Factors

Chapter 5

Stats 20 Lec 2

Fall 2017

Contents

Acknowledgements: Vivian Lew and Juana Sanchez

Learning Objectives

After studying this chapter, you should be able to:

- Identify when to use factors.
- Create factors using `factor()`.
- Differentiate between character vectors and factors.
- Understand how R stores factors.
- Summarize a categorical variable using `table()`.
- Assign and reassign levels to a factor.
- Order the levels of a factor.

1 Basic Definitions

In experimental design (the process of designing experiments), a **factor** is an explanatory variable controlled by the experimenter. The different values the factor can take are called **levels**. For example, if we are designing an experiment to understand differences in efficacy between several headache medications, the medication is a factor, and the types of medication (e.g., acetaminophen, ibuprofen, naproxen, etc.) are the levels of the factor. More generally, we can think of categorical variables as synonymous with factors, where the categories are the levels.

The levels (categories) of a factor are sometimes represented (coded) as numbers, often to denote an ordering to the levels. For example, the Saffir-Simpson hurricane wind scale (SSHWS) classifies hurricanes into five categories, labeled Category 1, Category 2, etc., based on the maximum sustained wind speed of the hurricane. If we had data on hurricanes and input the category classifications as a numeric vector, R would not recognize that the vector represents categorical data.

We typically analyze categorical variables and numerical variables using different methods. For example, the mean classification for a sample of hurricanes in a given year would not make much sense. Instead, we might be interested in the relative frequencies of each classification.

Factors in R are an alternative way to store character vectors, particularly when the vector represents categories (levels) from a categorical variable (factor). The **`factor()`** and **`as.factor()`** functions can be used to create or coerce a vector into a factor.

As an example, suppose we have five subjects who are assigned into control or treatment groups. We can create a factor of the group variable:

```r
group <- c("control","treatment","control","treatment","treatment")
group # This is a character vector
```

```
## [1] "control"   "treatment" "control"   "treatment" "treatment"
```

```r
# Convert the group vector into a factor and overwrite the original vector by the factor.
group <- factor(group)
group
```

```
## [1] control   treatment control   treatment treatment
## Levels: control treatment
```

Note: The values of the factor vector are not in quotation marks. This highlights the fact that the vector does not contain character values.

Note: Because factors represent categorical data, we cannot apply the usual arithmetic operations on them, even though the values of factors are stored as integers. Attempting to apply numeric operations to factors will cause R to throw a warning and produce a vector of `NA` values.

```
group + 1
```

```
## Warning in Ops.factor(group, 1): '+' not meaningful for factors
```

```
## [1] NA NA NA NA NA
```

2 Working with Levels

2.1 The `levels()` Function

Because the values of a factor are limited to just the levels, there are often many repeated values. R more efficiently stores factors than character vectors with repeated values by internally storing and coding the levels of a factor as integers.

```
typeof(group) # Internal storage type of the factor vector
```

```
## [1] "integer"
```

```
as.integer(group) # How the levels of group are coded/stored in R
```

```
## [1] 1 2 1 2 2
```

The labels for the levels of a factor are only stored once, rather than being repeated. The **`levels()`** function accesses the levels attribute of a factor vector. The levels themselves are characters. The integer codes are indices of the levels vector.

```
levels(group)
```

```
## [1] "control"   "treatment"
```

```
levels(group)[as.integer(group)]
```

```
## [1] "control"   "treatment" "control"   "treatment" "treatment"
```

The `levels()` function can also be used to change the factor labels by using the assignment `<-` operator. For example, we can change the `"control" label to`“placebo”‘:

```
levels(group)[1] <- "placebo"
group
```

```
## [1] placebo   treatment placebo   treatment treatment
## Levels: placebo treatment
```

The **`nlevels()`** function returns the number of levels in the factor. The **`table()`** function will output a frequency table that summarizes the factor.

```
nlevels(group)
```

```
## [1] 2
```

```
table(group)
```

```
## group
##   placebo treatment
##         2         3
```

