Factors - Releveling & Recoding

Author: Cole Brookson **Date:** 20 July 2022

Intro to Factors

Factors are an important but frustrating special data type in R. They can be thought of as "categorical" variables, in the sense that they can only take on a limited number of values. R uses a vector of integer values to store a factor, with a corresponding set of character values that are displayed while using the factor.

They are most commonly used to group numerical variables, and therefore commonly come into play when working with regression-related topics that require either categorical fixed effects or a random effect. An example might be the site at which some data were collected. In a study where there are only six sites, it might be the case that we need to use a *random effect* to control for variation between sites. In that scenario, it would be helpful to treat the variable **Site** as a factor in R.

We can make factors easily from any other type of variable. Take the following example dataset with three variables, one each of numeric, logical, and character types.

```
df = data.frame(
  num_ex = sample(c(1,2,3,4), replace = TRUE, 50),
  log_ex = sample(c(TRUE, FALSE), replace = TRUE, 50),
  chr_ex = sample(c("hehe", "haha"), replace = TRUE, 50)
)
str(df)
```

```
## 'data.frame':    50 obs. of  3 variables:
##  $ num_ex: num  2 1 2 4 2 2 3 2 3 2 ...
##  $ log_ex: logi  FALSE TRUE TRUE FALSE TRUE TRUE ...
##  $ chr_ex: chr  "haha" "hehe" "haha" "hehe" ...
```

We can make each of these into factors. **Note** that when working with factors in the context of columns in a dataframe (via the Tidyverse), it is most effective to make new columns replacing the old column with the `mutate()` function (LINK TO MAKING NEW COLUMNS).

We'll mutate each of these columns into factors:

```
library(tidyverse)

df_fac = df %>%
  dplyr::mutate(
    num_ex = as.factor(num_ex),
    log_ex = as.factor(log_ex),
    chr_ex = as.factor(chr_ex)
  )
str(df_fac)
```

```
## 'data.frame':    50 obs. of  3 variables:
##  $ num_ex: Factor w/ 4 levels "1","2","3","4": 2 1 2 4 2 2 3 2 3 2 ...
##  $ log_ex: Factor w/ 2 levels "FALSE","TRUE": 1 2 2 1 2 2 1 1 2 1 ...
##  $ chr_ex: Factor w/ 2 levels "haha","hehe": 1 2 1 2 1 1 1 1 1 1 ...
```

We can now see we only have columns that have the type `factor`. If we wanted to keep a version of the column that holds its original type, we could make new columns, as such:

```
df_both = df %>%
  dplyr::mutate(
    num_ex_fac = as.factor(num_ex),
    log_ex_fac = as.factor(log_ex),
    chr_ex_fac = as.factor(chr_ex)
  )
str(df_both)
```

```
## 'data.frame':    50 obs. of  6 variables:
##  $ num_ex    : num  2 1 2 4 2 2 3 2 3 2 ...
##  $ log_ex    : logi  FALSE TRUE TRUE FALSE TRUE TRUE ...
##  $ chr_ex    : chr  "haha" "hehe" "haha" "hehe" ...
##  $ num_ex_fac: Factor w/ 4 levels "1","2","3","4": 2 1 2 4 2 2 3 2 3 2 ...
##  $ log_ex_fac: Factor w/ 2 levels "FALSE","TRUE": 1 2 2 1 2 2 1 1 2 1 ...
##  $ chr_ex_fac: Factor w/ 2 levels "haha","hehe": 1 2 1 2 1 1 1 1 1 1 ...
```

There are now 6 columns in our dataframe as we wished.

Re-leveling Factors

When working with factors, we often find ourselves needing to "re-level" these variables. Since the order of factors matters, and there are often specific reasons we may want to re-name each occurence of one or more level in the factor, re-leveling is common.

Reordering/Releveling Factors

If we have a variable that takes the form of a factor, we can change the order of that variable relatively easily. This is useful if we are, for example, plotting the factor and want the x-axis of our plot to appear in alphabetical order. Here is an example:

Say we have a dataframe:

```
group = sample(c("First", "Second", "Third", "Fourth"), 500, replace = TRUE)
vals = rnorm(500, mean = 10, sd = 2)
df = data.frame(group = as.factor(group), vals)
```

