9 How to order factor levels

What will this tutorial cover?

In this tutorial you will learn about four functions that allow you to order factor levels: `fct_relevel`, `fct_infreq`, `fct_reorder`, and `fct_reorder2`.

Who do I have to thank?

I have Claus O. Wilke to thank for this tutorial. He wrote a popular presentation explaining the topic excellently (also check out his tweet). My thanks also go to the makers of the datavizpyr website, who wrote a great article about the fct_reorder function.

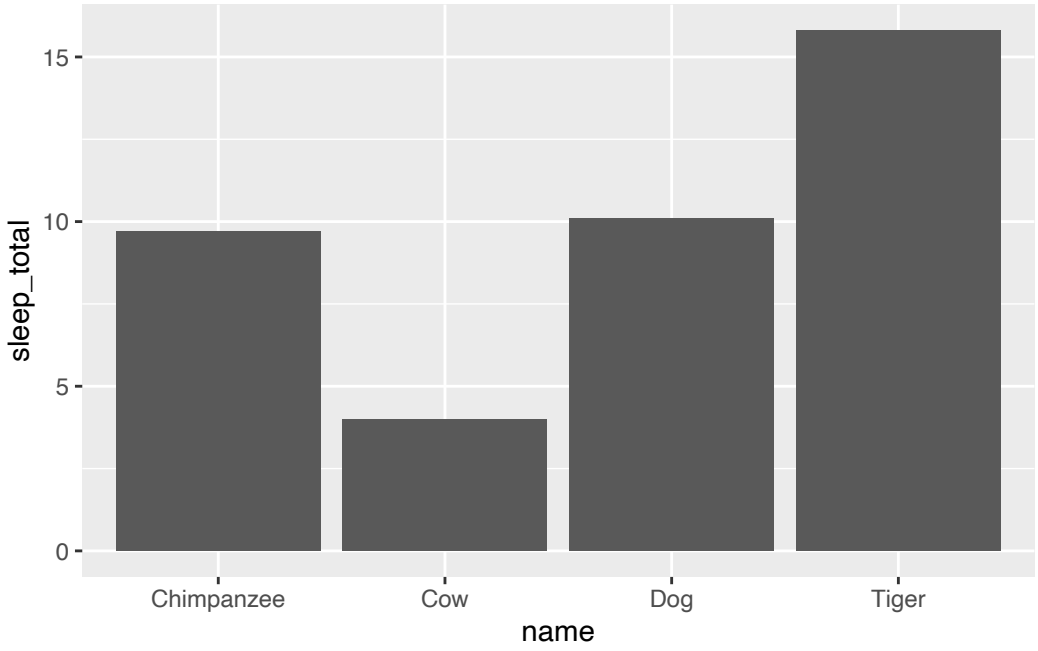
A factor is an ordered data structure in R. What is ordered in a factor are its levels. Usually, you need to know the order of levels when you try to visualize factor columns. Let's say you want to order the bars in your bar chart. Or you want to order the facets in your chart. Or you want to order the lines in your line chart. In this tutorial, we will delve into four functions from the forcats package that allow us to order factor levels:

- `fct_relevel`: Order the levels manually
- `fct_infreq`: Order the levels based on how frequently each level occurs
- `fct_reorder`: Order the levels based on the values of a numeric variable
- `fct_reorder2`: Order the levels based on the values of two numeric variables

9.1 How to order factor levels manually

Suppose you created this bar chart showing how many hours cows, dogs, tigers, and chimpanzees sleep per day:

```
msleep %>%
  filter(name %in% c("Cow", "Dog", "Tiger", "Chimpanzee")) %>%
  ggplot(aes(x = name, y = sleep_total)) +
  geom_col()
```



Let's say you want to put cows and dogs first in the bar chart. To make this task a little easier for us, let me first show how this works without a visualization. Here is the same column as a factor:

```
animal_factor <- as.factor(c("Cow", "Dog", "Tiger", "Chimpanzee"))
```

You can see how the factor levels are arranged by running the `levels` function:

```
levels(animal_factor)
```

```
[1] "Chimpanzee" "Cow"        "Dog"        "Tiger"
```

If you look at our bar chart again, you will notice that the output has the same order as the bars in the bar chart. We can change this order manually by using the function `fct_relevel`. It literally says what it does. "re", according to the Webster Dictionary, means to place something in front. So, in essence, we are putting some levels in front of others with `fct_relevel`. Let's try this:

```
fct_relevel(.f = animal_factor,
            "Cow", "Dog")
```

```
[1] Cow        Dog        Tiger      Chimpanzee
Levels: Cow Dog Chimpanzee Tiger
```

