**Factors in R**

Conceptually, factors are variables in R which take on a limited number of different values; such variables are often refered 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.

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 through the levels command.

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= argument is specified. See Section  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

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.

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:

> 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

Indexing the return value from the levels function is the most reliable way to convert numeric factors to their original numeric values.

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

1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1

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 simply use another call to factor

> table(factor(lets[1:5]))

a k q s z

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 inludes 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:

> l1 = factor(sample(letters,size=10,replace=TRUE))

> l2 = factor(sample(letters,size=10,replace=TRUE))

> l1

 [1] o b i v q n q w e z

Levels: b e i n o q v w z

> l2

 [1] b a s b l r g m z o

Levels: a b g l m o r s z

> l12 = factor(c(levels(l1)[l1],levels(l2)[l2]))

> l12

 [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

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 breakpoint. 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 data set, 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 make nicer default labels, 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] (120,140] (120,140] (120,140]

 [8] (120,140] (120,140] (140,160] (140,160] (140,160] (140,160] (140,160]

[15] (160,180]

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

As mentioned in Section , 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 an properly ordered factor.

For more details on formatting dates, see Section

Sometimes more flexibility can be acheived 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 2005-01-03 2005-01-03

53 Levels: 2004-12-27 2005-01-03 2005-01-10 2005-01-17 ... 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, 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