Factors

Chapter 6

Stats 20: Introduction to Statistical Programming with R

UCLA

Contents

Learning Objectives **2**

1 Basic Definitions **2**

2 Working with Levels **3**
2.1 The `levels()` Function 3
2.2 The `levels` Argument 4

3 Extracting Values from Factors **4**

4 Ordered Levels **5**

5 Operations on Subsets of Data **6**
5.1 The `tapply()` Function 6

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:

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

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

```
# 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)[group]
```

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

Notice that using the factor `group` is interpreted as the stored integer vector `as.integer(group)` when used in an index.

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.

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

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

```r
group
```

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

```r
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.

```r
# 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.

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

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

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

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

```r
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.

```r
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"       "June"
 [7] "July"      "August"    "September" "October"   "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"   "July"
 [7] "June"      "March"     "May"       "November"  "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 January
[10] January
12 Levels: January < February < March < April < May < June < ... < December
```

```
levels(f_month_day)
```

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

5 Operations on Subsets of Data

5.1 The `tapply()` Function

Recall that subsetting and logical indexing allow us to extract subsets of an object based on a condition or criterion. A natural application is to extract subsets of an object based on the levels of a factor (i.e., the categories of a categorical variable).

The **`tapply()`** function is used to apply a function to subsets of a vector.

The syntax of `tapply()` is `tapply(X, INDEX, FUN, ..., simplify = TRUE)`, where the arguments are:

- `X`: A numeric or logical vector
- `INDEX`: A factor or list of factors that identifies the subsets. Non-factors will be coerced into factors.
- `FUN`: The function to be applied.
- `...`: Any optional arguments to be passed to the `FUN` function.
- `simplify`: Logical value that specifies whether to simplify the output to a matrix or array.

The `tapply()` function splits the values of the vector `X` into groups, each group corresponding to a level of the `INDEX` factor, then applies the function in `FUN` to each group.

As an example, we will consider the `hurricanes.RData` file, which has four objects `category`, `pressure`, `wind`, and `year`, containing measurements on 455 hurricanes that occurred between 2006 and 2011.

```
load("hurricanes.RData") # Load the objects in the hurricanes data
category[1:10] # The Saffir-Simpson classification
```

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

```
pressure[1:10] # Air pressure at the hurricane's center (in millibars)
```

```
 [1] 983 968 981 960 952 983 981 953 985 990
```

```
wind[1:10]     # Hurricane's maximum sustained wind speed (in knots)
```

```
 [1] 65 90 80 65 95 85 70 95 80 70
```

```
year[1:10]     # Year of hurricane
```

```
 [1] 2006 2008 2008 2010 2008 2009 2011 2008 2007 2007
```

Side Note: The `hurricanes.RData` data was extracted from the `storms` dataset in the `dplyr` package, which itself is a subset of the NOAA Atlantic hurricane database best track data (HURDAT2), http://www.nhc.noaa.gov/data/#hurdat.

Note that the corresponding entry of each object refers to the same hurricane. For example, the 5th hurricane in the data was a Category `category[5]` hurricane, with air pressure of `pressure[5]`, maximum windspeed of `wind[5]`, and occurred in the year `year[5]`.

Suppose we are interested in whether the mean air pressure at a hurricane's center is related to the category of the hurricane. The `tapply()` function can split the pressures based on the category and compute the mean of each subset.

```
# Compute the mean pressure, grouped by category
tapply(pressure, category, mean)
```

```
       1        2        3        4        5
979.3766 964.3333 954.7407 940.3220 924.3000
```

From the output, we see that the mean pressure at a hurricane's center is lower for higher category hurricanes.

Question: How would you find the mean maximum sustained wind speed in each year?

Suppose we want to know the mean pressure for each category/year combination. The `tapply()` function can also group values based on combinations of levels from multiple factors. When using multiple factors in the `INDEX` argument, the factors need to be put into a list.

```
# Compute the mean pressure for each category/year combination
tapply(pressure, list(category, year), mean)
```

```
   2006     2007     2008     2009     2010     2011
1 983.9 981.5217 979.8158 977.9524 977.1948 979.3000
2 969.5 973.6000 957.0385 967.5000 966.5862 964.5385
3 957.0 948.0000 955.4286 953.6667 955.0000 954.6923
4    NA 933.7143 945.3750 948.8000 938.5238 942.6667
5    NA 924.3000       NA       NA       NA       NA
```

Question: How would you find out how many observations are in each category (or combination of categories)?