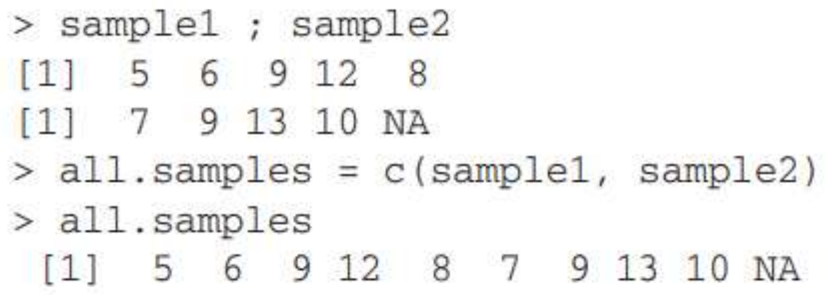
```
> sample3
[1] "a" "b" "c" "d" "e"
> mix.mat = cbind(sample1, sample2, sample3)
> mix.mat
     sample1 sample2 sample3
[1,] "5"     "7"     "a"
[2,] "6"     "9"     "b"
[3,] "9"     "13"    "c"
[4,] "12"    "10"    "d"
[5,] "8"     NA      "e"
```

```
> str(mix.mat)
 chr [1:5, 1:3] "5" "6" "9" "12" "8" "7" "9" "13" "10" NA ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:3] "sample1" "sample2" "sample3"
```

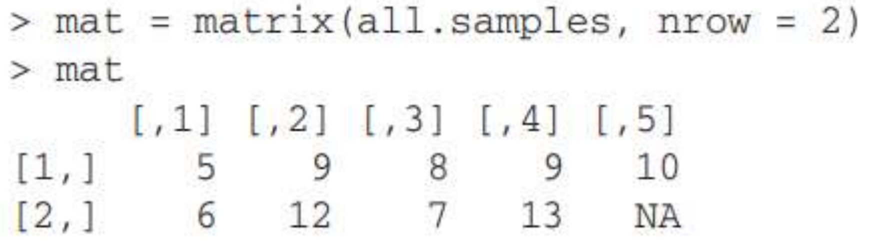
If you want to extract numbers from a “mixed” matrix, you must force the items to be numeric
> as.numeric(mix.mat[,1])
[1] 5 6 9 12 8

Note that the contents of the matrix have not been altered but merely extracted from the data matrix as numbers and NA is as it is.

another method to create a matrix from scratch: the matrix() command, use this for occasions where your data are in a single vector of values either numbers or characters. Mixed items will all end up as characters



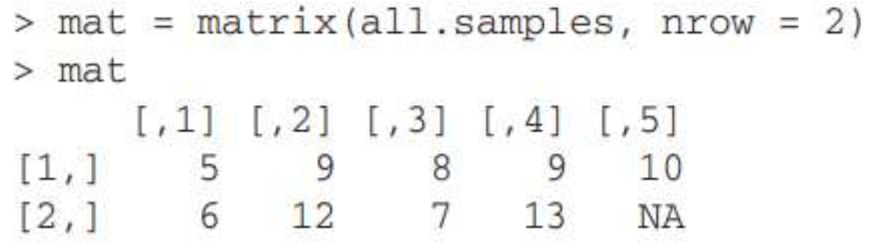
```
> sample1 ; sample2
[1]  5  6  9 12  8
[1]  7  9 13 10 NA
> all.samples = c(sample1, sample2)
> all.samples
 [1]  5  6  9 12  8  7  9 13 10 NA
```



```
> mat = matrix(all.samples, nrow = 2)
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,]    5    9    8    9   10
[2,]    6   12    7   13   NA
```

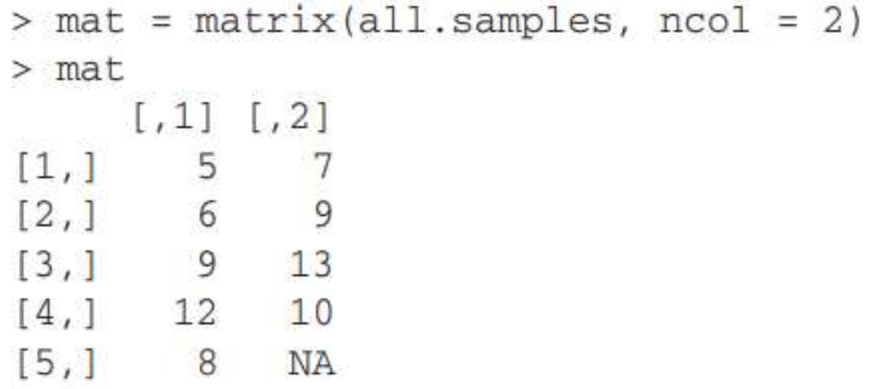
Now the matrix() command can be used to make a new matrix. When you do this you can choose how many rows or columns you want to make. there are two samples, you would want either two rows or two columns:

matrix was created with two rows using the nrow = 2 instruction



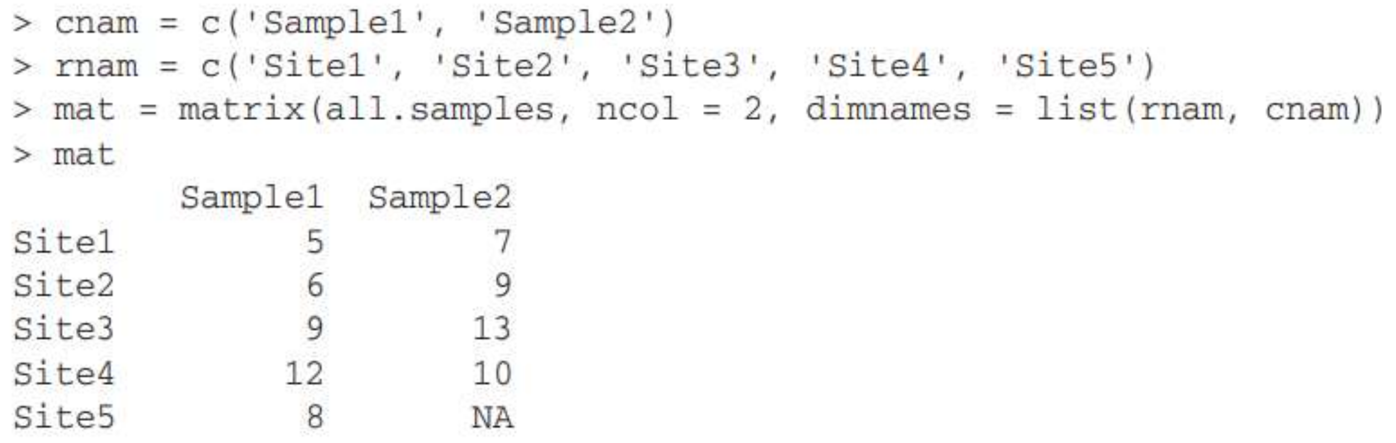
```
> mat = matrix(all.samples, nrow = 2)
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,]    5    9    8    9   10
[2,]    6   12    7   13   NA
```

the matrix with an appropriate number of columns



```
> mat = matrix(all.samples, ncol = 2)
> mat
     [,1] [,2]
[1,]    5    7
[2,]    6    9
[3,]    9   13
[4,]   12   10
[5,]    8   NA
```

Once a matrix is created you can use the rownames() or colnames() commands to create names for the rows or columns.



```
> cnam = c('Sample1', 'Sample2')
> rnam = c('Site1', 'Site2', 'Site3', 'Site4', 'Site5')
> mat = matrix(all.samples, ncol = 2, dimnames = list(rnam, cnam))
> mat
      Sample1 Sample2
Site1       5       7
Site2       6       9
Site3       9      13
Site4      12      10
Site5       8      NA
```

You might easily have specified the names all in one command

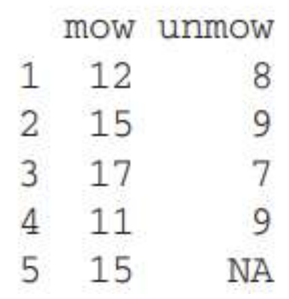
> mat = matrix(all.samples, ncol = 2, dimnames = list(c('Site1', 'Site2', 'Site3', 'Site4', 'Site5'), c('Sample1', 'Sample2')))

Re-ordering data Frames and matrix objects

Re-order a Data Frame and Add Additional Columns

Step 1 . Look at the data frame called grass2 simply by typing its name:

