5

Factors

Conceptually, factors are variables in R which take on a limited number of different values; such variables are often referred to as categorical variables. One of the most important uses of factors is in statistical modeling; since categorical variables enter into statistical models differently than continuous variables, storing data as factors insures that the modeling functions will treat such data correctly.

5.1 Using Factors

Factors in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed. The `factor` function is used to create a factor. The only required argument to `factor` is a vector of values which will be returned as a vector of factor values. Both numeric and character variables can be made into factors, but a factor's levels will always be character values. You can see the possible levels for a factor by calling the `levels` function; the `nlevels` function will return the number of levels of a factor.

To change the order in which the levels will be displayed from their default sorted order, the `levels=` argument can be given a vector of all the possible values of the variable in the order you desire. If the ordering should also be used when performing comparisons, use the optional `ordered=TRUE` argument. In this case, the factor is known as an ordered factor.

The levels of a factor are used when displaying the factor's values. You can change these levels at the time you create a factor by passing a vector with the new values through the `labels=` argument. Note that this actually changes the internal levels of the factor, and to change the labels of a factor after it has been created, the assignment form of the `levels` function is used. To illustrate this point, consider a factor taking on integer values which we want to display as roman numerals:

```
> data = c(1,2,2,3,1,2,3,3,1,2,3,3,1)
> fdata = factor(data)
> fdata
 [1] 1 2 2 3 1 2 3 3 1 2 3 3 1
Levels: 1 2 3
> rdata = factor(data,labels=c("I","II","III"))
> rdata
 [1] I   II  II  III I   II  III III I   II  III III I
Levels: I II III
```

To convert the default factor `fdata` to roman numerals, we use the assignment form of the `levels` function:

```
> levels(fdata) = c('I','II','III')
> fdata
 [1] I   II  II  III I   II  III III I   II  III III I
Levels: I II III
```

Factors represent a very efficient way to store character values, because each unique character value is stored only once, and the data itself is stored as a vector of integers. Because of this, `read.table` will automatically convert character variables to factors unless the `as.is=TRUE` or `stringsAsFactors=FALSE` arguments are specified, or the `stringsAsFactors` system option is set to `FALSE`. See Section 2.2 for details.

As an example of an ordered factor, consider data consisting of the names of months:

```
> mons = c("March","April","January","November","January",
+ "September","October","September","November","August",
+ "January","November","November","February","May","August",
+ "July","December","August","August","September","November",
+ "February","April")
> mons = factor(mons)
> table(mons)
mons
    April    August  December  February   January      July
        2         4         1         2         3         1
    March       May  November   October September
        1         1         5         1         3
```

Although the months clearly have an ordering, this is not reflected in the output of the `table` function. Additionally, comparison operators are not supported for unordered factors. Creating an ordered factor solves these problems:

```
> mons = factor(mons,levels=c("January","February","March",
+              "April","May","June","July","August","September",
+              "October","November","December"),ordered=TRUE)
> mons[1] < mons[2]
[1] TRUE
```

```
> table(mons)
mons
  January  February     March     April       May      June
        3         2         1         2         1         0
     July    August September   October  November  December
        1         4         3         1         5         1
```

The order in which the levels are displayed is determined by the order in which they appear in the `levels=` argument to `factor`.

In the previous example, the levels of the factors had a natural ordering. Sometimes, a factor needs to be reordered on the basis of some property of that factor. For example, consider the `InsectSpray` data frame, which contains data on the numbers of insects seen (`count`) when an experimental unit was treated with one of six sprays (`spray`). The `spray` variable is stored as a factor with default ordering:

```
> levels(InsectSprays$spray)
[1] "A" "B" "C" "D" "E" "F"
```

Suppose we wish to reorder the factor levels of `spray` based on the mean value of the `count` variable for each level of `spray`. The `reorder` function takes three arguments: a factor, a vector of values on which the reordering is based, and a function to operate on those values for each factor level. Suppose we wish to reorder the levels of `spray` so that they are stored in the order of the mean value of `count` for each level of `spray`:

```
> InsectSprays$spray = with(InsectSprays,
+                           reorder(spray,count,mean))
> levels(InsectSprays$spray)
[1] "C" "E" "D" "A" "B" "F"
```

When `reorder` is used, it assigns an attribute called `scores` which contains the value used for the reordering:

```
> attr(InsectSprays$spray,'scores')
        A         B         C         D         E         F
14.500000 15.333333  2.083333  4.916667  3.500000 16.666667
```

As always, changes to system datasets are made in the local workspace; the original dataset is unchanged.