Caution: Changing an element of a factor to a new value will *not* change or add the factor label. If the new value is not already a level, R will replace the value by an NA and throw a warning.

```
group[5] <- "control" # Change the value from placebo to control (Warning!)
```

```
## Warning in `[<-.factor`(`*tmp*`, 5, value = "control"): invalid factor
## level, NA generated
```

```
group
```

```
## [1] placebo   treatment placebo   treatment <NA>
## Levels: placebo treatment
```

```
group[5] <- "placebo" # Change the value to placebo (No warning)
group
```

```
## [1] placebo   treatment placebo   treatment placebo
## Levels: placebo treatment
```

2.2 The `levels` Argument

The `levels` argument in the `factor()` function can be used to specify all possible levels of a factor, even if some are not observed in the data itself.

```
# Sample hurricane category data
hurricanes <- factor(c(3,1,2,5,3,3,5),levels=c(1,2,3,4,5))
hurricanes
```

```
## [1] 3 1 2 5 3 3 5
## Levels: 1 2 3 4 5
```

This can also be done by adding an element to the levels attribute through the `levels()` function.

```
# Sample self-identified gender data
gender <- factor(c("M","F","F","M","M"))
levels(gender) # Currently 2 levels
```

```
## [1] "F" "M"
```

```
levels(gender)[3] <- "X"
levels(gender) # Now has 3 levels
```

```
## [1] "F" "M" "X"
```

```
gender
```

```
## [1] M F F M M
## Levels: F M X
```

3 Extracting Values from Factors

Because a factor is a special type of vector, we can still use square brackets to extract values. However, extracting a subset of values from a factor will retain the levels attribute of the original factor, even if the subset of values does not contain all the levels.

```
hurricanes[1:3] # Only contains 1, 2, 3
```

```
## [1] 3 1 2
## Levels: 1 2 3 4 5
```

To remove the unobserved levels, we could invoke the `factor()` function again to reset the levels attribute.

```
factor(hurricanes[1:3])
```

```
## [1] 3 1 2
## Levels: 1 2 3
```

A more direct way to remove levels when subsetting values is to use the argument `drop=TRUE` in the square brackets.

```
hurricanes[1:3,drop=TRUE]
```

```
## [1] 3 1 2
## Levels: 1 2 3
```

4 Ordered Levels

Categorical variables which have a natural ordering to the categories (like hurricane categories or coffee cup sizes) are called **ordinal** variables. Ones which do not have a natural ordering (like gender or eye color) are called **nominal** variables.

By default, the `factor()` function will order the character levels in alphabetical (lexicographic) order and numeric levels in numerical (increasing) order. Lowercase will be ordered before their uppercase versions (so `a < A`).

For example, if we had data consisting of the names of the months, the natural ordering in the months would not be preserved. We will illustrate this with the built-in vector in base R called **`month.name`** that contains the names of the months.

```
month.name # Built-in character vector of the month names
```

```
##  [1] "January"   "February"  "March"     "April"     "May"
##  [6] "June"      "July"      "August"    "September" "October"
## [11] "November"  "December"
```

```
# Create a vector of month names for each day of the year
month.day <- rep(month.name,c(31,28,31,30,31,30,31,31,30,31,30,31))
f.month.day <- factor(month.day) # Convert into a factor
levels(f.month.day)
```

```
##  [1] "April"     "August"    "December"  "February"  "January"
##  [6] "July"      "June"      "March"     "May"       "November"
## [11] "October"   "September"
```

To specify the ordering of the levels, we can input the levels in the correct order in the `levels` argument of the `factor()` function and also set the argument `ordered` to be `TRUE`.

```
f.month.day <- factor(month.day,levels=month.name,ordered=TRUE)
f.month.day[1:10]
```

```
##  [1] January January January January January January January January
##  [9] January January
## 12 Levels: January < February < March < April < May < June < ... < December
```

```
levels(f.month.day)
```

```
##  [1] "January"   "February"  "March"     "April"     "May"
##  [6] "June"      "July"      "August"    "September" "October"
## [11] "November"  "December"
```