> grass2



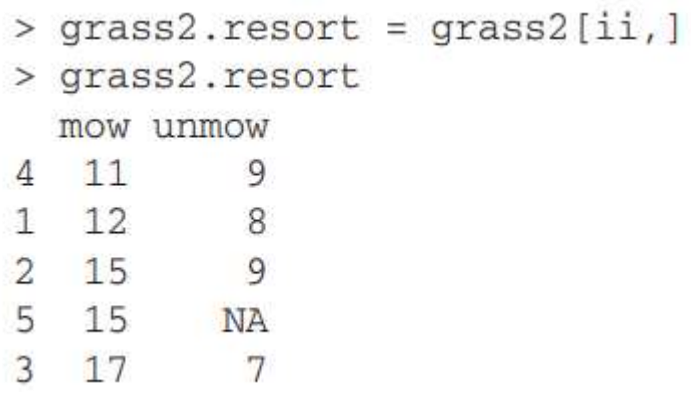
```
  mow unmow
1  12     8
2  15     9
3  17     7
4  11     9
5  15    NA
```

2. Create an index using the values in the mow column, with ties resolved by the unmow column:

> ii = with(grass2, order(mow, unmow))

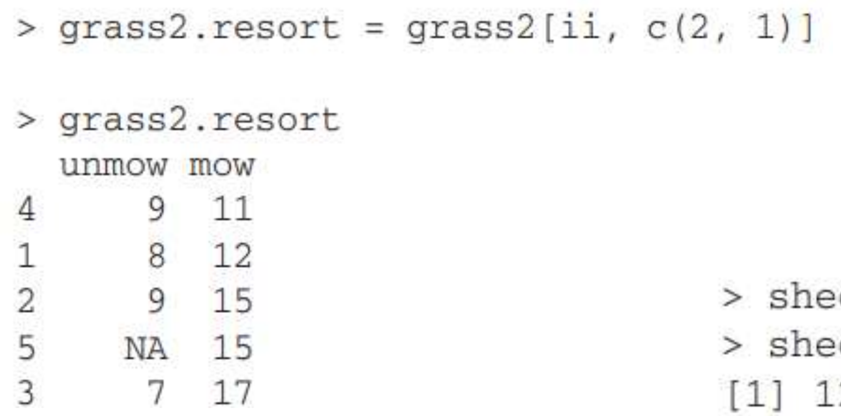
3. Look at the index you just created: > ii [1] 4 1 2 5 3

4. Now create a new data frame using the sort index you just made:



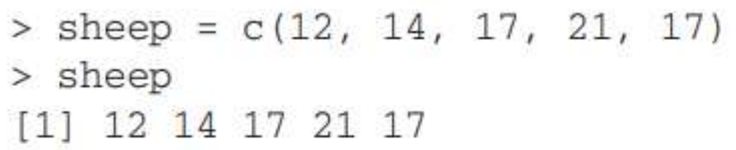
```
> grass2.resort = grass2[ii,]
> grass2.resort
  mow unmow
4  11     9
1  12     8
2  15     9
5  15    NA
3  17     7
```

5. Select a different order for the columns by specifying them in the square brackets in a new order:



```
> grass2.resort = grass2[ii, c(2, 1)]

> grass2.resort
  unmow mow
4     9  11
1     8  12
2     9  15
5    NA  15
3     7  17
```

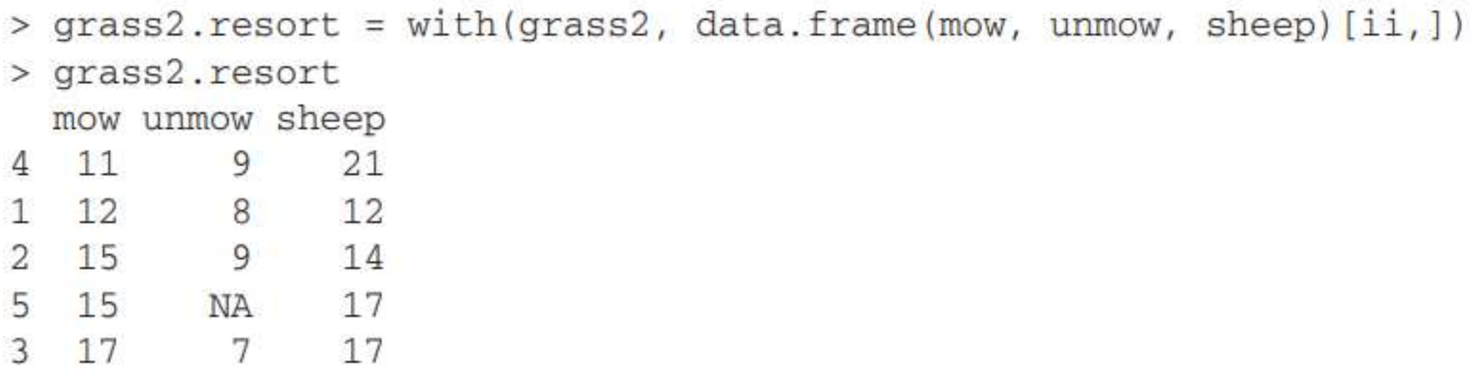
6. Now create a new vector of values:



```
> sheep = c(12, 14, 17, 21, 17)
> sheep
[1] 12 14 17 21 17
```

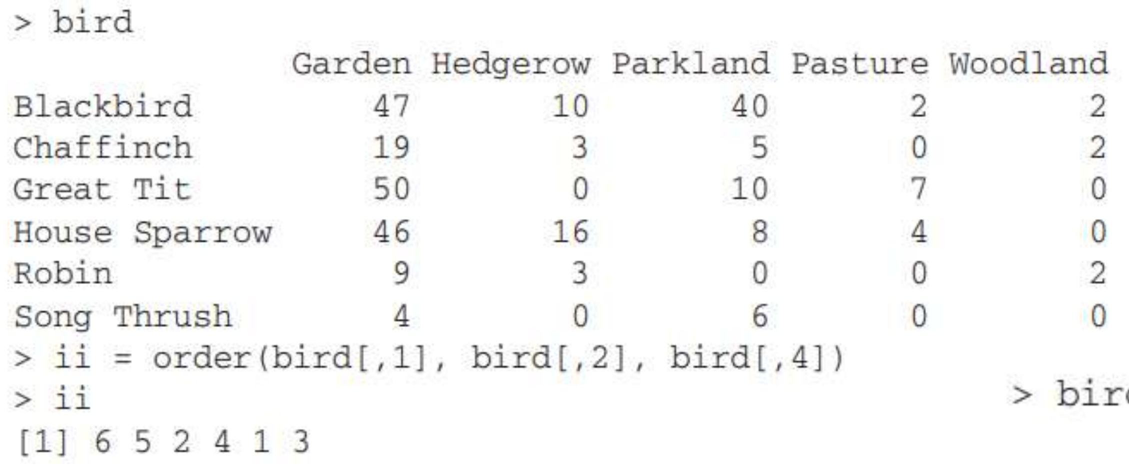
7. Finally, create a data frame that includes the original data plus the new vector you just created.

Use the sort index from before:



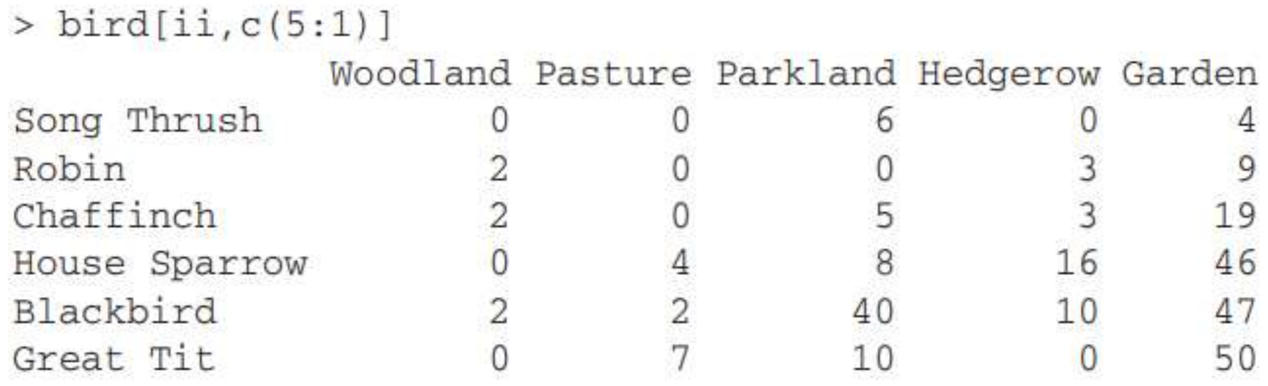
```
> grass2.resort = with(grass2, data.frame(mow, unmow, sheep)[ii,])
> grass2.resort
  mow unmow sheep
4  11     9    21
1  12     8    12
2  15     9    14
5  15    NA    17
3  17     7    17
```

