8 How to lump factors

What will this tutorial cover?

In this tutorial, we will talk about four functions that allow you to lump factors together: `fct_lump_min`, `fct_lump_prop`, `fct_lump_n`, `fct_lump_lowfreq`.

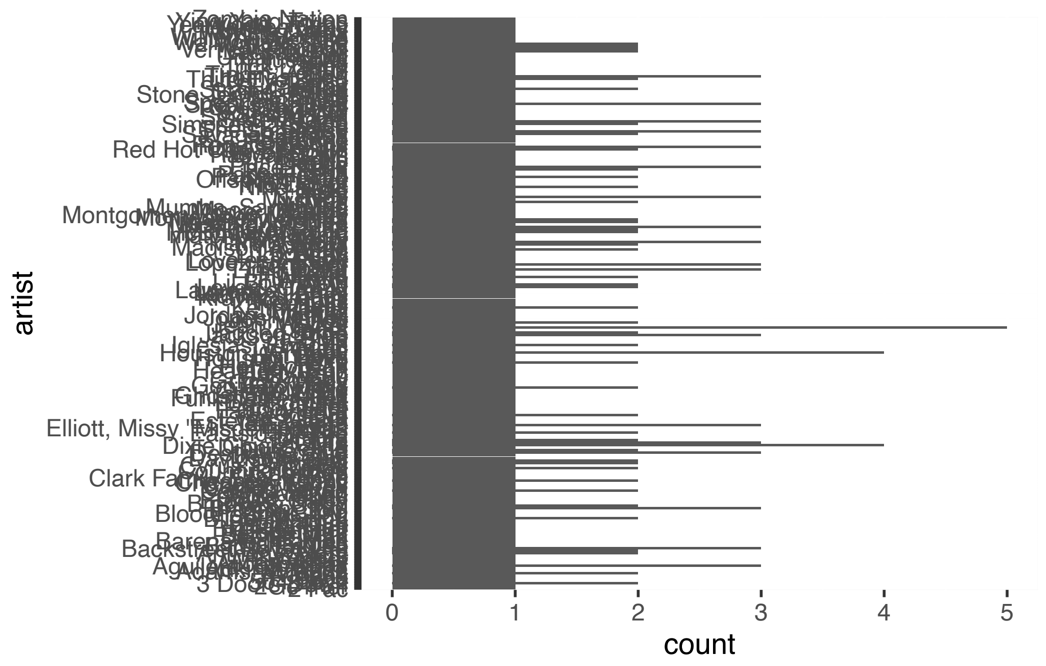
Who do I have to thank?

For this tutorial I relied on the official forcats documentation

A factor is a data structure in R that allows you to create a set of categories. We call these categories levels. It is well known in the psychological literature that we can only store a certain number of things in our working memory. Therefore, to help people make sense of categories, we shouldn't show too many of them. This is where the strength of lumping factors shows.

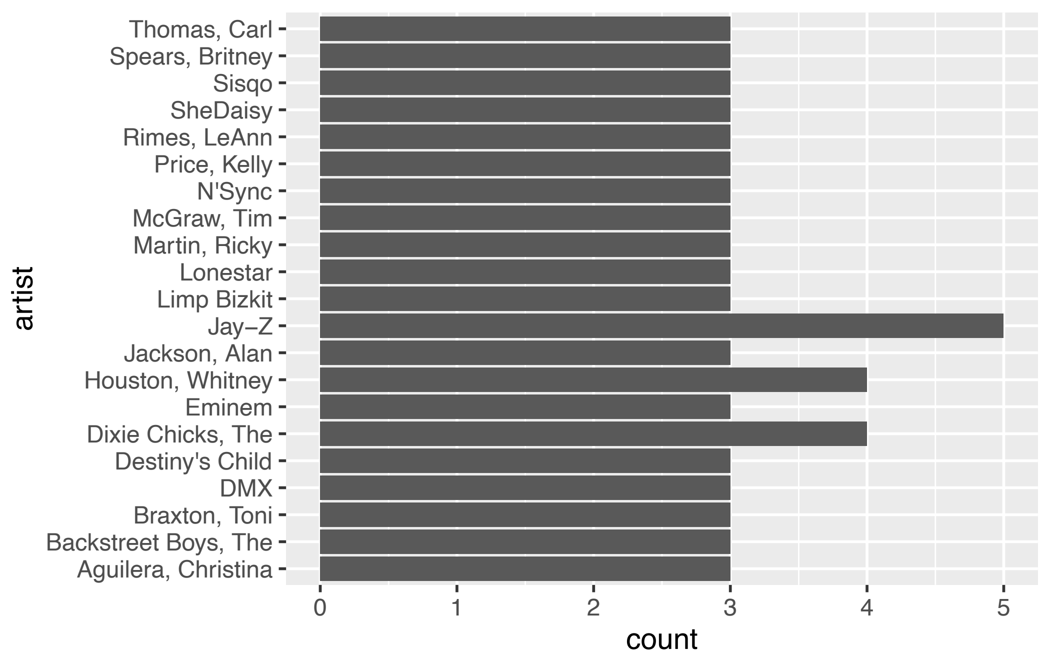
Lumping is nothing more than combining factor levels into a new and larger category. Let's say we want to visualize how many musicians made it to the top 100 in 2000. Here is a bar chart that shows what this looks like:

```
billboard %>%
  ggplot(aes(y = artist)) +
  geom_bar()
```



