# Factors

*I find factors, levels and labels one of the hardest things to remember and be confident that they are coded correctly. I have added my notes to the R4DS RMarkdown file on factors to hopefully help me to understand it better. Maybe it will help you too*, **Anthony**.

## Introduction

In R, factors are used to work with categorical variables. Categorical variables have a fixed and known set of possible values.

* They are also useful when you want to display character vectors in a non-alphabetical order. For example with ggplot2 barplots.

Historically, factors were much easier to work with than characters. As a result, many of the functions in base R automatically convert characters to factors. This means that factors often crop up in places where they’re not actually helpful. Fortunately, you don’t need to worry about that in the tidyverse, and can focus on situations where factors are genuinely useful.

### Prerequisites

To work with factors, we’ll use the forcats package, which provides tools for dealing with **cat**egorical variables (and it’s an anagram of factors!). It provides a wide range of helpers for working with factors. forcats is not part of the core tidyverse, so we need to load it explicitly as so.

library(tidyverse)

## Warning: package 'ggplot2' was built under R version 3.5.3

## Warning: package 'tibble' was built under R version 3.5.3

## Warning: package 'tidyr' was built under R version 3.5.3

## Warning: package 'purrr' was built under R version 3.5.3

library(forcats)

### Learning more

If you want to learn more about factors, I recommend reading Amelia McNamara and Nicholas Horton’s paper, [*Wrangling categorical data in R*](https://peerj.com/preprints/3163/). This paper lays out some of the history discussed in [*stringsAsFactors: An unauthorized biography*](http://simplystatistics.org/2015/07/24/stringsasfactors-an-unauthorized-biography/) and [*stringsAsFactors = <sigh>*](http://notstatschat.tumblr.com/post/124987394001/stringsasfactors-sigh), and compares the tidy approaches to categorical data outlined in this book with base R methods. A early version of the paper help motivate and scope the forcats package; thanks Amelia & Nick!

## Creating factors

Imagine that you have a variable that records month:

x1 <- c("Dec", "Apr", "Jan", "Mar")

Using a string to record this variable has two problems:

1. There are only twelve possible months, and there’s nothing saving you from typos:

* x2 <- c("Dec", "Apr", "Jam", "Mar")

1. It doesn’t sort in a useful way:

* sort(x1)
* ## [1] "Apr" "Dec" "Jan" "Mar"

You can fix both of these problems with a factor. To create a factor you must start by creating a list of the valid levels:

month\_levels <- c(  
 "Jan", "Feb", "Mar", "Apr", "May", "Jun",  
 "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"  
)

Now you can create a factor:

y1 <- factor(x1, levels = month\_levels)  
y1

## [1] Dec Apr Jan Mar  
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

sort(y1)

## [1] Jan Mar Apr Dec  
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

And any values not in the set will be silently converted to NA:

y2 <- factor(x2, levels = month\_levels)  
y2

## [1] Dec Apr <NA> Mar   
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

If you want a warning, you can use readr::parse\_factor():

y2 <- parse\_factor(x2, levels = month\_levels)

## Warning: 1 parsing failure.  
## row col expected actual  
## 3 -- value in level set Jam

If you omit the levels, they’ll be taken from the data in alphabetical order:

factor(x1)

## [1] Dec Apr Jan Mar  
## Levels: Apr Dec Jan Mar

Sometimes you’d prefer that the order of the levels match the order of the first appearance in the data. *This is where I get stuck.* You can do that when creating the factor by setting levels to unique(x):

f1 <- factor(x1, levels = unique(x1))  
f1

## [1] Dec Apr Jan Mar  
## Levels: Dec Apr Jan Mar

**or** after the fact with fct\_inorder():

f2 <- x1 %>% factor() %>% fct\_inorder()  
f2

## [1] Dec Apr Jan Mar  
## Levels: Dec Apr Jan Mar

If you ever need to access the set of valid levels directly, you can do so with levels():

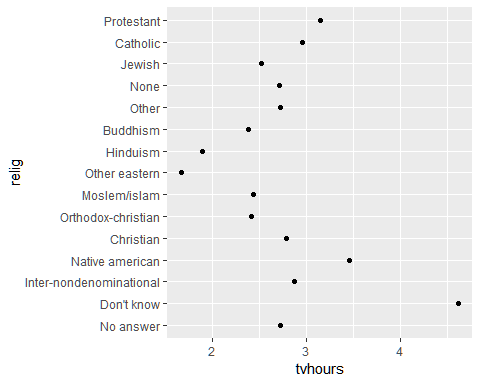
levels(f2)