Matrix objects can be re-ordered in a similar fashion to data frames. A new sort index is created: The first column is selected as the main re-sorted column and the second and fourth columns are used as tie-breakers



```
> bird
              Garden Hedgerow Parkland Pasture Woodland
Blackbird         47       10       40       2        2
Chaffinch         19        3        5       0        2
Great Tit         50        0       10       7        0
House Sparrow     46       16        8       4        0
Robin              9        3        0       0        2
Song Thrush        4        0        6       0        0
> ii = order(bird[,1], bird[,2], bird[,4])
> ii
[1] 6 5 2 4 1 3
```

you can simply specify what you want using the square brackets:



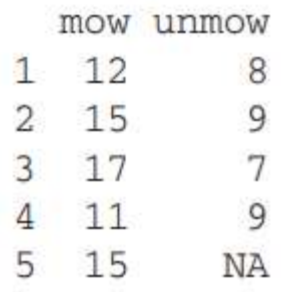
```
> bird[ii,c(5:1)]
              Woodland Pasture Parkland Hedgerow Garden
Song Thrush          0       0        6        0      4
Robin                2       0        0        3      9
Chaffinch            2       0        5        3     19
House Sparrow        0       4        8       16     46
Blackbird            2       2       40       10     47
Great Tit            0       7       10        0     50
```

Re-ordering data Frames and matrix objects

Re-order a Data Frame and Add Additional Columns

order() command, which was used to get an index relating to the order of items in a vector

1 . Look at the data frame called grass2 simply by typing its name: > grass2



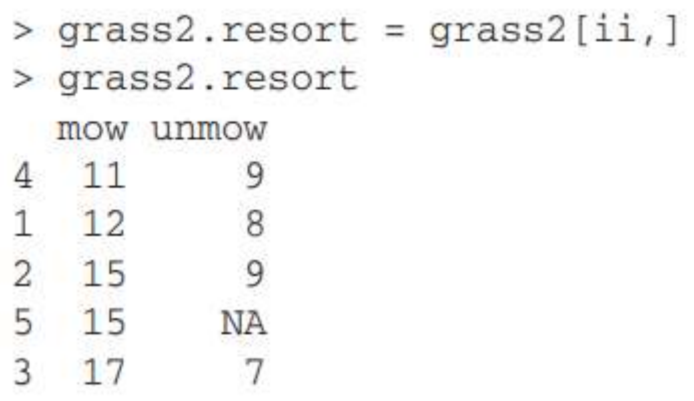
```
  mow unmow
1  12     8
2  15     9
3  17     7
4  11     9
5  15    NA
```

2. Create an index using the values in the mow column, with ties resolved by the unmow column:

> ii = with(grass2, order(mow, unmow))

3. Look at the index you just created: > ii [1] 4 1 2 5 3 4.

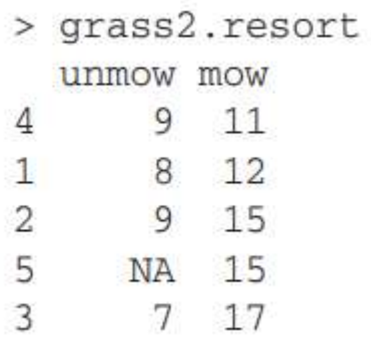
Now create a new data frame using the sort index you just made:



```
> grass2.resort = grass2[ii,]
> grass2.resort
  mow unmow
4  11     9
1  12     8
2  15     9
5  15    NA
3  17     7
```

5. Select a different order for the columns by specifying them in the square brackets in a new order:

> grass2.resort = grass2[ii, c(2, 1)]



```
> grass2.resort
  unmow mow
4     9  11
1     8  12
2     9  15
5    NA  15
3     7  17
```

6. Now create a new vector of values:

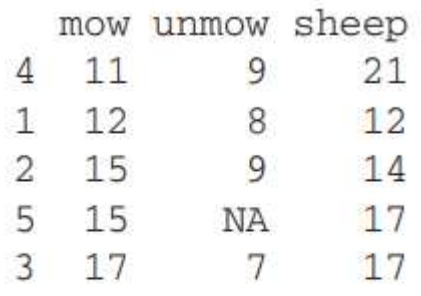
> sheep = c(12, 14, 17, 21, 17)

> sheep

[1] 12 14 17 21 17

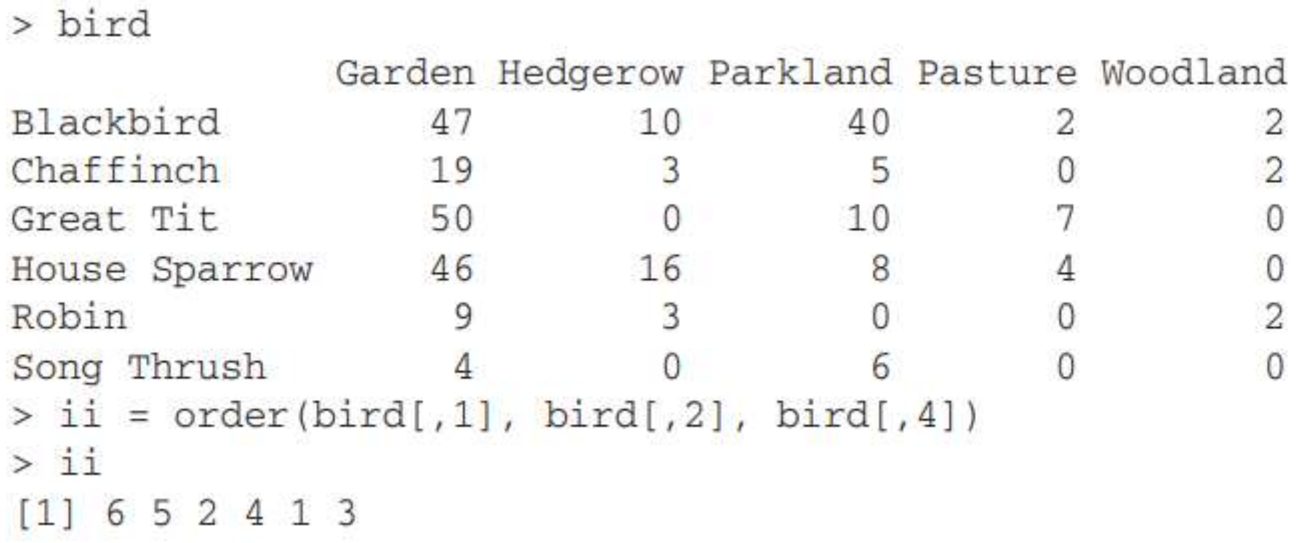
7. Finally, create a data frame that includes the original data plus the new vector you just created. Use the sort index from before:

> grass2.resort = with(grass2, data.frame(mow, unmow, sheep)[ii,]) > grass2.resort



```
  mow unmow sheep
4  11     9    21
1  12     8    12
2  15     9    14
5  15    NA    17
3  17     7    17
```

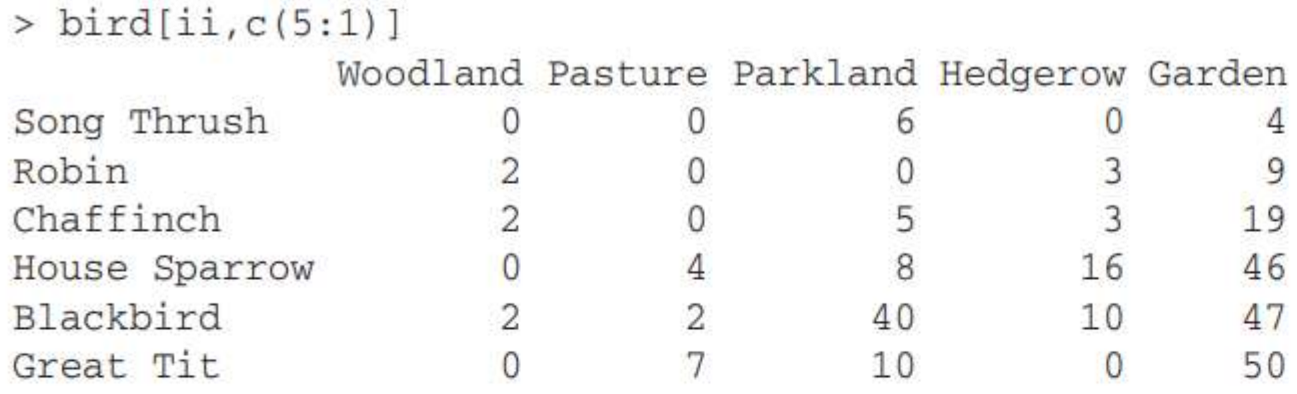
Matrix objects can be re-ordered in a similar fashion to data frames
example shows the bird data ,A new sort index is created. A new sort index is created



```
> bird
              Garden Hedgerow Parkland Pasture Woodland
Blackbird         47       10       40       2        2
Chaffinch         19        3        5       0        2
Great Tit         50        0       10       7        0
House Sparrow     46       16        8       4        0
Robin              9        3        0       0        2
Song Thrush        4        0        6       0        0
> ii = order(bird[,1], bird[,2], bird[,4])
> ii
[1] 6 5 2 4 1 3
```