This visualization is completely incomprehensible. `fct_lump` comes to our rescue:

```
billboard %>%
  mutate(artist = fct_lump(as_factor(artist), 10)) %>%
  filter(artist != "Other") %>%
  ggplot(aes(y = artist)) +
  geom_bar()
```



We provide `fct_lump` with a number. The number indicates, well, what? 10 certainly does not indicate the remaining levels that were not categorized under "Other". Nor is 10 the levels we lumped together. As it turns out, `fct_lump` is a pretty obscure function because it chooses different methods based on its arguments. The tidyverse team no longer recommends the use of this function.

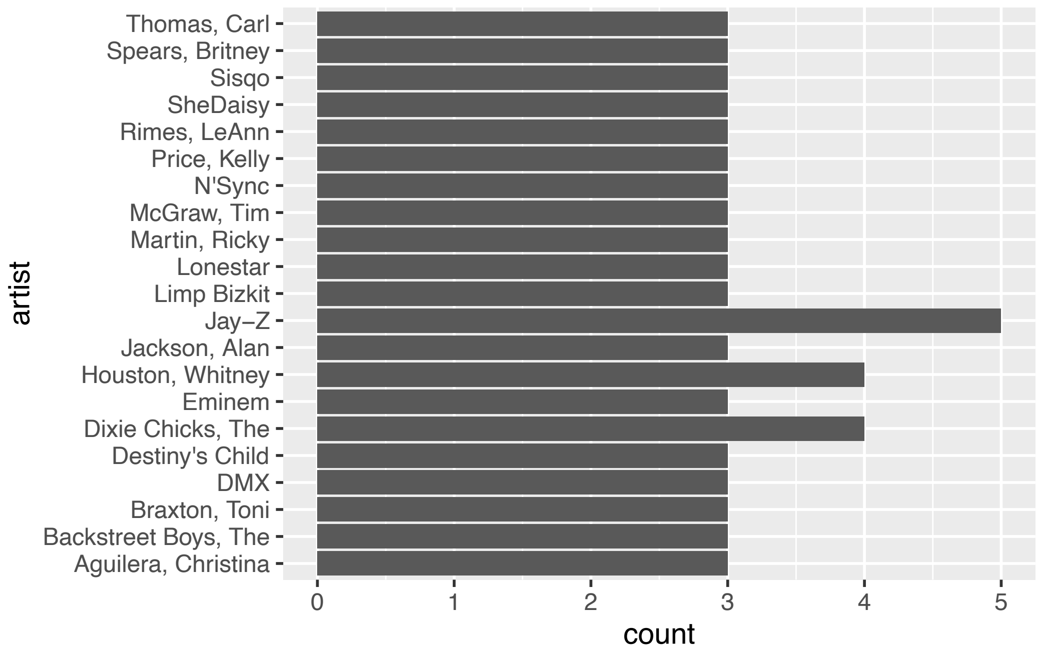
That's why four new functions were created in 2020, which we'll go through in this tutorial:

- `fct_lump_min`: lump levels that do not occur more than min times.
- `fct_lump_n`: lumps n of the most or least frequently occurring levels
- `fct_lump_prop`: lumps of levels that occur at most n times * prop
- `fct_lump_lowfreq`: lumps the least frequent levels

8.1 `fct_lump_min`: How to lump levels that occur no more than min times

Let's start with the simplest function. `fct_lump_min` summarizes all levels that do not appear more than min times. Let's stay with our billboard example. Let's assume we want to lump together all levels (or musicians) that made it into the top 100 less than three times in 2000:

```
billboard %>%
  mutate(artist = fct_lump_min(as_factor(artist), 3)) %>%
  filter(artist != "Other") %>%
  ggplot(aes(y = artist)) +
  geom_bar()
```



Clearly, all the other musicians had at least three songs that made it into the Top 100.