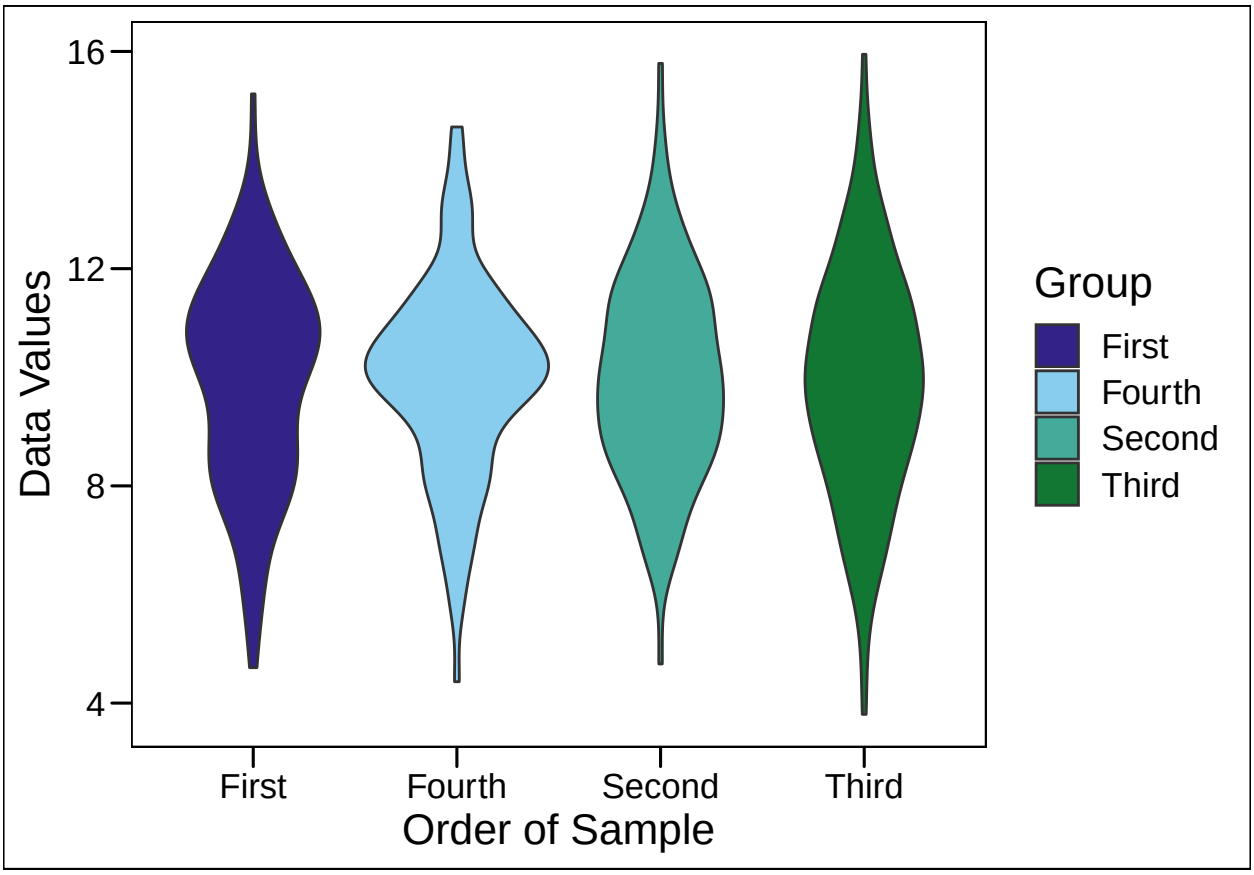
We can look at the levels of our factor (the `group` variable) with the function `levels()` from base R:

```
levels(df$group)
```

```
## [1] "First"  "Fourth" "Second" "Third"
```

We can see here that the order is not in numerically logical order. Does that matter? Well if we plot those data, we can see it looks like this:

```
library(ggthemes)
ggplot(data = df) +
  geom_violin(aes(x = group, y = vals, fill = group)) +
  theme_base() +
  labs(x = "Order of Sample", y = "Data Values") +
  scale_fill_manual("Group", values = c("#332288", "#88CCEE",
                                        "#44AA99", "#117733"))
```



Now this is a nice plot, but it doesn't really make sense that on the x-axis, the order doesn't correspond to a logical numeric order. This is easily remedied with the `forcats` package from the Tidyverse. We can *re-order*, or in R-speak *re-level* the factor, and then R will know what order we want when we use it for any further plotting or other activities.

```
library(forcats)
df$group = forcats::fct_relevel(
  df$group, levels = c("First", "Second", "Third", "Fourth")
  )
```