The first column is selected as the main
re-sorted column and the second and
fourth columns are used as tie-breakers

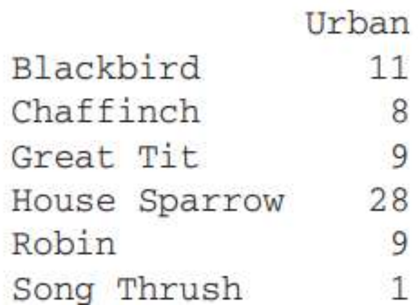
you can simply specify what you want
using the square brackets:



```
> bird[ii,c(5:1)]
              Woodland Pasture Parkland Hedgerow Garden
Song Thrush          0       0        6        0      4
Robin                2       0        0        3      9
Chaffinch            2       0        5        3     19
House Sparrow        0       4        8       16     46
Blackbird            2       2       40       10     47
Great Tit            0       7       10        0     50
```

In this instance the sort index is used to
re-order the rows; the columns are specified in
reverse order.

The new data you want to add can be in the
form of a simple vector or a matrix.
the Urban item is a vector whereas the bird.extra item is a matrix
> Urban [1] 11 8 9 28 9 1
> bird.extra
The cbind() command can be used to make the new matrix because there
either of the following commands will create the same result



```
              Urban
Blackbird        11
Chaffinch         8
Great Tit         9
House Sparrow    28
Robin             9
Song Thrush       1
```

In the first case the simple vector is used to form the extra column; the row names are already in place from the original bird matrix.

In the second example the matrix object is used as the source of the additional column, and once again the row names are transferred.

```
> cbind(bird, Urban)[ii,]
> cbind(bird, bird.extra)[ii,]

              Garden Hedgerow Parkland Pasture Woodland Urban
Song Thrush        4        0        6       0        0     1
Robin              9        3        0       0        2     9
Chaffinch         19        3        5       0        2     8
House Sparrow     46       16        8       4        0    28
Blackbird         47       10       40       2        2    11
Great Tit         50        0       10       7        0     9
```

You can also re-order the columns as you create the new matrix simply by typing the order you want them to appear in the square brackets. > cbind(bird, bird.extra)[ii,6:1]

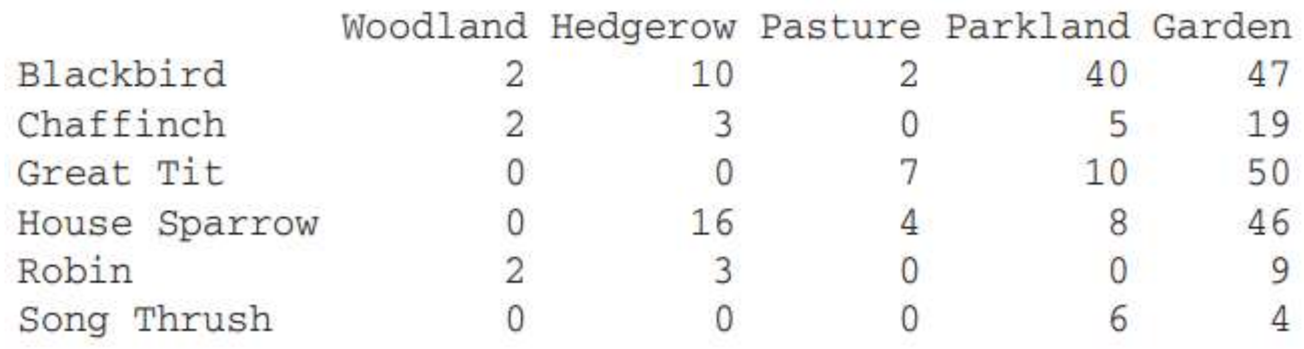
In the following example a new matrix is created using the existing data and the new Urban data:
> matrix(c(bird, Urban), ncol = 6)[ii,]

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    4    0    6    0    0    1
[2,]    9    3    0    0    2    9
[3,]   19    3    5    0    2    8
[4,]   46   16    8    4    0   28
[5,]   47   10   40    2    2   11
[6,]   50    0   10    7    0    9
```

The sort index is applied to re-order the rows.
You see that the names are lost;
you can add them afterwards using the rownames()
and colnames() commands or
you might add the dimnames = instruction to the matrix() command

use the order() command to alter the order of the columns as well as the rows, the sort index is created using the information in the third row. using the fourth and first rows as tie-breakers. the rbind() command is used to reassemble the matrix into a new order based on the sort index.

```
> ii = order(bird[3,], bird[4,], bird[1,])
> ii
 [1] 5 2 4 3 1

> rbind(bird[,ii])
```



```
              Woodland Hedgerow Pasture Parkland Garden
Blackbird            2       10       2       40     47
Chaffinch            2        3       0        5     19
Great Tit            0        0       7       10     50
House Sparrow        0       16       4        8     46
Robin                2        3       0        0      9
Song Thrush          0        0       0        6      4
```

Forms of data objects: Testing and Converting

R utilizes different forms of data objects, You can determine the form of object you have using the class() command.
You can convert an object from one form to another using a variety of commands .

Testing to See what type of object you have

The class() command gives you direct information in a single category about an object form. The following command can be used as a class test:

if(any(class(test.subject) == 'test.type') == TRUE) TRUE else FALSE

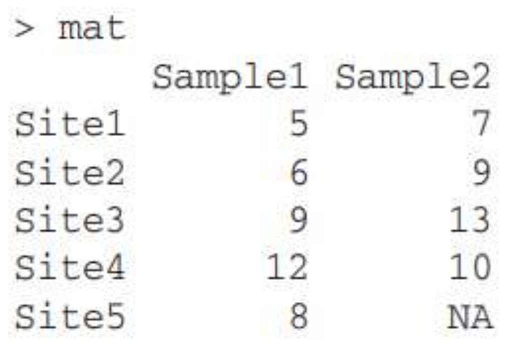
You simply replace the test.subject part with the name of the object you want to test and replace the test.type part with the class type you want to return as TRUE.
Some objects can be of more than one class-type and the **any() command** takes care of this. It looks at the entire result of the class() command and if any of them give a TRUE result, then you get a TRUE result. If all of them are FALSE, the final result is also FALSE.

Converting from One Object Form to Another

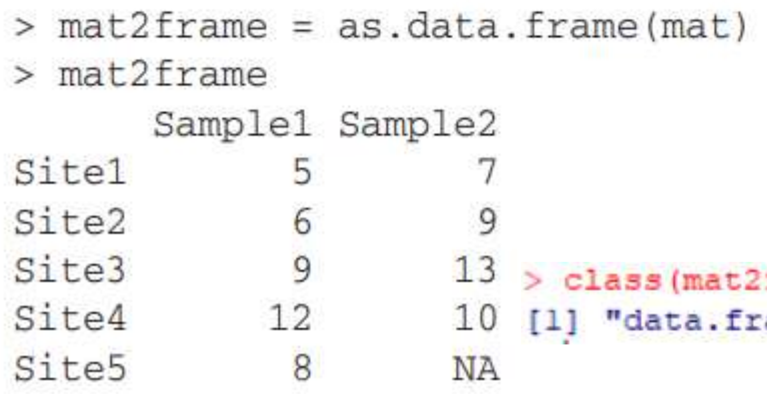
For day-to-day operations use the class () command.To write a program or script, use writing a script to run automatically you must get more in depth and use the programming method (the class-test).

Convert a Matrix to a Data Frame

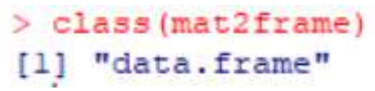
The matrix and data frame objects are similar in that they are both rectangular, two-dimensional objects. You can convert a matrix into a data frame using the as.data.frame() command.
you can use str() or class() commands to confirm that the object is in a new form.



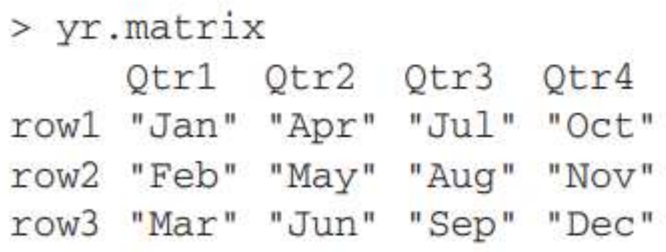
```
> mat
      Sample1 Sample2
Site1       5       7
Site2       6       9
Site3       9      13
Site4      12      10
Site5       8      NA
```