Let's try another example. The dataset `gss_cat` contains data on the General Social Survey. This survey contains, among other things, data on the marital status, age or stated income of persons in the years 2000 to 2014. The column `income` stands for the stated income:

```
table(gss_cat$rincome)
```

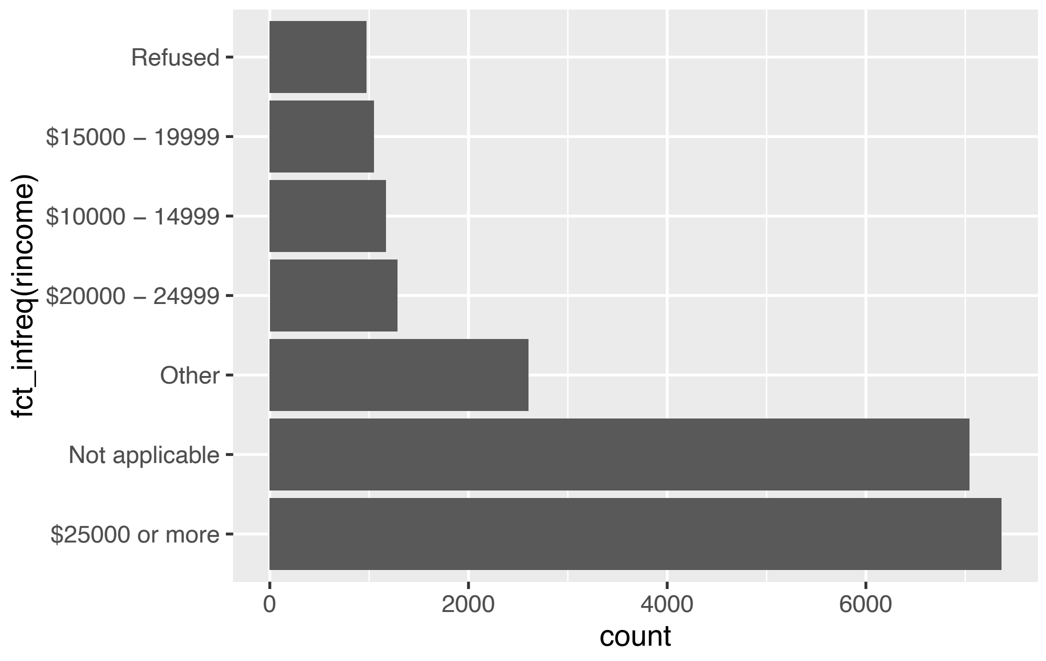
```

     No answer     Don't know        Refused $25000 or more $20000 - 24999 
           183            267            975           7363           1283 
$15000 - 19999 $10000 - 14999  $8000 to 9999  $7000 to 7999  $6000 to 6999 
          1048           1168            340            188            215 
 $5000 to 5999  $4000 to 4999  $3000 to 3999  $1000 to 2999       Lt $1000 
           227            226            276            395            286 
```

```
Not applicable
          7043
```

Now suppose we want to lump together all incomes that occur less than 600 times:

```r
gss_cat %>%
  mutate(rincome = fct_lump_min(rincome, 600)) %>%
  ggplot(aes(y = fct_infreq(rincome))) +
  geom_bar()
```

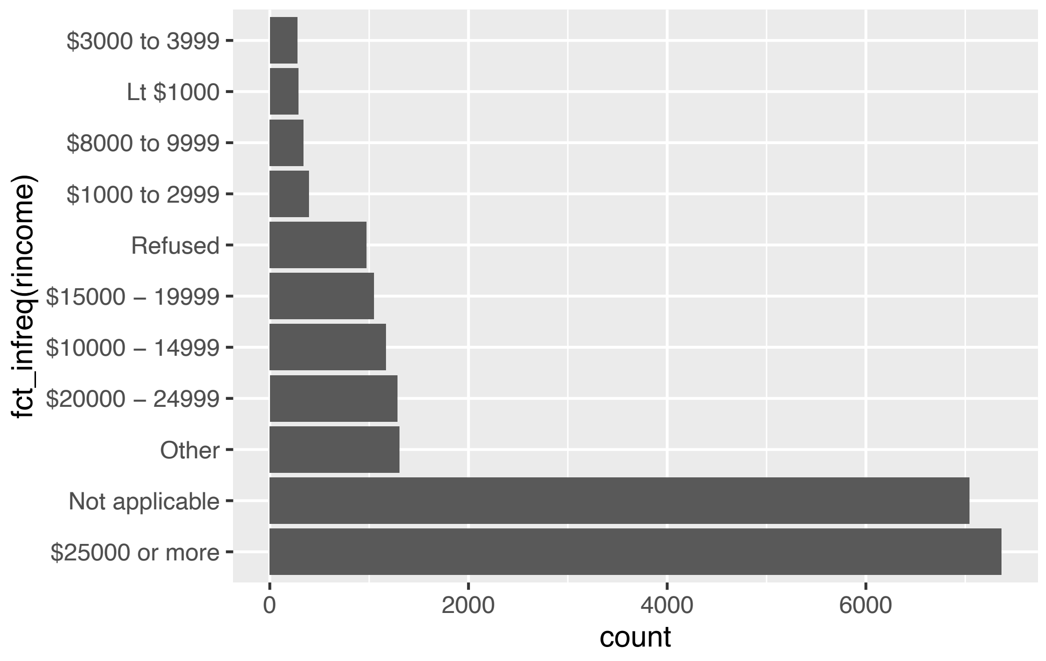


This time I kept the "Other" level. As you can see, six levels occurred more than 600 times and were therefore not lumped together.