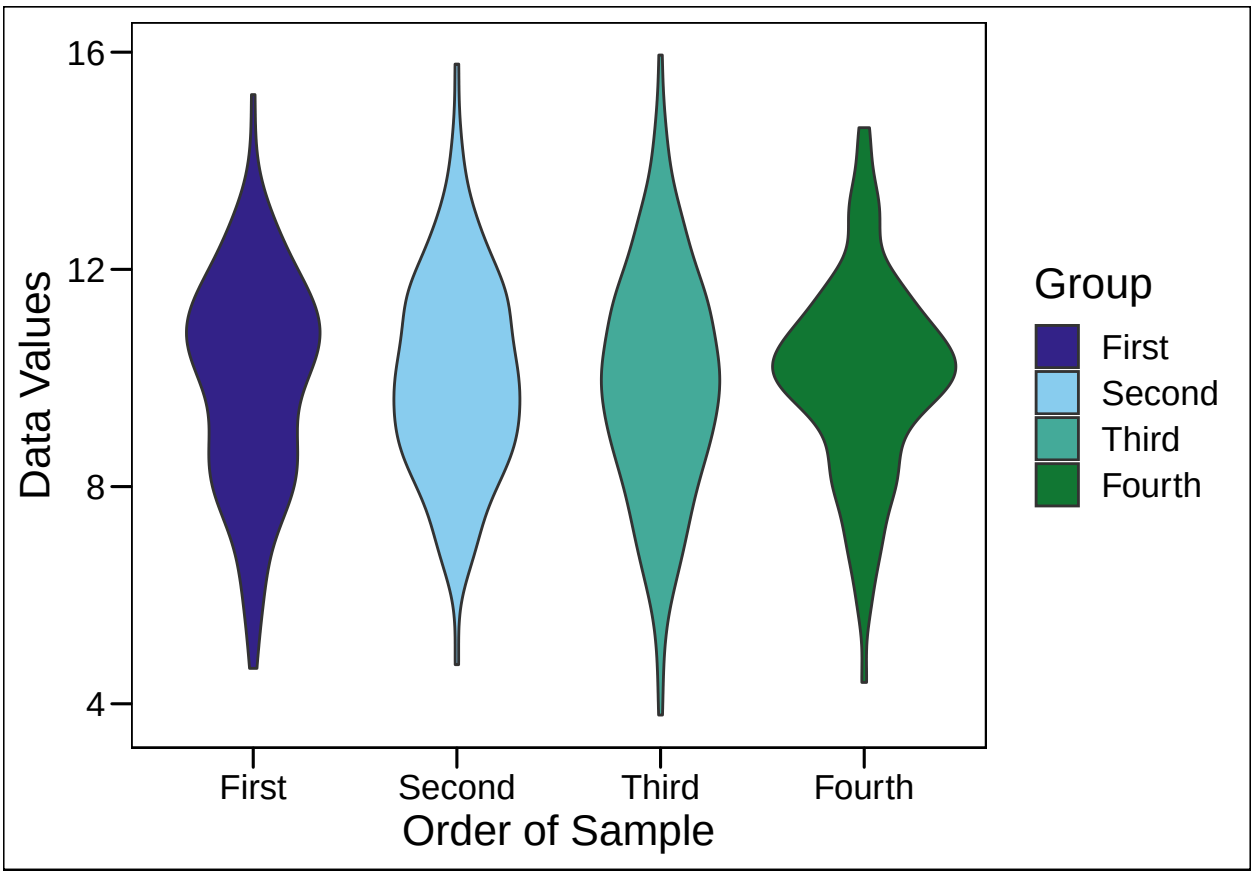
```
## Warning: Outer names are only allowed for unnamed scalar atomic inputs
```

```
# now we can use the base R function `levels()` to check if the order is what
# we want it to be
levels(df$group)
```

```
## [1] "First"  "Second" "Third"  "Fourth"
```

And we can see that the order is what we want. So that if we were to make the same plot again...

```
ggplot(data = df) +
  geom_violin(aes(x = group, y = vals, fill = group)) +
  theme_base() +
  labs(x = "Order of Sample", y = "Data Values") +
  scale_fill_manual("Group", values = c("#332288", "#88CCEE",
                                        "#44AA99", "#117733"))
```



The x-axis and the legend are now in a logical numeric order.

Recoding Factors

Another common operation is deciding that we actually don't want a level of our factor to read the same way anymore, that is we want to replace it with something else. We may have a dataframe, with a factor column called `group`:

```
group = factor(sample(c("4", "5", "Six", "7"), size = 500, replace = TRUE))
vals = rnorm(500, mean = 2, sd = 2.5)
df = data.frame(group, vals)
# let's look at the levels of our `group` column
levels(df$group)
```

```
## [1] "4"   "5"   "7"   "Six"
```

We can see that for some reason here we have the "6" replaced by the written out version of the word. This will make the order not logical, and may cause problems if for some reason we wanted this variable to be numeric at some point (**Remember, you can make a character value into a numeric value**).

To repalce this value in our column, we can use the `recode()` function from dplyr. It is most consistent to do this within a `mutate()`.

```
df = df %>%
  dplyr::mutate(
    group = recode(
      # first argument is the data object
      group,
      # next argument is the level we want to replace
      "Six" = "6")
  )
```

We can check if it worked with:

```
levels(df$group)
```

```
## [1] "4" "5" "7" "6"
```

So this worked! However the order is still off because these are characters, so R can't tell that they're numbers that can be ordered. We could reorder the factor with our approach we took above:

```
df = df %>%
  dplyr::mutate(
    group = forcats::fct_relevel(group, c("4", "5", "6", "7"))
  )
levels(df$group)
```

```
## [1] "4" "5" "6" "7"
```

The order is now correct.

Re-leveling According to a Function

Content Comimg Soon