```
> mat2frame = as.data.frame(mat)
> mat2frame
      Sample1 Sample2
Site1       5       7
Site2       6       9
Site3       9      13
Site4      12      10
Site5       8      NA
```



```
> class(mat2frame)
[1] "data.frame"
```

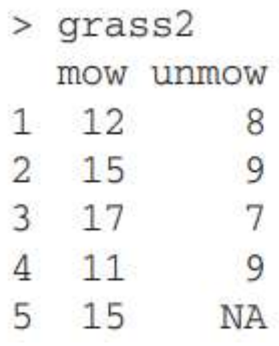


```
> yr.matrix
     Qtr1  Qtr2  Qtr3  Qtr4
row1 "Jan" "Apr" "Jul" "Oct"
row2 "Feb" "May" "Aug" "Nov"
row3 "Mar" "Jun" "Sep" "Dec"
```

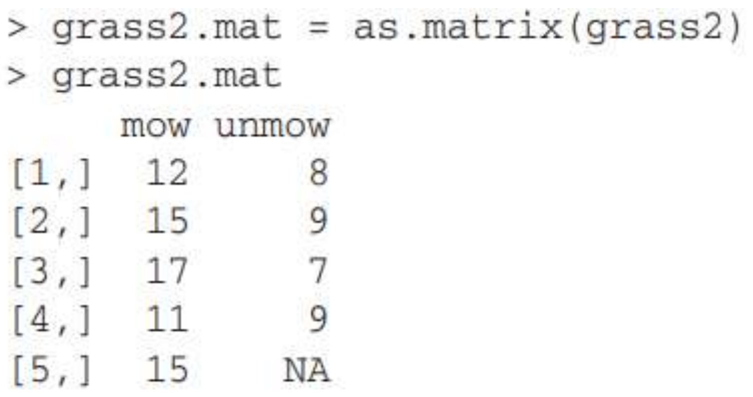
```
> yr.frame = as.data.frame(yr.matrix)
> yr.frame
 Qtr1 Qtr2 Qtr3 Qtr4
row1 Jan Apr Jul Oct
row2 Feb May Aug Nov
row3 Mar Jun Sep Dec
```

Convert a Data Frame into a Matrix

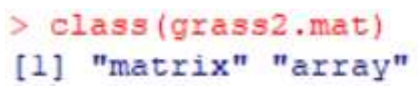
you can switch a data frame into a matrix using the as.matrix() command, if you have a data frame with mixed number and character variables the result will be a character matrix. If the data frame is all numeric, the matrix will be numeric. The final matrix is all numeric; remember that any NA items are kept as they are.



```
> grass2
  mow unmow
1  12     8
2  15     9
3  17     7
4  11     9
5  15    NA
```

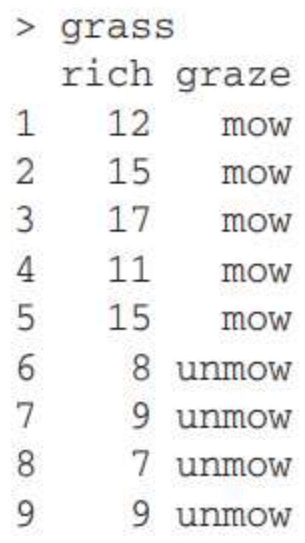


```
> grass2.mat = as.matrix(grass2)
> grass2.mat
     mow unmow
[1,]  12     8
[2,]  15     9
[3,]  17     7
[4,]  11     9
[5,]  15    NA
```



```
> class(grass2.mat)
[1] "matrix" "array"
```

The next example begins with a data frame that has two columns, one is numeric and the other is a character (in this instance actually a factor):Here you can tell that the results are all characters because they show the quotation marks.



```
> grass
  rich graze
1   12   mow
2   15   mow
3   17   mow
4   11   mow
5   15   mow
6    8 unmow
7    9 unmow
8    7 unmow
9    9 unmow
```

```
> grass.mat
 rich graze
1 "12" "mow"
2 "15" "mow"
3 "17" "mow"
4 "11" "mow"
5 "15" "mow"
6 " 8" "unmow"
7 " 9" "unmow“
8 " 7" "unmow"
9 " 9" "unmow"
```

Here you can tell that the results are all characters because they show the quotation marks

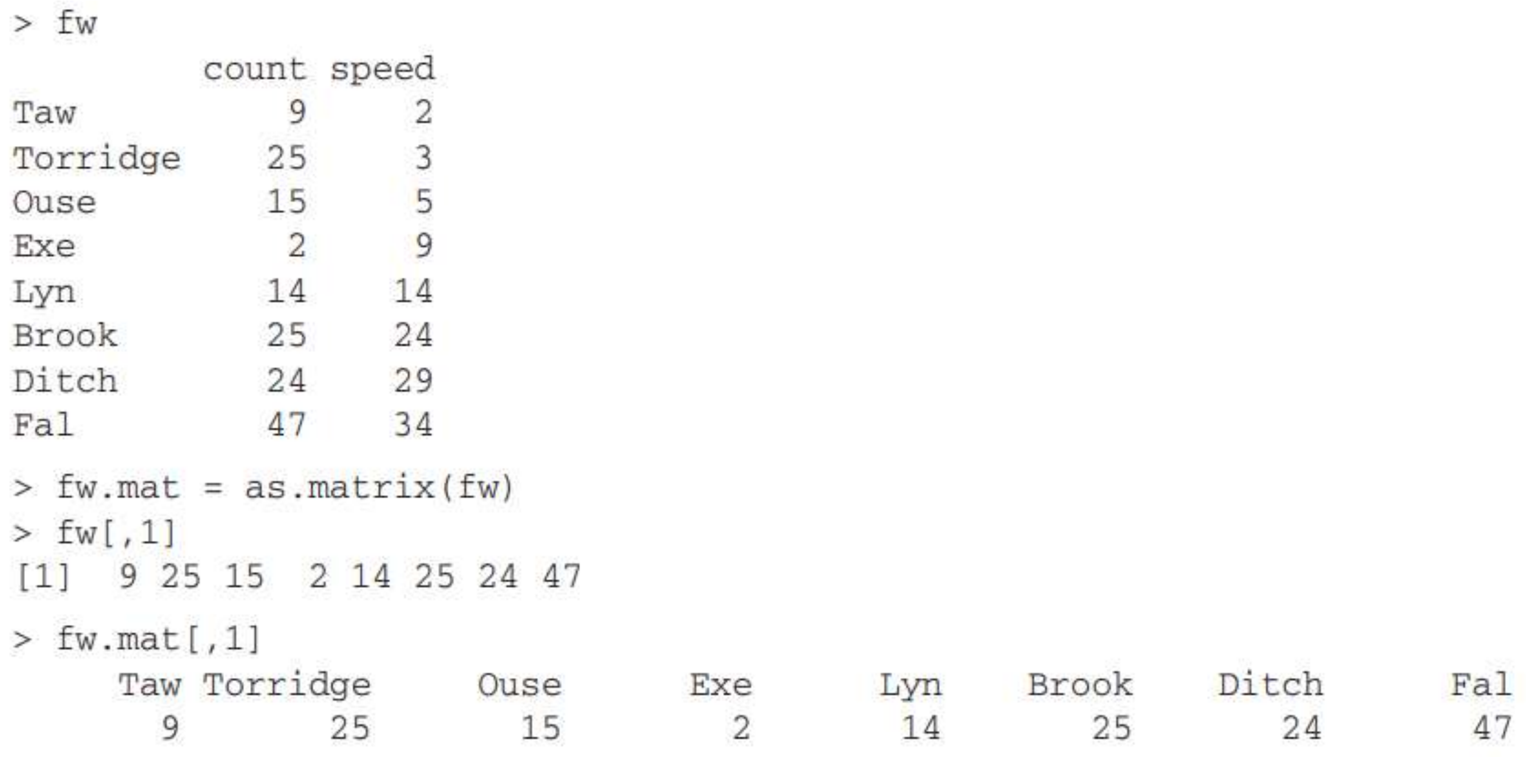
If your data frame contains row names , you can see an important difference between the frame and the matrix.

A new matrix is made from the existing data frame, and if you display each object they appear identical.

 if you display only the first column you see a difference.

In the data frame you get to see just the values.

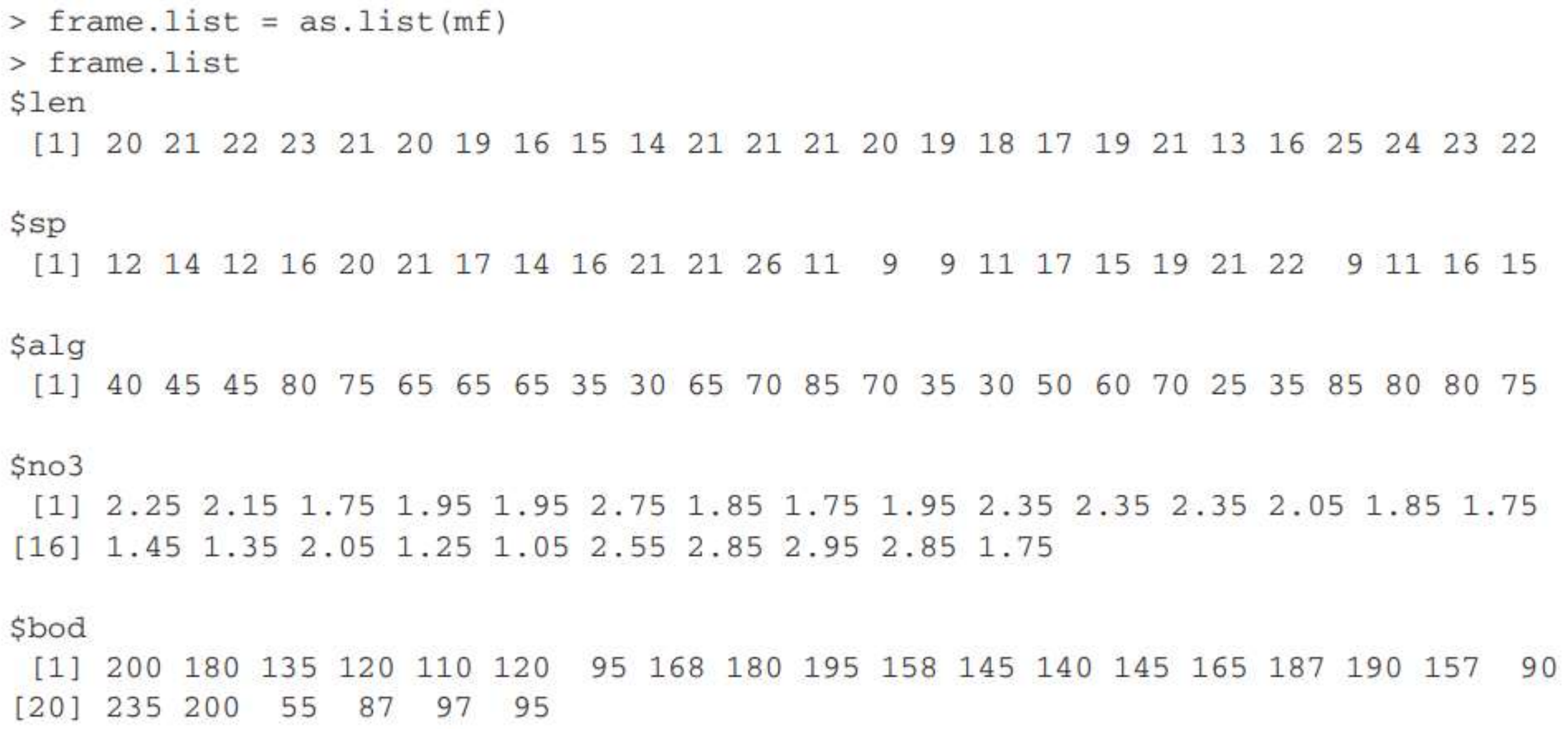
 If you display the matrix you see the row names as well as the data