## [1] "Dec" "Apr" "Jan" "Mar"

## Modifying factor order

It’s often useful to change the order of the factor levels in a visualisation. For example, imagine you want to explore the average number of hours spent watching TV per day across religions:

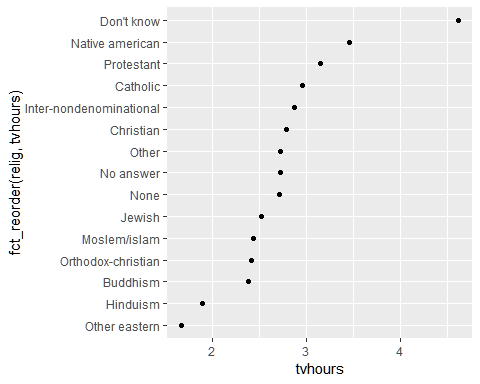
relig\_summary <- gss\_cat %>%  
 group\_by(relig) %>%  
 summarise(  
 age = mean(age, na.rm = TRUE),  
 tvhours = mean(tvhours, na.rm = TRUE),  
 n = n()  
 )  
  
ggplot(relig\_summary, aes(tvhours, relig)) + geom\_point()



It is difficult to interpret this plot because there’s no overall pattern. We can improve it by reordering the levels of relig using fct\_reorder(). fct\_reorder() takes three arguments:

* f, the factor whose levels you want to modify.
* x, a numeric vector that you want to use to reorder the levels.
* Optionally, fun, a function that’s used if there are multiple values of x for each value of f. The default value is median.

ggplot(relig\_summary, aes(tvhours, fct\_reorder(relig, tvhours))) +  
 geom\_point()



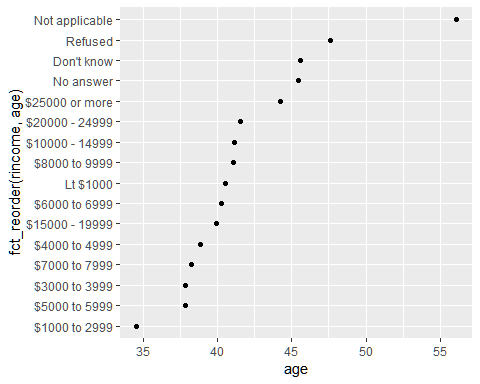
Reordering religion makes it much easier to see that people in the “Don’t know” category watch much more TV, and Hinduism & Other Eastern religions watch much less.

As you start making more complicated transformations, I’d recommend moving them out of aes() and into a separate mutate() step. For example, you could rewrite the plot above as:

relig\_summary %>%  
 mutate(relig = fct\_reorder(relig, tvhours)) %>%  
 ggplot(aes(tvhours, relig)) +  
 geom\_point()

What if we create a similar plot looking at how average age varies across reported income level?

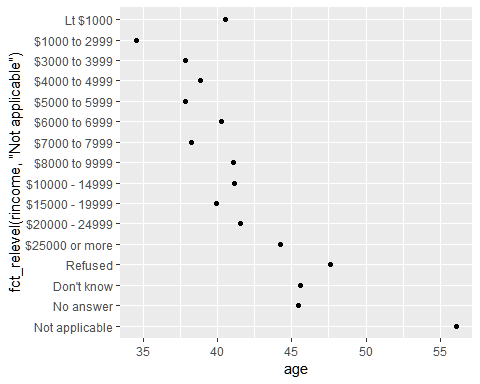
rincome\_summary <- gss\_cat %>%  
 group\_by(rincome) %>%  
 summarise(  
 age = mean(age, na.rm = TRUE),  
 tvhours = mean(tvhours, na.rm = TRUE),  
 n = n()  
 )  
  
ggplot(rincome\_summary, aes(age, fct\_reorder(rincome, age))) + geom\_point()



Here, arbitrarily reordering the levels isn’t a good idea! That’s because rincome already has a principled order that we shouldn’t mess with. Reserve fct\_reorder() for factors whose levels are arbitrarily ordered.

However, it does make sense to pull “Not applicable” to the front with the other special levels. You can use fct\_relevel(). It takes a factor, f, and then any number of levels that you want to move to the front of the line.