8.2 `fct_lump_n`: Lumps n of the most or least frequent levels

Compared to `fct_lump_min`, `fct_lump_n` is not about the number of levels. Instead, it simply keeps the most frequent levels or the least frequent levels. For example, in our survey example, we might say that we want to lump all levels except the 10 most frequent:

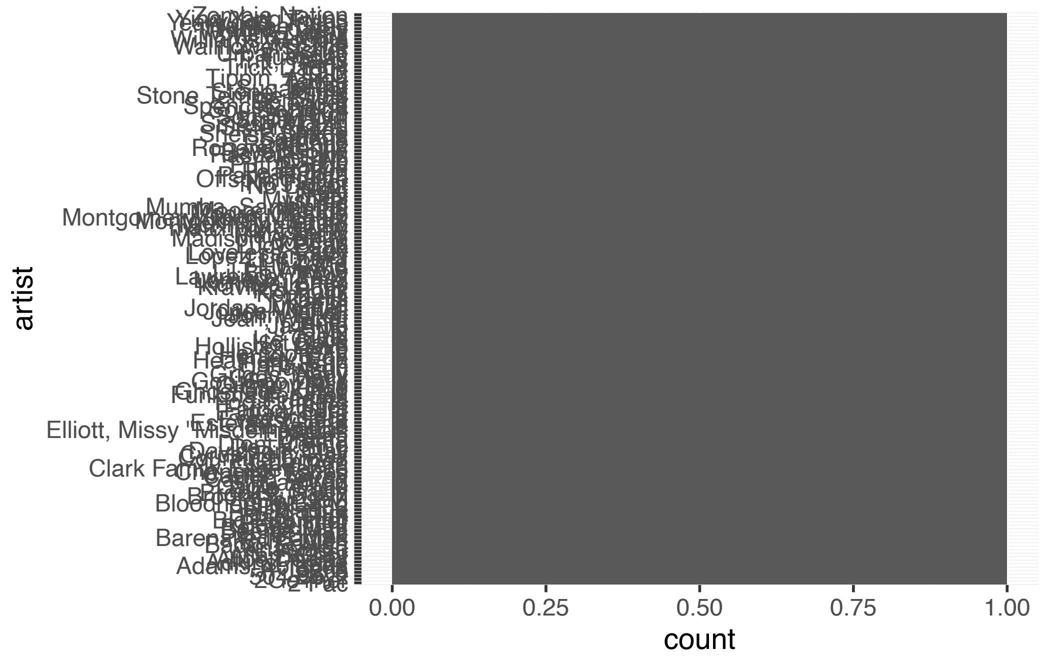
```r
gss_cat %>%
  mutate(rincome = fct_lump_n(rincome, n = 10)) %>%
  ggplot(aes(y = fct_infreq(rincome))) +
  geom_bar()
```



This gives a total of 11 levels (including the `Other` level).

If you specify the `n` argument with a negative number, the function will lump all the levels that occur most often (exactly the opposite of what a positive number does). For example, we could lump together the 5 musicians with the most songs in the Top 100.

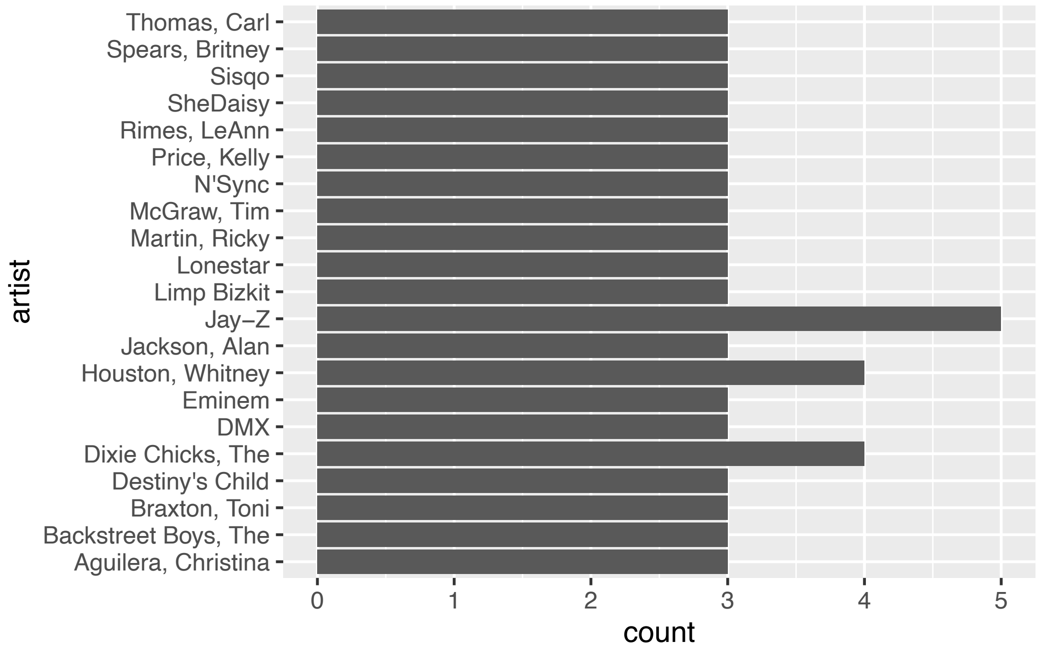
```r
billboard %>%
  mutate(artist = fct_lump_n(artist, n = -5)) %>%
  filter(artist != "Other") %>%
  ggplot(aes(y = artist)) +
  geom_bar()
```