```
> fw
         count speed
Taw          9     2
Torridge    25     3
Ouse        15     5
Exe          2     9
Lyn         14    14
Brook       25    24
Ditch       24    29
Fal         47    34
> fw.mat = as.matrix(fw)
> fw[,1]
[1]  9 25 15  2 14 25 24 47
> fw.mat[,1]
     Taw Torridge     Ouse      Exe      Lyn    Brook    Ditch      Fal
       9       25       15        2       14       25       24       47
```

Convert a Data Frame into a List

You can make a list object from a data frame very easily by using the as.list() command



```
> frame.list = as.list(mf)
> frame.list
$len
 [1] 20 21 22 23 21 20 19 16 15 14 21 21 21 20 19 18 17 19 21 13 16 25 24 23 22

$sp
 [1] 12 14 12 16 20 21 17 14 16 21 21 26 11  9  9 11 17 15 19 21 22  9 11 16 15

$alg
 [1] 40 45 45 80 75 65 65 65 35 30 65 70 85 70 35 30 50 60 70 25 35 85 80 80 75

$no3
 [1] 2.25 2.15 1.75 1.95 1.95 2.75 1.85 1.75 1.95 2.35 2.35 2.35 2.05 1.85 1.75
[16] 1.45 1.35 2.05 1.25 1.05 2.55 2.85 2.95 2.85 1.75

$bod
 [1] 200 180 135 120 110 120  95 168 180 195 158 145 140 145 165 187 190 157  90
[20] 235 200  55  87  97  95
```

Convert a Matrix into a List

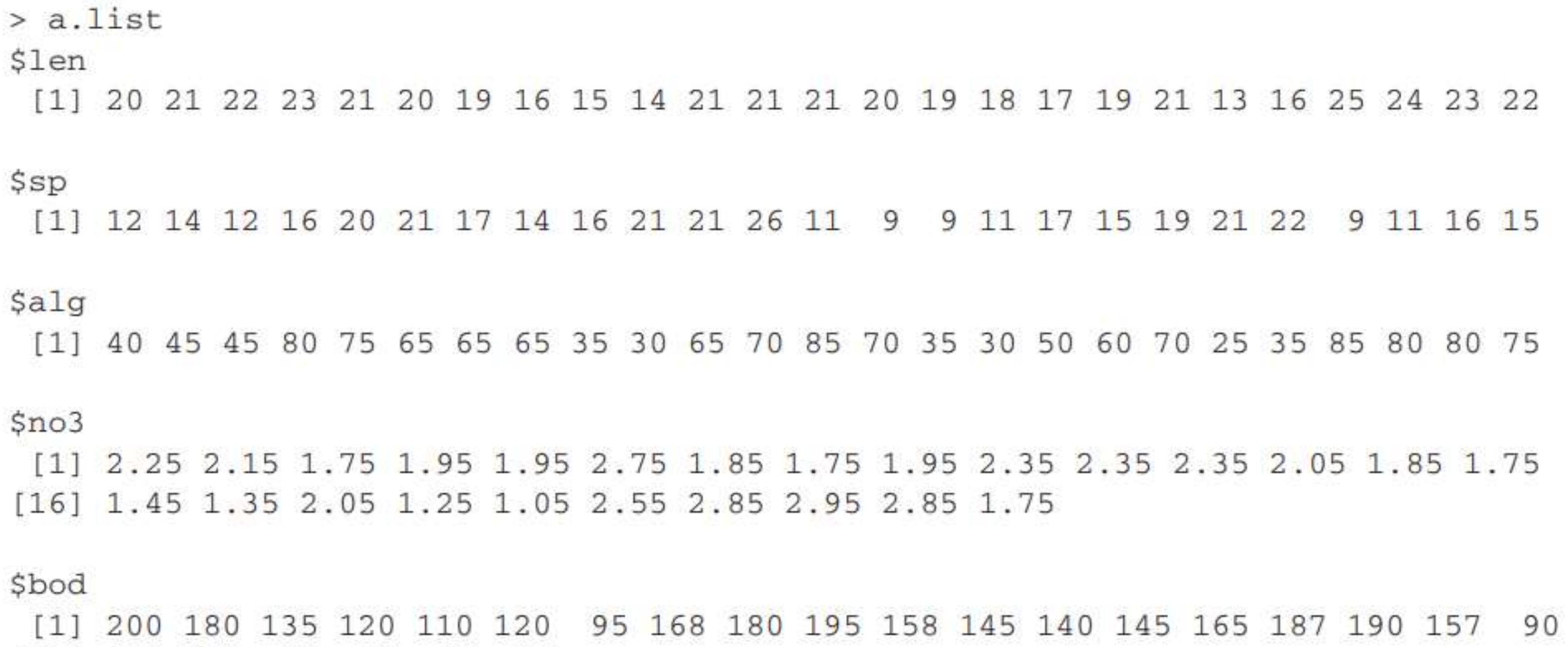
If you try to convert a matrix to a list, the answer is to convert the matrix to a data frame first and then convert this data frame into a list object.

> yr.list = as.list(as.data.frame(yr.matrix))

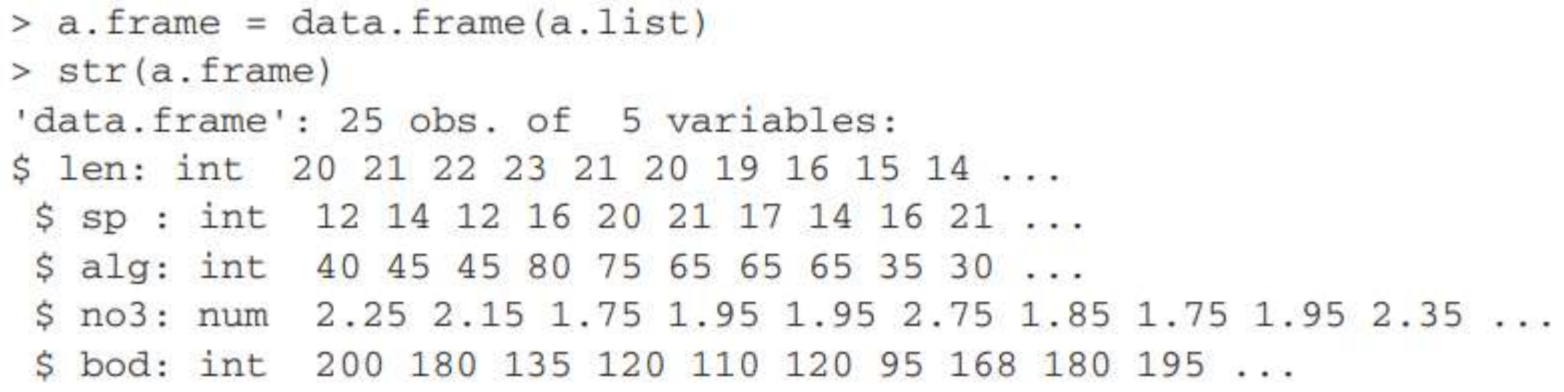
Convert a List to Something Else

If you start with a list it is generally a bit more difficult to convert it to another type of object. The main reasons are that the list can contain items of differing length, and these can be of differing sorts.

If all the items in the list happen to be the same length, it is easy to convert into a data frame using the data.frame() command. It does not matter if the individual items are numeric or character vectors, because the data frame can handle mixed items.



