Lesson 06 - Controlling your factors

Last Updated 02-03-2021

Introduction

In this lesson we will discuss ways to organize and deal with categorical data, also known as factor data types.

Student Learning Objectives

After completing this lesson students will be able to

- Convert a numeric variable to a factor variable.
- Apply and change labels to factor
- Understand and control the ordering of the factor.
- Combine multiple levels of a factor variable into one level

Preparation

Prior to this lesson learners should

- Download the [06_factors_notes.Rmd] R markdown file and save into your `notes/Math130` folder.
  - Open this in RStudio and ensure the first code chunk will run without error.
- Install the `forcats` package (this was done as part of lesson 02).

We're going to use two data sets from the `openintro` package. The `email` data set contains information on emails received by a specific account for the first three months in 2012, and the `fastfood` data set describes nutrition amounts in 515 fast food items. More information can be seen about these data sets by viewing the help files using `?email` and `?fastfood`.

```
library(forcats)
email <- openintro::email
ff    <- openintro::fastfood
```

The goal of the `forcats` package is to provide a suite of useful tools that solve common problems with factors. Often in R there are multiple ways to accomplish the same task. Some examples in this lesson will show how to perform a certain task using base R functions, as well as functions from the `forcats` package.

What is a factor?

The term factor refers to a data type used to store categorical variables. The difference between a categorical variable and a continuous variable is that a categorical variable corresponds to a limited number of categories, while a continuous variable can correspond to an infinite number of values.

An example of a categorical variable is the `number` variable in the `email` data set. This variable contains data on whether there was no number, a small number (under 1 million), or a big number in the content of the email.

First we should confirm that R sees `number` as a factor.

```
class(email$number)
```

```
## [1] "factor"
```

We can use the `levels()` function to get to know factor variables.

```
levels(email$number)
```

```
## [1] "none"  "small" "big"
```

There are three levels: `none`, `small`, and `big`.

How many records are in each level? There are a few ways we can determine this, here are two:

Base R

```
table(email$number)
```

```
##
##  none small   big
##   549  2827   545
```

forcats

```
fct_count(email$number)
```

```
## # A tibble: 3 x 2
##   f         n
##   <fct> <int>
## 1 none    549
## 2 small  2827
## 3 big     545
```

Note that `fct_count` also gives you the number of records with that factor level. This is yet another way to generate a frequency table.

Convert a number to Factor

Sometimes data are entered into the computer using numeric codes such as 0 and 1. These codes stand for categories, such as "no" and "yes". Sometimes we want to analyze these binary variables in two ways:

- For statistical analyses, the data must be numeric 0/1.
- For many graphics, the data must be a factor, "no/yes".

Example: Is the email flagged as spam? The `spam` variable is recorded as an integer variable with values 0 and 1.

```
table(email$spam)
```

```
##
##    0    1
## 3554  367
```

```
class(email$spam)
```

```
## [1] "numeric"
```

We use the function `factor()` to convert the numeric variable `spam` to a factor, applying `labels` to convert 0 to "no" and 1 to "yes".

```
email$spam_fac <- factor(email$spam, labels=c("no", "yes"))
```

The ordering of the `labels` argument *must* be in the same order (left to right) as the factor levels themselves. Look back at the order of columns in the `table` - it goes 0 then 1. Thus our labels need to go "no" then "yes".

Always confirm your recode

Here we confirm that the new variable was created corectly by creating a two-way contingency table by calling the `table(old variable, new variable)` function on both the old and new variables.

```
table(email$spam, email$spam_fac, useNA="always")
```

```
##
##          no  yes <NA>
##   0    3554    0    0
##   1       0  367    0
##   <NA>    0    0    0
```

Here we see that all the 0's were recoded to 'no's, and all the 1's recoded to "yes"'s, and there are no new missing values. Success!