This is not a bar chart you want to design, but it proofs the point.

Let's try another example. Let's say we want to lump all levels together except for the 5 most common:

```r
billboard %>%
  mutate(artist = fct_lump_n(artist, 5)) %>%
  filter(artist != "Other") %>%
  ggplot(aes(y = artist)) +
  geom_bar()
```

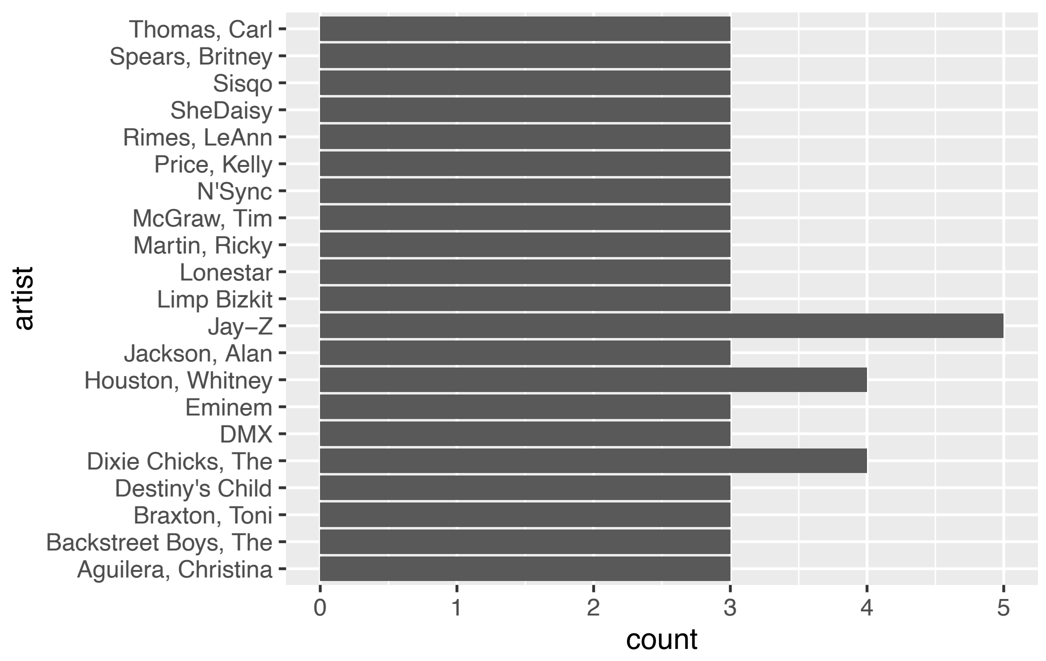


Clearly, these are not five levels What went wrong? It turns out that many levels occur three times. So we have to decide what to do with the levels that occur the same number of times. The most common three levels are Jay-Z, Whitney Houston and The Dixie Chicks. But what should be the 4th and 5th most frequent levels?

If you don't give the function any additional information, `fct_lump_n` will show you all the levels whose number falls below the last level, which is clearly one of the most frequent levels.