```
> a.list
$len
 [1] 20 21 22 23 21 20 19 16 15 14 21 21 21 20 19 18 17 19 21 13 16 25 24 23 22

$sp
 [1] 12 14 12 16 20 21 17 14 16 21 21 26 11  9  9 11 17 15 19 21 22  9 11 16 15

$alg
 [1] 40 45 45 80 75 65 65 65 35 30 65 70 85 70 35 30 50 60 70 25 35 85 80 80 75

$no3
 [1] 2.25 2.15 1.75 1.95 1.95 2.75 1.85 1.75 1.95 2.35 2.35 2.35 2.05 1.85 1.75
[16] 1.45 1.35 2.05 1.25 1.05 2.55 2.85 2.95 2.85 1.75

$bod
 [1] 200 180 135 120 110 120  95 168 180 195 158 145 140 145 165 187 190 157  90
```



```
> a.frame = data.frame(a.list)
> str(a.frame)
'data.frame': 25 obs. of  5 variables:
$ len: int  20 21 22 23 21 20 19 16 15 14 ...
 $ sp : int  12 14 12 16 20 21 17 14 16 21 ...
 $ alg: int  40 45 45 80 75 65 65 65 35 30 ...
 $ no3: num  2.25 2.15 1.75 1.95 1.95 2.75 1.85 1.75 1.95 2.35 ...
 $ bod: int  200 180 135 120 110 120 95 168 180 195 ...
```

You can see by using the str() command that the new object is indeed a data frame. If you have a list, convert to a data frame first and then make that into a matrix using the as.matrix() command;

> a.matrix = as.matrix(data.frame(a.list))

If your list object contains items that are of differing lengths, it is a little trickier to get them into a different form. The simplest way is to extract the components of the list to separate vectors, pad them with NA items as required, and then re-assemble into a data frame.

extract the individual parts by appending the $ to the list name

> algae = a.list$alg

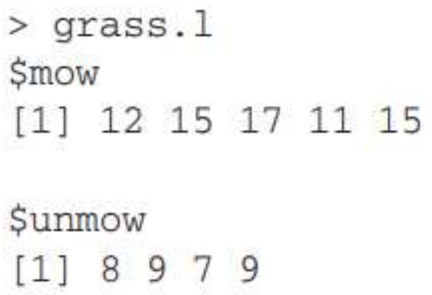
> algae

[1] 40 45 45 80 75 65 65 65 35 30 65 70 85 70 35 30 50 60 70 25 35 85 80 80 75

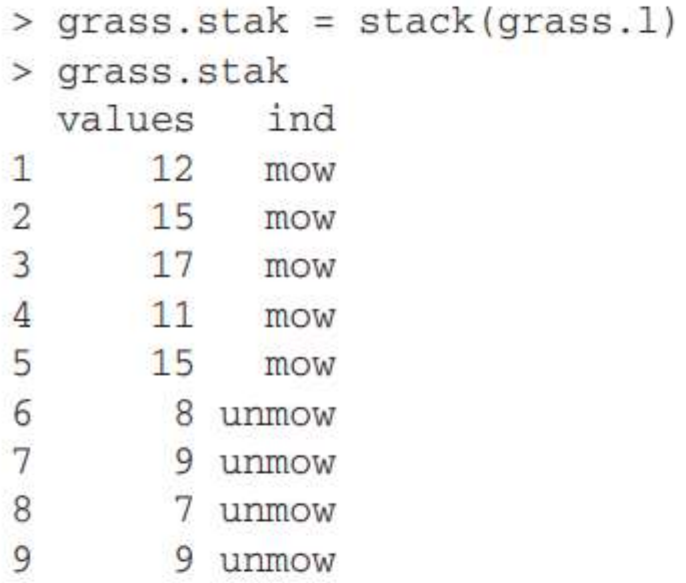
try to combine the list data into a two-column data frame where one column contained the values and the other, Use the stack() command to combine the values into a data frame

The command has taken each vector in the list and joined them together to make a column in the data frame.

The second column relates to the name of the list item



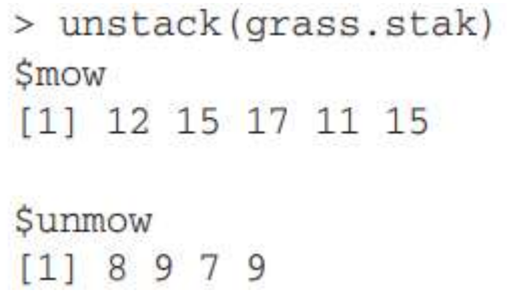
```
> grass.l
$mow
[1] 12 15 17 11 15

$unmow
[1] 8 9 7 9
```



```
> grass.stak = stack(grass.l)
> grass.stak
  values   ind
1     12   mow
2     15   mow
3     17   mow
4     11   mow
5     15   mow
6      8 unmow
7      9 unmow
8      7 unmow
9      9 unmow
```

```
> names(grass.stak) = c('species', 'cut')
> grass.stak
 species cut
1 12 mow
2 15 mow
3 17 mow
4 11 mow
5 15 mow
6 8 unmow
7 9 unmow
8 7 unmow
9 9 unmow
```

You can reverse the process using the unstack() command