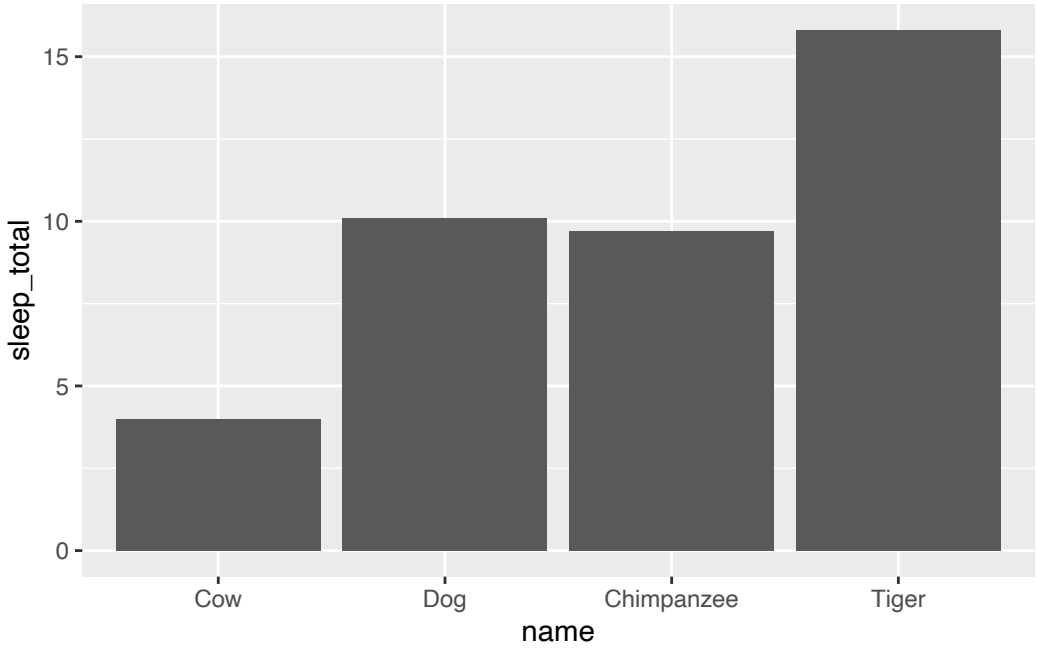
The first argument is the factor column. Then you list the levels you want to place first. You could also do this with a vector:

```
fct_relevel(.f = animal_factor,
            c("Cow", "Dog"))
```

```
[1] Cow        Dog        Tiger      Chimpanzee
Levels: Cow Dog Chimpanzee Tiger
```

Let's try this in our visualization:

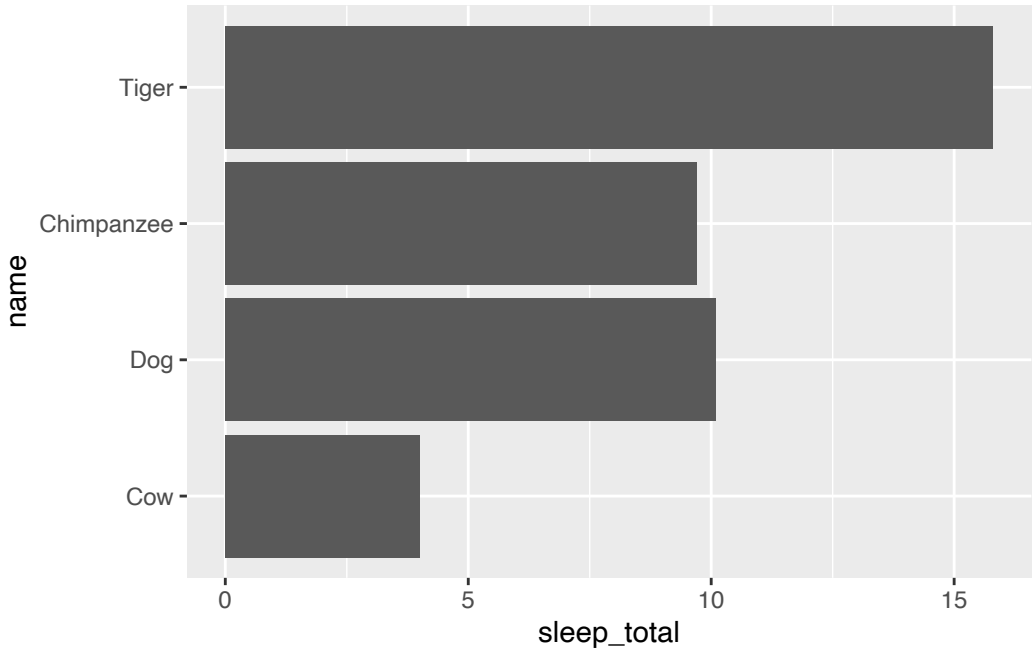
```
msleep %>%
  dplyr::filter(name %in% c("Cow", "Dog", "Tiger", "Chimpanzee")) %>%
  mutate(
    name = fct_relevel(name, c("Cow", "Dog"))
  ) %>%
  ggplot(aes(x = name, y = sleep_total)) +
  geom_col()
```