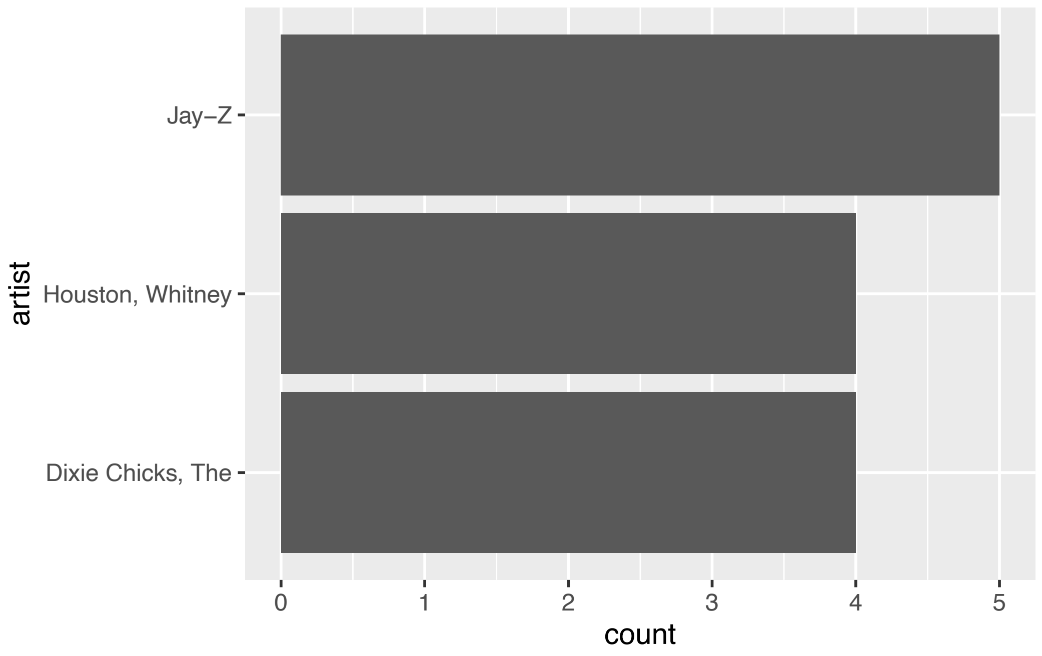
You can change this behavior with the `ties.method` argument. The default argument is `min`, which we have just seen:

```
billboard %>%
  mutate(artist = fct_lump_n(artist, 5, ties.method = "min")) %>%
  filter(artist != "Other") %>%
  ggplot(aes(y = artist)) +
  geom_bar()
```



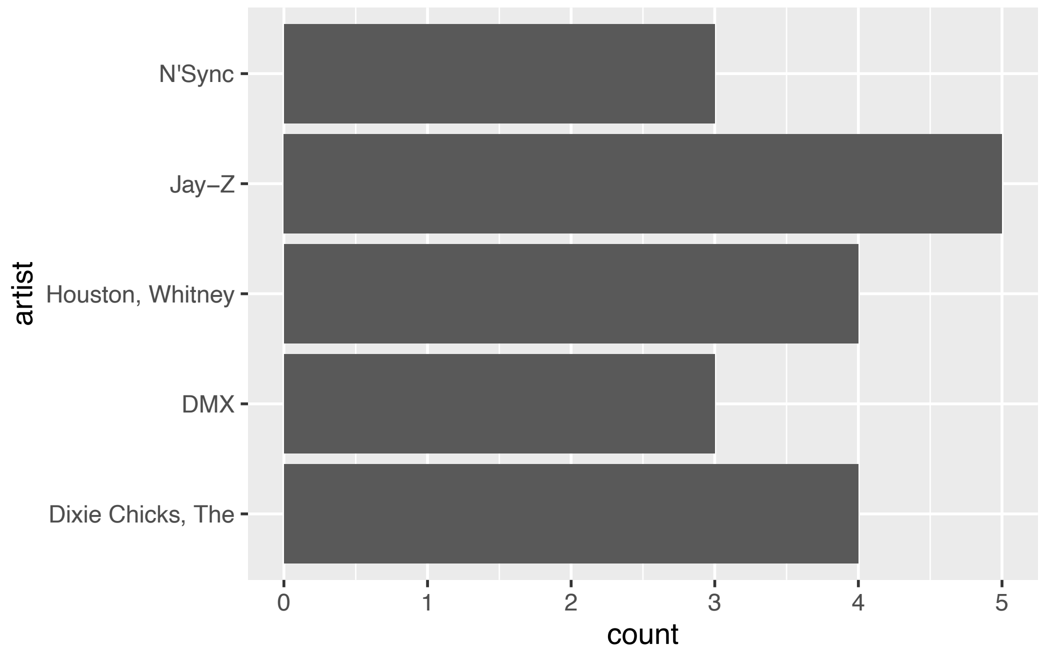
The other options are "average", "first", "last", "random" and "max". We'll not go through them all in detail here, but let's take a look at two of them. "max"" removes all levels that cannot be uniquely identified.

```
billboard %>%
  mutate(artist = fct_lump_n(artist, 5, ties.method = "max")) %>%
  filter(artist != "Other") %>%
  ggplot(aes(y = artist)) +
  geom_bar()
```



"random" randomly selects the levels that cannot be uniquely identified as the most frequent levels:

```
billboard %>%
  mutate(artist = fct_lump_n(artist, 5, ties.method = "random")) %>%
  filter(artist != "Other") %>%
  ggplot(aes(y = artist)) +
  geom_bar()
```



You will probably see another set of steps, but that is a feature and not a bug.

8.3 `fct_lump_prop`: Lumps levels that appear no more than n * prop times

The next function is a bit more complicated. To understand what it does, we need a bit of math.

