Factors

R Data Objects

Gaston Sanchez

STAT 33B, Fall 2025

About

In these slides we talk about some basic concepts of R lists.

One of the nicest features about R is that it provides a data structure exclusively designed to handle categorical data: **factors**

The term **factor** as used in R for handling categorical variables, comes from the terminology used in *Analysis of Variance*, commonly referred to as ANOVA. In this statistical method, a categorical variable is commonly referred to as *factor* and its categories are known as *levels*.

Main functions

- `factor()`: to create a factor.
- `levels()`: provides access to the levels attribute of a factor.
- `nlevels()`: gives the number of levels (i.e. categories).
- `length()`: gives the number of elements.
- `is.factor()`: tests if an object is of class `"factor"`
- `as.factor()`: coerces an object into a factor
- `is.ordered()`: checks if an object is an *ordered* factor

Creating Factors

To create a factor in you use the function `factor()`, which takes a vector as input. The vector can be either numeric, character or logical.

Here's an example:

```r
# numeric vector
num_vector <- c(1, 2, 3, 1, 2, 3, 2)

# creating a factor from num_vector
first_factor <- factor(num_vector)
first_factor
```

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

Creating Factors

You can also obtain a factor from a character vector:

```
# string vector
str_vector <- c('a', 'b', 'c', 'b', 'c', 'a', 'c', 'b')
str_vector
```

```
[1] "a" "b" "c" "b" "c" "a" "c" "b"
```

```
# creating a factor from str_vector
second_factor <- factor(str_vector)
second_factor
```

```
[1] a b c b c a c b
Levels: a b c
```

Notice how `str_vector` and `second_factor` are displayed. The letters in the character vector are displayed with quotes, while the letters in the factor are printed without quotes.

How R treats factors?

Technically speaking (https://cran.r-project.org/manuals.html), factors are referred to as *compound objects*. Factors are currently implemented using an integer array to specify the actual levels and a second array of names that are mapped to the integers.

Essentially, a factor is internally stored using two arrays: one is an integer array containing the values of categories, the other array is the `levels` which has the names of categories which are mapped to the integers.

How R treats factors?

Under the hood, the way R stores factors is as vectors of integer values. One way to confirm this is using the function `storage.mode()`

```r
# storage of factor
storage.mode(first_factor)
```

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

This means that we can manipulate factors just like we manipulate vectors.

Manipulate factors as any other vector

```r
# first element
first_factor[1]

# third element
first_factor[3]

# second to fourth elements
first_factor[2:4]

# last element
first_factor[length(first_factor)]

# logical subsetting
first_factor[rep(c(TRUE, FALSE), length.out = 7)]
```

Manipulate factors as any other vector

If you have a factor with named elements, you can also specify the names of the elements within the brackets:

```r
names(first_factor) <- letters[1:length(first_factor)]
first_factor
```

```
a b c d e f g
1 2 3 1 2 3 2
Levels: 1 2 3
```

```r
first_factor[c('b', 'd', 'f')]
```

```
b d f
2 1 3
Levels: 1 2 3
```

So what makes a factor different from a vector?

It turns out that factors have an additional attribute that vectors don't: `levels`. And as you can expect, the class of a factor is indeed `"factor"` (not `"vector"`).

```r
# attributes of a factor
attributes(first_factor)
```

```
$levels
[1] "1" "2" "3"

$class
[1] "factor"

$names
[1] "a" "b" "c" "d" "e" "f" "g"
```

So what makes a factor different from a vector?

Another feature that makes factors so special is that their values (the levels) are mapped to a set of character values for displaying purposes.

This implies that factors provide a way to store character values very efficiently. Each unique character value is stored only once, and the data itself is stored as a vector of integers.

So what makes a factor different from a vector?

Let's compare the sizes of apparently the same type of data:

```
# species in two formats
iris_factor <- iris$Species
iris_string <- as.character(iris$Species)

# comparison of memory size
object.size(iris_factor)
```

```
1248 bytes
```

```
object.size(iris_string)
```

```
1432 bytes
```

Note that the size of the factor `iris_factor` is less than the character vector `iris_string`

So what makes a factor different from a vector?

Storing a factor as integers will usually be more efficient than storing a character vector, especially when factors are of considerable size.

Factors can be used to encode **ordinal** variables. Qualitative data can be classified into nominal and ordinal variables. Nominal variables could be easily handled with character vectors. A different case is when we have ordinal variables, like sizes `"small"`, `"medium"`, `"large"` or college years `"freshman"`, `"sophomore"`, `"junior"`, `"senior"`. In these cases we are still using names of categories, but they can be arranged in increasing or decreasing order.

Ordinal Factors

We can use a character vector to store the values. But a character vector does not allow us to store the ranking of categories. The solution in R comes via factors. We can use factors to define ordinal variables, like the following example:

```r
sizes <- factor(
  x = c('sm', 'md', 'lg', 'sm', 'md'),
  levels = c('sm', 'md', 'lg'),
  ordered = TRUE)

sizes
```

```
[1] sm md lg sm md
Levels: sm < md < lg
```

When to factor?

When do we want data into factors? There is no universal answer to this question.