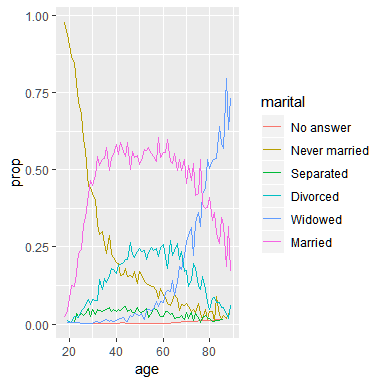
ggplot(rincome\_summary, aes(age, fct\_relevel(rincome, "Not applicable"))) +  
 geom\_point()



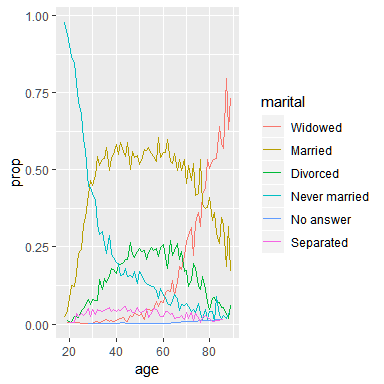
Why do you think the average age for “Not applicable” is so high?

Another type of reordering is useful when you are colouring the lines on a plot. fct\_reorder2() reorders the factor by the y values associated with the largest x values. This makes the plot easier to read because the line colours line up with the legend.

by\_age <- gss\_cat %>%  
 filter(!is.na(age)) %>%  
 count(age, marital) %>%  
 group\_by(age) %>%  
 mutate(prop = n / sum(n))  
  
ggplot(by\_age, aes(age, prop, colour = marital)) +  
 geom\_line(na.rm = TRUE)

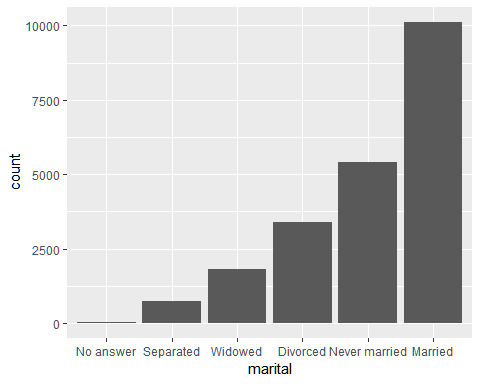


ggplot(by\_age, aes(age, prop, colour = fct\_reorder2(marital, age, prop))) +  
 geom\_line() +  
 labs(colour = "marital")



Finally, for bar plots, you can use fct\_infreq() to order levels in increasing frequency: this is the simplest type of reordering because it doesn’t need any extra variables. You may want to combine with fct\_rev().

gss\_cat %>%  
 mutate(marital = marital %>% fct\_infreq() %>% fct\_rev()) %>%  
 ggplot(aes(marital)) +  
 geom\_bar()



## General Social Survey

For the rest of this chapter, we’re going to focus on forcats::gss\_cat. It’s a sample of data from the [General Social Survey](http://gss.norc.org), which is a long-running US survey conducted by the independent research organization NORC at the University of Chicago. The survey has thousands of questions, so in gss\_cat I’ve selected a handful that will illustrate some common challenges you’ll encounter when working with factors.

gss\_cat

## # A tibble: 21,483 x 9  
## year marital age race rincome partyid relig denom tvhours  
## <int> <fct> <int> <fct> <fct> <fct> <fct> <fct> <int>  
## 1 2000 Never ma~ 26 White $8000 to~ Ind,near ~ Protes~ Southe~ 12  
## 2 2000 Divorced 48 White $8000 to~ Not str r~ Protes~ Baptis~ NA  
## 3 2000 Widowed 67 White Not appl~ Independe~ Protes~ No den~ 2  
## 4 2000 Never ma~ 39 White Not appl~ Ind,near ~ Orthod~ Not ap~ 4  
## 5 2000 Divorced 25 White Not appl~ Not str d~ None Not ap~ 1  
## 6 2000 Married 25 White $20000 -~ Strong de~ Protes~ Southe~ NA  
## 7 2000 Never ma~ 36 White $25000 o~ Not str r~ Christ~ Not ap~ 3  
## 8 2000 Divorced 44 White $7000 to~ Ind,near ~ Protes~ Luther~ NA  
## 9 2000 Married 44 White $25000 o~ Not str d~ Protes~ Other 0  
## 10 2000 Married 47 White $25000 o~ Strong re~ Protes~ Southe~ 3  
## # ... with 21,473 more rows

(Remember, since this dataset is provided by a package, you can get more information about the variables with ?gss\_cat.)