First we count how many times all levels occur in total. Let's try it with our `gss_cat` data frame:

```
(total_count_income <- gss_cat %>% count(rincome) %>% {sum(.$n)})
```

```
[1] 21483
```

We have data of 21483 incomes. Next, we choose a specific income range and that is the range between "$20000 - 24999". This range occurs so often:

```
(count_one_range <- gss_cat$rincome[gss_cat$rincome == "$20000 - 24999"] %>%
  length)
```

```
[1] 1283
```

This figure represents about 6% of all data points on people's incomes:

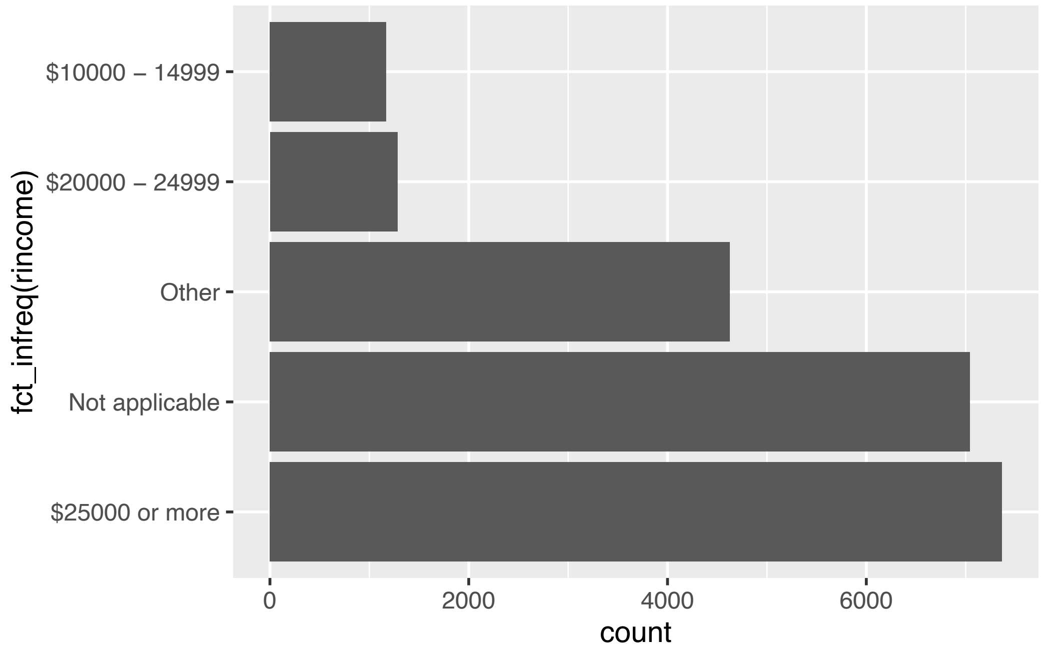
```
count_one_range / total_count_income
```

```
[1] 0.05972164
```

And this is the number we are looking for in `fct_lump_prop`. It represents the percentage at which a particular level occurs within the total number of levels.

Now suppose we want to lump together all the levels that occur in less than 5% of all counts:

```
gss_cat %>%
  mutate(rincome = fct_lump_prop(rincome, .05)) %>%
  ggplot(aes(y = fct_infreq(rincome))) +
  geom_bar()
```



Let's prove this with some Tidyverse functions:

```
gss_cat %>%
  count(rincome, name = "count_per_income_range") %>%
  select(rincome, count_per_income_range) %>%
  mutate(
    total_count_income = sum(count_per_income_range),
    percentage = count_per_income_range / total_count_income
  ) %>%
  filter(percentage >= .05)
```

```
# A tibble: 4 x 4
  rincome        count_per_income_range total_count_income percentage
  <fct>                           <int>              <int>      <dbl>
1 $25000 or more                   7363              21483     0.343
2 $20000 - 24999                   1283              21483     0.0597
3 $10000 - 14999                   1168              21483     0.0544
4 Not applicable                   7043              21483     0.328
```

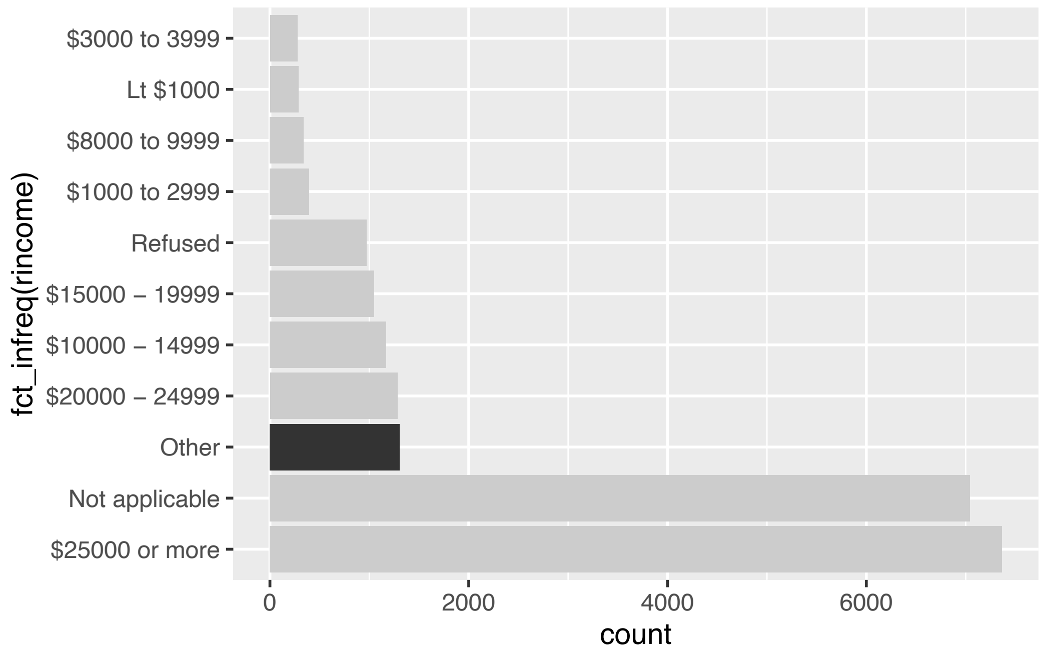
As you can see, all four levels occur more often than 6%.