The decision of whether to convert strings into factors is going to depend on various aspects.

For instance, the purpose of the analysis, or the type of variable that contains the strings.

Sometimes it will make sense to have a variable as `factor`, like *gender* (e.g. male, female) or *ethnicity* (e.g. Hispanic, African-American, Native-American).

In other cases it does not make much sense to create a factor from a variable containing addresses or telephone numbers or names of individuals.

Function `factor()`

Function factor()

The usage of the function `factor()` is:

```r
factor(x, levels, labels = levels, exclude = NA,
       ordered = is.ordered(x), nmax = NA)
```

where:

- `x` a vector of data
- `levels` an optional vector for the categories
- `labels` an optional character vector of labels for the levels
- `exclude` a vector of values to be excluded when forming the set of levels
- `ordered` logical value to indicate if the levels should be regarded as ordered
- `nmax` an upper bound on the number of levels

Function `factor()`

The main argument of `factor()` is the input vector `x`.

The next argument is `levels`, followed by `labels`, both of which are optional arguments.

Although you won't always be providing values for `levels` and `labels`, it is important to understand how R handles these arguments by default.

Argument `levels`

If `levels` is not provided (which is what happens in most cases), then R assigns the unique values in `x` as the category levels.

For example, consider our first numeric vector:

```r
num_vector <- c(1, 2, 3, 1, 2, 3, 2)

first_factor <- factor(num_vector)
first_factor
```

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

Argument `levels`

Now imagine we want to have `levels` 1, 2, 3, 4, and 5. This is how you can define the factor with an extended set of levels:

```r
# defining levels
one_factor <- factor(num_vector, levels = 1:5)
one_factor
```

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

Although the created factor only has values between 1 and 3, the `levels` range from 1 to 5. This can be useful if we plan to add elements whose values are not in the input vector `num_vector`.

Argument labels

Another very useful argument is `labels`, which allows you to provide a string vector for naming the `levels` in a different way from the values in `x`.

```
num_vector <- c(1, 2, 3, 1, 2, 3, 2)

# defining labels
num_word_vector <- factor(
  x = num_vector,
  labels = c("one", "two", "three"))

num_word_vector
```

```
[1] one   two   three one   two   three two
Levels: one two three
```

Setting/changing `levels`

If what you want is to specify the `levels` attribute, you must use the function `levels()` followed by the assignment operator `<-`.

Suppose that we want to change the levels of `first_factor` and express them in roman numerals. You can achieve this with:

```r
# copy of first factor
first_factor_copy <- first_factor
first_factor_copy
```

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

```r
# setting new levels
levels(first_factor_copy) <- c("I", "II", "III")
first_factor_copy
```

```
[1] I   II  III I   II  III II
Levels: I II III
```

Merging Levels

Sometimes we may need to **merge** or collapse two or more different levels into one single level. We can achieve this with `levels()` by assigning a new vector of levels containing repeated values for those categories that we wish to merge.

For example, say we want to combine categories `I` and `III` into a new level `I+III`. Here's how to do it:

```r
# copy of first factor
first_factor_merged <- first_factor_copy
first_factor_merged
```

```
[1] I   II  III I   II  III II
Levels: I II III
```

```r
# merging levels
levels(first_factor_merged) <- c("I+III", "II", "I+III")
first_factor_merged
```

```
[1] I+III II    I+III I+III II    I+III II
Levels: I+III II
```

Ordinal Factors

Ordinal Factors

By default, `factor()` creates a *nominal* categorical variable, not an ordinal. One way to check that you have a nominal factor is to use the function `is.ordered()`.

```r
num_vector <- c(1, 2, 3, 1, 2, 3, 2)
num_factor <- factor(num_vector)

# ordinal factor?
is.ordered(num_factor)
```

```
[1] FALSE
```

Ordinal Factors

If you want to specify an ordinal factor you must use the `ordered` argument of `factor()`.

```
# ordinal factor from numeric vector
ordinal_num <- factor(num_vector, ordered = TRUE)
ordinal_num
```

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

Notice that the levels of `ordinal_factor` are displayed with less-than symbols `'<'`.

Ordinal Factors

An alternative way to specify an ordinal variable is by using the function `ordered()` which is just a convenient wrapper for `factor(x, ..., ordered = TRUE)`:

```r
# ordinal factor with ordered()
ordered(num_vector)
```

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

```r
# same as using 'ordered' argument
factor(num_vector, ordered = TRUE)
```

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

A word of caution. Don't confuse the function `ordered()` with `order()`

Ordinal Factors

Sometimes you want to determine categories in a particular order.

For example, say we want to make an ordinal factor from `str_vector` such that its levels are `c < b < a`

```r
str_vector <- c('a', 'b', 'c', 'b', 'c', 'a', 'c', 'b')

# setting levels with specified order
factor(str_vector, levels = c("c", "b", "a"), ordered = TRUE)
```

```
[1] a b c b c a c b
Levels: c < b < a
```

```r
# equivalently
ordered(str_vector, levels = c("c", "b", "a"))
```

```
[1] a b c b c a c b
Levels: c < b < a
```