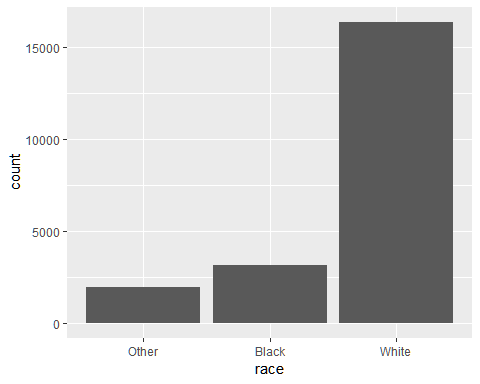
When factors are stored in a tibble, you can’t see their levels so easily. One way to see them is with count():

gss\_cat %>%  
 count(race)

## # A tibble: 3 x 2  
## race n  
## <fct> <int>  
## 1 Other 1959  
## 2 Black 3129  
## 3 White 16395

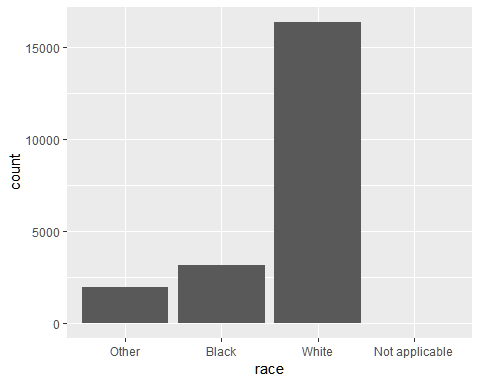
Or with a bar chart:

ggplot(gss\_cat, aes(race)) +  
 geom\_bar()



By default, ggplot2 will drop levels that don’t have any values. You can force them to display with:

ggplot(gss\_cat, aes(race)) +  
 geom\_bar() +  
 scale\_x\_discrete(drop = FALSE)



These levels represent valid values that simply did not occur in this dataset. Unfortunately, dplyr doesn’t yet have a drop option, but it will in the future.

When working with factors, the two most common operations are changing the order of the levels, and changing the values of the levels. Those operations are described in the sections below.

## Modifying factor levels

More powerful than changing the orders of the levels is changing their values. This allows you to clarify labels for publication, and collapse levels for high-level displays. The most general and powerful tool is fct\_recode(). It allows you to recode, or change, the value of each level. For example, take the gss\_cat$partyid:

gss\_cat %>% count(partyid)

## # A tibble: 10 x 2  
## partyid n  
## <fct> <int>  
## 1 No answer 154  
## 2 Don't know 1  
## 3 Other party 393  
## 4 Strong republican 2314  
## 5 Not str republican 3032  
## 6 Ind,near rep 1791  
## 7 Independent 4119  
## 8 Ind,near dem 2499  
## 9 Not str democrat 3690  
## 10 Strong democrat 3490

The levels are terse and inconsistent. Let’s tweak them to be longer and use a parallel construction.

gss\_cat %>%  
 mutate(partyid = fct\_recode(partyid,  
 "Republican, strong" = "Strong republican",  
 "Republican, weak" = "Not str republican",  
 "Independent, near rep" = "Ind,near rep",  
 "Independent, near dem" = "Ind,near dem",  
 "Democrat, weak" = "Not str democrat",  
 "Democrat, strong" = "Strong democrat"  
 )) %>%  
 count(partyid)

## # A tibble: 10 x 2  
## partyid n  
## <fct> <int>  
## 1 No answer 154  
## 2 Don't know 1  
## 3 Other party 393  
## 4 Republican, strong 2314  
## 5 Republican, weak 3032  
## 6 Independent, near rep 1791  
## 7 Independent 4119  
## 8 Independent, near dem 2499  
## 9 Democrat, weak 3690  
## 10 Democrat, strong 3490

fct\_recode() will leave levels that aren’t explicitly mentioned as is, and will warn you if you accidentally refer to a level that doesn’t exist.

To combine groups, you can assign multiple old levels to the same new level:

gss\_cat %>%  
 mutate(partyid = fct\_recode(partyid,  
 "Republican, strong" = "Strong republican",  
 "Republican, weak" = "Not str republican",  
 "Independent, near rep" = "Ind,near rep",  
 "Independent, near dem" = "Ind,near dem",  
 "Democrat, weak" = "Not str democrat",  
 "Democrat, strong" = "Strong democrat",  
 "Other" = "No answer",  
 "Other" = "Don't know",  
 "Other" = "Other party"  
 )) %>%  
 count(partyid)

## # A tibble: 8 x 2  
## partyid n  
## <fct> <int>  
## 1 Other 548  
## 2 Republican, strong 2314  
## 3 Republican, weak 3032  
## 4 Independent, near rep 1791  
## 5 Independent 4119  
## 6 Independent, near dem 2499  
## 7 Democrat, weak 3690  
## 8 Democrat, strong 3490

You must use this technique with care: if you group together categories that are truly different you will end up with misleading results.