For some statistical procedures, the interpretation of results can be simplified by forcing a particular order to a factor; in particular, it may be useful to choose a "reference" level, which should be the first level of the factor. The `relevel` function allows you to choose a reference level, which will then be treated as the first level of the factor. For example, to make level "`C`" of `InsectSprays$spray` the first level, we could call `relevel` as follows:

```
> levels(InsectSprays$spray)
[1] "A" "B" "C" "D" "E" "F"
```

```
> InsectSprays$spray = relevel(InsectSprays$spray,'C')
> levels(InsectSprays$spray)
[1] "C" "A" "B" "D" "E" "F"
```

5.2 Numeric Factors

While it may be necessary to convert a numeric variable to a factor for a particular application, it is often very useful to convert the factor back to its original numeric values, since even simple arithmetic operations will fail when using factors. Since the `as.numeric` function will simply return the internal integer values of the factor, the conversion must be done using the `levels` attribute of the factor, or by first converting the factor to a character value using `as.character`.

Suppose we are studying the effects of several levels of a fertilizer on the growth of a plant. For some analyses, it might be useful to convert the fertilizer levels to an ordered factor:

```
> fert = c(10,20,20,50,10,20,10,50,20)
> fert = factor(fert,levels=c(10,20,50),ordered=TRUE)
> fert
[1] 10 20 20 50 10 20 10 50 20
Levels: 10 < 20 < 50
```

If we wished to calculate the mean of the original numeric values of the `fert` variable, we would have to convert the values using the `levels` function or `as.character`:

```
> mean(fert)
[1] NA
Warning message:
argument is not numeric or logical:
      returning NA in: mean.default(fert)
> mean(as.numeric(levels(fert)[fert]))
[1] 23.33333
> mean(as.numeric(as.character(fert)))
[1] 23.33333
```

Either method will achieve the desired result.

5.3 Manipulating Factors

When a factor is first created, all of its levels are stored along with the factor, and if subsets of the factor are extracted, they will retain all of the original levels. This can create problems when constructing model matrices and may or may not be useful when displaying the data using, say, the `table` function.

As an example, consider a random sample from the `letters` vector, which is part of the base R distribution:

```
> lets = sample(letters,size=100,replace=TRUE)
> lets = factor(lets)
> table(lets[1:5])

a b c d e f g h i j k l m n o p q r s t u v w x y z
0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
```

Even though only five of the levels were actually represented, the `table` function shows the frequencies for all of the levels of the original factors. To change this, we can use the `drop=TRUE` argument to the subscripting operator. When used with factors, this argument will remove the unused levels:

```
> table(lets[1:5,drop=TRUE])

c h j w x
1 1 1 1 1
```

A similar result can be achieved by creating a new factor:

```
> table(factor(lets[1:5]))

c h j w x
1 1 1 1 1
```

To exclude certain levels from appearing in a factor, the `exclude=` argument can be passed to `factor`. By default, the missing value (`NA`) is excluded from factor levels; to create a factor that includes missing values from a numeric variable, use `exclude=NULL`.

Care must be taken when combining variables which are factors, because the `c` function will interpret the factors as integers. To combine factors, they should first be converted back to their original values (through the `levels` function), then catenated and converted to a new factor:

```
> fact1 = factor(sample(letters,size=10,replace=TRUE))
> fact2 = factor(sample(letters,size=10,replace=TRUE))
> fact1
 [1] o b i v q n q w e z
Levels: b e i n o q v w z
> fact2
 [1] b a s b l r g m z o
Levels: a b g l m o r s z
> fact12 = factor(c(levels(fact1)[fact1],
                  levels(fact2)[fact2]))
> fact12
 [1] o b i v q n q w e z b a s b l r g m z o
Levels: a b e g i l m n o q r s v w z
```

5.4 Creating Factors from Continuous Variables

The `cut` function is used to convert a numeric variable into a factor. The `breaks=` argument to `cut` is used to describe how ranges of numbers will be converted to factor values. If a number is provided through the `breaks=` argument, the resulting factor will be created by dividing the range of the variable into that number of equal-length intervals; if a vector of values is provided, the values in the vector are used to determine the breakpoints. Note that if a vector of values is provided, the number of levels of the resultant factor will be one less than the number of values in the vector.