Factor ordering

Let's revisit the variable `number`, that contains the size of the number in the email.

```
table(email$number)
```

```
##
##  none small   big
##   549  2827   545
```

Specifically the ordering from left to right of the factors. This is ordinal data, in that `none` is inherently "smaller" than `small`, which is smaller than `big`. Though R correctly ordered these data, it may default to alphabetical order in other cases, so beware! You may need to correct the ordering for other data sets.

Let's see a few ways of how to control the ordering.

Manually specified

We need to take control of these factors! We can do that by re-factoring the existing factor variable, but this time specifying the `levels` of the factor (since it already has labels). Say we decide to reverse the order so we go from big to small to none, in decreasing order of size.

Base R

```
factor(email$number, levels=c("big", "small", "none")) %>% table()
```

```
## .
##   big small  none
##   545  2827   549
```

forcats

```
email$number %>% fct_relevel("big", "small", "none") %>% table()
```

```
## .
##   big small  none
##   545  2827   549
```

In each of these examples, I pipe (`%>%`) the `table()` function at the end so I can see the results of the reordering. This helps me visually confirm that the code used changed the factor levels to be left to right in decreasing content size order. This will be important for graphing.

> Since I did not use the assignment operator (`<-`) here, these changes were not made to the variable in the `email` data set. The examples below demonstrate making an adjustmet to a factor variable and saving that adjustment as a new variable in the data set.

Factor (re)naming

Sometimes factors come to us in names we don't prefer. We want them to say something else.

Base R The easiest way here is to re-factor the variable and apply new `labels`.

```
email$my_new_number <- factor(email$number, labels=c( "None", "<1M","1M+"))
```

Ok, but did this work? Trust, but verify.

```
table(email$number, email$my_new_number, useNA="always")
```

```
##
##         None  <1M  1M+ <NA>
##   none   549    0    0    0
##   small    0 2827    0    0
##   big      0    0  545    0
##   <NA>     0    0    0    0
```

The "big" factor is now labeled "1M+", "none" is named "None", and "small" is "<1M".

forcats: use the `fct_recode("NEW" = "old")` function here.

```
email$my_forcats_number <- fct_recode(email$number, "BIG" = "big", "NONE" = "none", "SMALL" = "small")
table(email$number, email$my_forcats_number, useNA="always")
```

```
##
##         NONE SMALL  BIG <NA>
##   none   549     0    0    0
##   small    0  2827    0    0
##   big      0     0  545    0
##   <NA>     0     0    0    0
```

Collapsing factor levels.

For analysis purposes, sometimes you want to work with a smaller number of factor variables. Let's look at the restaurants that are included in the `fastfood` data set.

```
table(ff$restaurant)
```

```
##
##       Arbys Burger King Chick Fil-A Dairy Queen   Mcdonalds       Sonic
##          55          70          27          42          57          53
##      Subway   Taco Bell
##          96         115
```

Let's combine all the sandwich, and burger joints together. I am going to save this new variable as `restaurant_new`.

The syntax for the `fct_collapse` function is `new level = "old level"`, where the "old level" is in quotes. As always, it is good practice to create a two way table to make sure the code typed does what we expected it to do.

```
ff$restaurant_new <- fct_collapse(ff$restaurant,
                                  BurgerJoint = c("Burger King", "Mcdonalds", "Sonic"),
```

```
                                    Sammich = c("Arbys", "Subway"))

table(ff$restaurant, ff$restaurant_new, useNA="always")
```

```
##
##                Sammich BurgerJoint Chick Fil-A Dairy Queen Taco Bell <NA>
##   Arbys             55           0           0           0         0    0
##   Burger King        0          70           0           0         0    0
##   Chick Fil-A        0           0          27           0         0    0
##   Dairy Queen        0           0           0          42         0    0
##   Mcdonalds          0          57           0           0         0    0
##   Sonic              0          53           0           0         0    0
##   Subway            96           0           0           0         0    0
##   Taco Bell          0           0           0           0       115    0
##   <NA>               0           0           0           0         0    0
```