8.4 `fct_lump_lowfreq`: Lumps the least frequent levels

The last function is quite nifty. It has no additional arguments except `other_level`, which is used to specify the name of the "other" level.

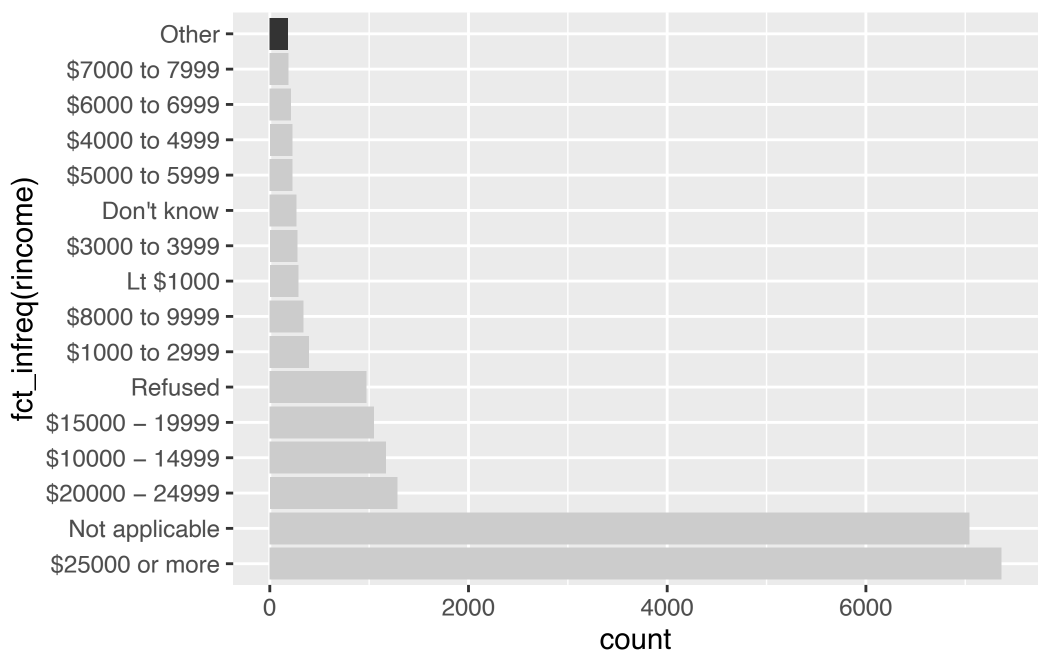
This is how it works. You may have realized that some of these four functions cause the `Other` level not to be the least common level:

```
gss_cat %>%
  mutate(
    rincome = fct_lump_n(rincome, n = 10),
    color_coding_rincome = ifelse(rincome == "Other", "a", "b")
  ) %>%
  ggplot(aes(y = fct_infreq(rincome),
             fill = color_coding_rincome)) +
  scale_fill_manual(values = c("grey20", "grey80")) +
  geom_bar(show.legend = FALSE)
```



`fct_lump_lowfreq` simply ensures that so many levels are grouped together that the "Other" is still the least frequent level. Let's try the same example:

```
gss_cat %>%
  mutate(
    rincome = fct_lump_lowfreq(rincome),
    color_coding_rincome = ifelse(rincome == "Other", "a", "b")
  ) %>%
  ggplot(aes(y = fct_infreq(rincome),
             fill = color_coding_rincome)) +
  scale_fill_manual(values = c("grey20", "grey80")) +
  geom_bar(show.legend = FALSE)
```



This is an amazingly simple but effective idea.

Summary

Here is what you can take from this tutorial.

- Do not use `fct_lump` anymore. Use the other factor functions instead
- If you need to visualize factors quickly, use `fct_lump_lowfreq` to find the best size of "Other".
- With `fct_lump_min` you lump all levels that occur less than min times, with `fct_lump_n` you keep the most frequent n levels.