```
> unstack(grass.stak)
$mow
[1] 12 15 17 11 15

$unmow
[1] 8 9 7 9
```

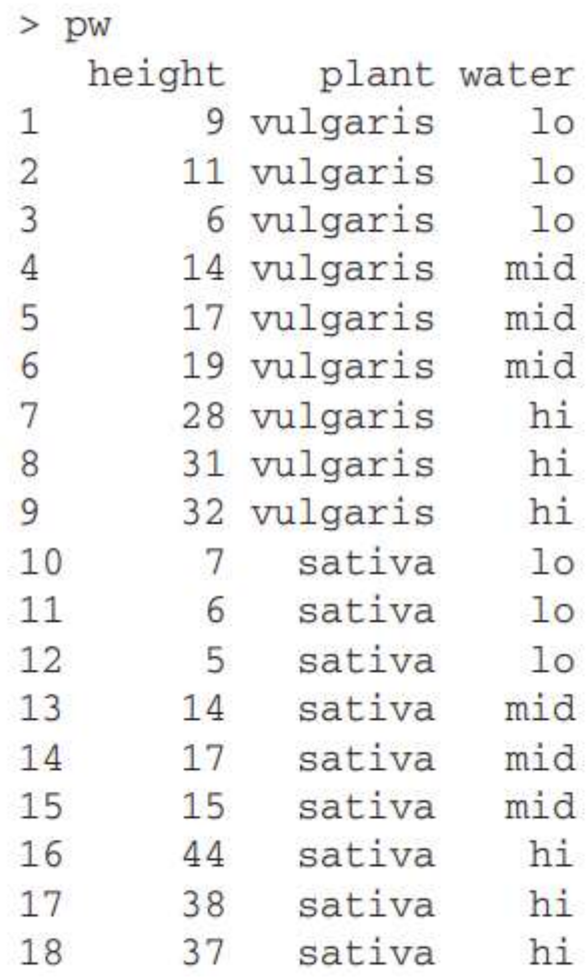
When you have a more complicated situation, there is a data frame with three columns; the first column holds numeric data and relates to the height of plants grown under various conditions.

The second column is a character variable, a factor relating to the species grown. The final column relates to the watering treatment.

You can use an additional instruction in the unstack() command to create the new two-column object.

unstack(object, form = response ~ grouping)

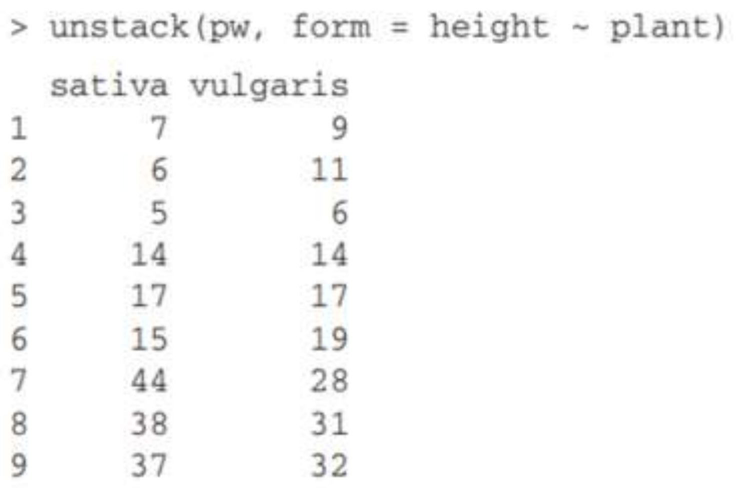
The form = part tells the unstack() command which columns of data to select from the original data frame



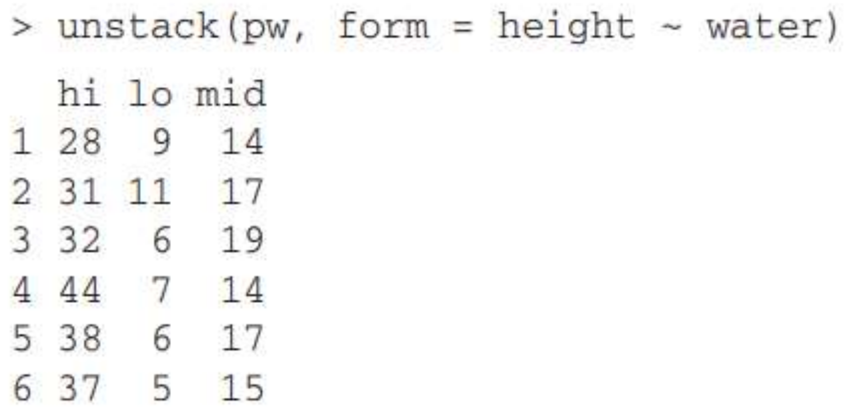
```
> pw
   height    plant water
1       9 vulgaris    lo
2      11 vulgaris    lo
3       6 vulgaris    lo
4      14 vulgaris   mid
5      17 vulgaris   mid
6      19 vulgaris   mid
7      28 vulgaris    hi
8      31 vulgaris    hi
9      32 vulgaris    hi
10      7   sativa    lo
11      6   sativa    lo
12      5   sativa    lo
13     14   sativa   mid
14     17   sativa   mid
15     15   sativa   mid
16     44   sativa    hi
17     38   sativa    hi
18     37   sativa    hi
```

the plant variable was chosen and this results in two columns of data, one for each of the two species.

In the second example the water variable was chosen and this results in three columns that correspond to each of the three original watering treatments

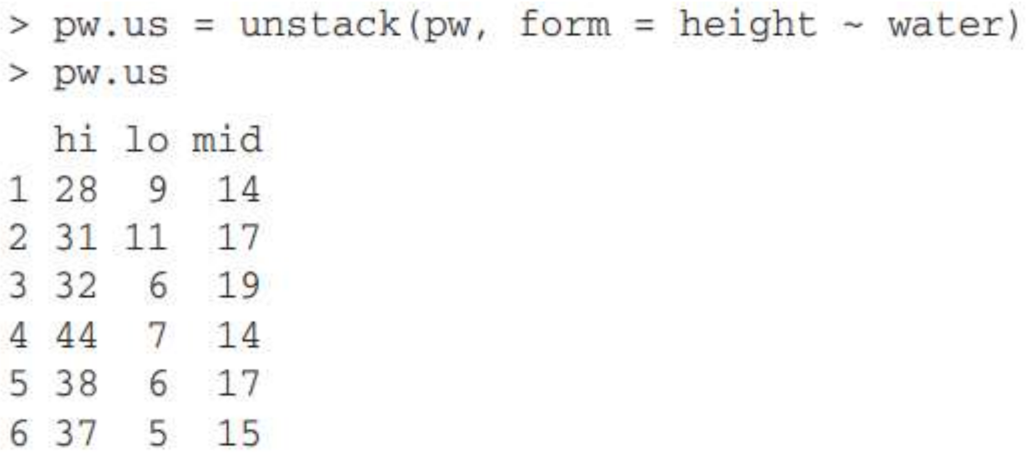


```
> unstack(pw, form = height ~ plant)
  sativa vulgaris
1      7        9
2      6       11
3      5        6
4     14       14
5     17       17
6     15       19
7     44       28
8     38       31
9     37       32
```



```
> unstack(pw, form = height ~ water)
  hi lo mid
1 28  9  14
2 31 11  17
3 32  6  19
4 44  7  14
5 38  6  17
6 37  5  15
```

You can choose which columns to stack up by adding the select = instruction to the stack() command.

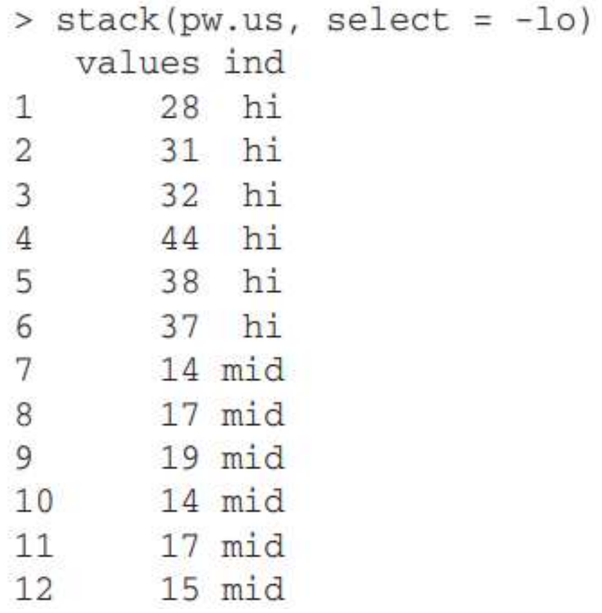


```
> pw.us = unstack(pw, form = height ~ water)
> pw.us
  hi lo mid
1 28  9  14
2 31 11  17
3 32  6  19
4 44  7  14
5 38  6  17
6 37  5  15
```

```
> stack(pw.us, select = c(hi, lo))
     values ind
     1 28    hi
     2 31    hi
     3 32    hi
     4 44    hi
     5 38    hi
     6 37    hi
     7 9     lo
     8 11    lo
     9 6     lo
     10 7    lo
     11 6    lo
     12 5    lo
```

In this case the required columns were given in a c() command as part of the select = instruction.

In the following example the columns you do not want are given by prefacing the name with a minus sign (-):



```
> stack(pw.us, select = -lo)
   values ind
1      28  hi
2      31  hi
3      32  hi
4      44  hi
5      38  hi
6      37  hi
7      14 mid
8      17 mid
9      19 mid
10     14 mid
11     17 mid
12     15 mid
```