Let's take this one step further and shorten the factor names and rename the levels with spaces (like `Taco Bell`).

```
ff$restaurant_shortname <- fct_recode(ff$restaurant_new,
                                      "Ckn" = "Chick Fil-A",
                                      "DQ" = "Dairy Queen",
                                      "TB" = "Taco Bell",
                                      "Samm" = 'Sammich',
                                      "Burg"= "BurgerJoint")

table(ff$restaurant, ff$restaurant_shortname, useNA="always")
```

```
##
##                Samm Burg Ckn  DQ  TB <NA>
##   Arbys          55    0   0   0   0    0
##   Burger King     0   70   0   0   0    0
##   Chick Fil-A     0    0  27   0   0    0
##   Dairy Queen     0    0   0  42   0    0
##   Mcdonalds       0   57   0   0   0    0
##   Sonic           0   53   0   0   0    0
##   Subway         96    0   0   0   0    0
##   Taco Bell       0    0   0   0 115    0
##   <NA>            0    0   0   0   0    0
```

Let's chain this all together.

Sporadically throughout this lesson, and at the end of the last lesson i've used the pipe `%>%` operator as somewhat of a shortcut. This amazing piece of code lets us string together commands. Let's see how to accomplish both changes to the restaurant variable in one single step.

```
ff$restaurant %>% fct_collapse(BurgerJoint = c("Burger King", "Mcdonalds", "Sonic"),
                               Sammich     = c("Arbys", "Subway")) %>%
                  fct_recode("Ckn"  = "Chick Fil-A",
                             "DQ"   = "Dairy Queen",
                             "TB"   = "Taco Bell",
                             "Samm" = 'Sammich',
                             "Burg "= "BurgerJoint") %>%
                  table()
```

```
## .
```

```
##  Samm Burg    Ckn    DQ    TB
##   151   180    27    42   115
```

Few things to note when chaining commands together:

1. the first argument (the data or variable) is not included.
2. the pipe itself must be at the end of a "sentence"

Don't worry if this doesn't make immediate sense right now. We'll talk more on chaining commands together in a later lesson. There is no harm in accomplishing a task in more than one step.

Removing factor levels.

Sometimes, you don't even want to consider certain levels. This often occurs in survey data where the respondent provides an answer of "Refuse to answer" or the data is coded as the word "missing". The word "missing' is fundamentally different than the `NA` code for a missing value.

For demonstration purposes, let's get rid of the data from DQ. Who eats something other than ice cream at that place anyhow?

The most straight forward way to set a varible missing based on it's value is to use a logical statement to choose what rows the variable is set to missing.

```
ff$restaurant_shortname[ff$restaurant_shortname == "DQ"] <- NA
table(ff$restaurant_shortname, useNA="always")
```

```
##
## Samm Burg  Ckn   DQ   TB <NA>
##  151  180   27    0  115   42
```

This table shows that there are 42 records missing `restaraunt_shortname`, and 0 records with the level `DQ`. Unfortunately, the level itself still is there. R does not assume just because there are no records with that level, that the named level itself should be removed. We can use the function `fct_drop` to drop the levels with no records.

```
fct_drop(ff$restaurant_shortname) %>% table()
```

```
## .
## Samm Burg  Ckn   TB
##  151  180   27  115
```

If we knew ahead of time we wanted to drop DQ, we could have set that level to `NULL` when we did the initial recode.

```
ff$restaurant %>% fct_collapse(BurgerJoint = c("Burger King", "Mcdonalds", "Sonic"),
                               Sammich = c("Arbys", "Subway")) %>%
                  fct_recode("Ckn"  = "Chick Fil-A",
                             NULL   = "Dairy Queen",
                             "TB"   = "Taco Bell",
                             "Samm" = 'Sammich',
                             "Burg "= "BurgerJoint") %>%
                  table()
```

```
## .
##  Samm Burg    Ckn    TB
##   151   180    27   115
```

#

Go Back to Week 2