If you want to collapse a lot of levels, fct\_collapse() is a useful variant of fct\_recode(). For each new variable, you can provide a vector of old levels:

gss\_cat %>%  
 mutate(partyid = fct\_collapse(partyid,  
 other = c("No answer", "Don't know", "Other party"),  
 rep = c("Strong republican", "Not str republican"),  
 ind = c("Ind,near rep", "Independent", "Ind,near dem"),  
 dem = c("Not str democrat", "Strong democrat")  
 )) %>%  
 count(partyid)

## # A tibble: 4 x 2  
## partyid n  
## <fct> <int>  
## 1 other 548  
## 2 rep 5346  
## 3 ind 8409  
## 4 dem 7180

Sometimes you just want to lump together all the small groups to make a plot or table simpler. That’s the job of fct\_lump():

gss\_cat %>%  
 mutate(relig = fct\_lump(relig)) %>%  
 count(relig)

## # A tibble: 2 x 2  
## relig n  
## <fct> <int>  
## 1 Protestant 10846  
## 2 Other 10637

The default behaviour is to progressively lump together the smallest groups, ensuring that the aggregate is still the smallest group. In this case it’s not very helpful: it is true that the majority of Americans in this survey are Protestant, but we’ve probably over collapsed.

Instead, we can use the n parameter to specify how many groups (excluding other) we want to keep:

gss\_cat %>%  
 mutate(relig = fct\_lump(relig, n = 10)) %>%  
 count(relig, sort = TRUE) %>%  
 print(n = Inf)

## # A tibble: 10 x 2  
## relig n  
## <fct> <int>  
## 1 Protestant 10846  
## 2 Catholic 5124  
## 3 None 3523  
## 4 Christian 689  
## 5 Other 458  
## 6 Jewish 388  
## 7 Buddhism 147  
## 8 Inter-nondenominational 109  
## 9 Moslem/islam 104  
## 10 Orthodox-christian 95

### Exercise 1

1. Explore the distribution of rincome (reported income). What makes the default bar chart hard to understand? How could you improve the plot?
2. What is the most common relig in this survey? What’s the most common partyid?
3. Which relig does denom (denomination) apply to? How can you find out with a table? How can you find out with a visualisation?

### Exercises 2

1. There are some suspiciously high numbers in tvhours. Is the mean a good summary?
2. For each factor in gss\_cat identify whether the order of the levels is arbitrary or principled.
3. Why did moving “Not applicable” to the front of the levels move it to the bottom of the plot?

### Exercises 3

1. How have the proportions of people identifying as Democrat, Republican, and Independent changed over time?
2. How could you collapse rincome into a small set of categories?

## Wrangling categorical data in R: Appendix

Amelia McNamara and Nicholas Horton’s paper, [*Wrangling categorical data in R*](https://peerj.com/preprints/3163/) has a good example of refactoring and releveling in “standard R” and “tidyverse”.

### McNamara et al. 2017

#### Abstract

Data wrangling is a critical foundation of data science, and wrangling of categorical data is an important component of this process. However, categorical data can introduce unique issues in data wrangling, particularly in real-world settings with collaborators and periodically-updated dynamic data. This paper discusses common problems arising from categorical variable transformations in R, demonstrates the use of factors, and suggests approaches to address data wrangling challenges. For each problem, we present at least two strategies for management, one in base R and the other from the `tidyverse.’ We consider several motivating examples, suggest defensive coding strategies, and outline principles for data wrangling to help ensure data quality and sound analysis.

#### Direct notes

Wrangling skills provide an intellectual and practical foundation for data science. Careless data cleaning operations can lead to errors or inconsistencies in analysis~. The wrangling of categorical data presents particular challenges and is highly relevant because many variables are categorical (e.g., gender, income bracket, U.S. state), and categorical data is often coded with numerical values. It is easy to break the relationship between category numbers and category labels without realizing it, thus losing the information encoded in a variable. If data sources change upstream (for example, if a domain expert is providing spreadsheet data at regular intervals), code that worked on the initial data may not generate an error message, but could silently produce incorrect results.