For example, consider the `women` dataset, which contains height and weights for a sample of women. If we wanted to create a factor corresponding to `weight`, with three equally spaced levels, we could use the following:

```
> wfact = cut(women$weight,3)
> table(wfact)
wfact
(115,131] (131,148] (148,164]
        6         5         4
```

Notice that the default label for factors produced by `cut` contains the actual range of values that were used to divide the variable into factors. The `pretty` function can be used to choose cut points that are round numbers, but it may not return the number of levels that's actually desired:

```
> wfact = cut(women$weight,pretty(women$weight,3))
> wfact
 [1] (100,120] (100,120] (100,120] (120,140]
 [5] (120,140] (120,140] (120,140] (120,140]
 [9] (120,140] (140,160] (140,160] (140,160]
[13] (140,160] (140,160] (160,180]
4 Levels: (100,120] (120,140] (140,160] (160,180]
> table(wfact)
wfact
(100,120] (120,140] (140,160] (160,180]
        3         6         5         1
```

The `labels=` argument to `cut` allows you to specify the levels of the factors:

```
> wfact = cut(women$weight,3,labels=c('Low','Medium','High'))
> table(wfact)
wfact
   Low Medium   High
     6      5      4
```

To produce factors based on percentiles of your data (for example, quartiles or deciles), the `quantile` function can be used to generate the `breaks=` argument, insuring nearly equal numbers of observations in each of the levels of the factor:

```
> wfact = cut(women$weight,quantile(women$weight,(0:4)/4))
> table(wfact)
wfact
(115,124] (124,135] (135,148] (148,164]
        3         4         3         4
```

5.5 Factors Based on Dates and Times

As mentioned in Section 4.6, there are a number of ways to create factors from date/time objects. If you wish to create a factor based on one of the components of that date, you can extract it with `strftime` and convert it to a factor directly. For example, we can use the `seq` function to create a vector of dates representing each day of the year:

```
> everyday = seq(from=as.Date('2005-1-1'),
+                 to=as.Date('2005-12-31'),by='day')
```

To create a factor based on the month of the year in which each date falls, we can extract the month name (full or abbreviated) using `format`:

```
> cmonth = format(everyday,'%b')
> months = factor(cmonth,levels=unique(cmonth),ordered=TRUE)
> table(months)
months
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
 31  28  31  30  31  30  31  31  30  31  30  31
```

Since `unique` returns unique values in the order they are encountered, the `levels` argument will provide the month abbreviations in the correct order to produce a properly ordered factor.

For more details on formatting dates, see Section 4.3.

Sometimes more flexibility can be achieved by using the `cut` function, which understands time units of `months`, `days`, `weeks`, and `years` through the `breaks=` argument. (For date/time values, units of `hours`, `minutes`, and `seconds` can also be used.) For example, to format the days of the year based on the week in which they fall, we could use `cut` as follows:

```
> wks = cut(everyday,breaks='week')
> head(wks)
[1] 2004-12-27 2004-12-27 2005-01-03 2005-01-03
[5] 2005-01-03 2005-01-03
53 Levels: 2004-12-27 2005-01-03 ... 2005-12-26
```

Note that the first observation had a date earlier than any of the dates in the `everyday` vector, since the first date was in middle of the week. By default, `cut` starts weeks on Mondays; to use Sundays instead, pass the `start.on.monday=FALSE` argument to `cut`.

Multiples of units can also be specified through the `breaks=` argument. For example, to create a factor based on the quarter of the year an observation is in, we could use `cut` as follows:

```
> qtrs = cut(everyday,"3 months",labels=paste('Q',1:4,sep=''))
> head(qtrs)
[1] Q1 Q1 Q1 Q1 Q1 Q1
Levels: Q1 Q2 Q3 Q4
```

5.6 Interactions

Sometimes it is useful to treat all combinations of several factors as if they were a single factor. In situations like these, the `interaction` function can be used. This function will take two or more factors, and create a new, unordered factor whose levels correspond to the combinations of the levels of the input factors. For example, consider the data frame `CO2`, with factors `Plant`, `Type`, and `Treatment`. Suppose we wish to create a new factor representing the interaction of `Plant` and `Type`:

```
> data(CO2)
> newfact = interaction(CO2$Plant,CO2$Type)
> nlevels(newfact)
[1] 24
```

The factor `Plant` has 12 levels, and `Type` has two, resulting in 24 levels in the new factor. However, some of these combinations never occur in the dataset. Thus, `interaction`'s default behavior is to include all possible combinations of its input factors. To retain only those combinations for which there were observations, the `drop=TRUE` argument can be passed to `interaction`:

```
> newfact1 = interaction(CO2$Plant,CO2$Type,drop=TRUE)
> nlevels(newfact1)
[1] 12
```

By default, `interaction` forms levels for the new factor by joining the levels of its component factors with a period (`.`). This can be overridden with the `sep=` argument.