You may wonder what scheme the other levels are ordered by. By default by alphabetical order. In our case, the "C" in "Chimpanzee" comes before the "T" in "Tiger".

If we put the factor on the y-axis and the sleep hours on the x-axis, we can see that the values are arranged from bottom to top and not from top to bottom:

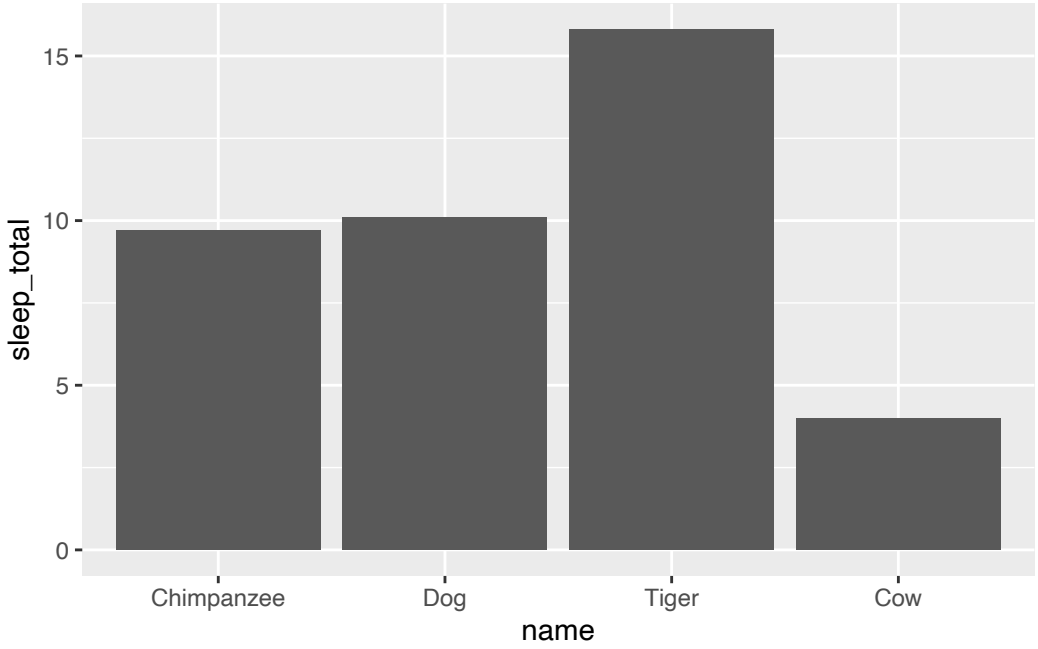
```
msleep %>%
  filter(name %in% c("Cow", "Dog", "Tiger", "Chimpanzee")) %>%
  mutate(
    name = fct_relevel(name, c("Cow", "Dog"))
  ) %>%
  ggplot(aes(x = sleep_total, y = name)) +
  geom_col()
```



Here is another interesting trick. Instead of placing the levels to the front, we can place them wherever we like with the `after` argument. Suppose we want to place the Cow after the Tiger:

```
msleep %>%
  filter(name %in% c("Cow", "Dog", "Tiger", "Chimpanzee")) %>%
  mutate(
    name = fct_relevel(name, "Cow", after = 3)
```

```
) %>%
ggplot(aes(x = name, y = sleep_total)) +
geom_col()
```



I wish you could use a string for the `after` argument, but you must specify a number. Your levels will be placed at the nth + 1 position of this number. In our case the layer `cow` is placed at the 4th position (3th + 1).

9.2 How to order the levels based on how frequently each level occurs

For the next trick, we must first note that it will hardly work with our previous data set. The reason for this is that each level in our factor column occurs only once:

```
msleep %>%
  filter(name %in% c("Cow", "Dog", "Tiger", "Chimpanzee")) %>%
  count(name)
```

```
# A tibble: 4 x 2
```

```
  name           n
  <chr>      <int>
1 Chimpanzee     1
2 Cow            1
3 Dog            1
4 Tiger          1
```

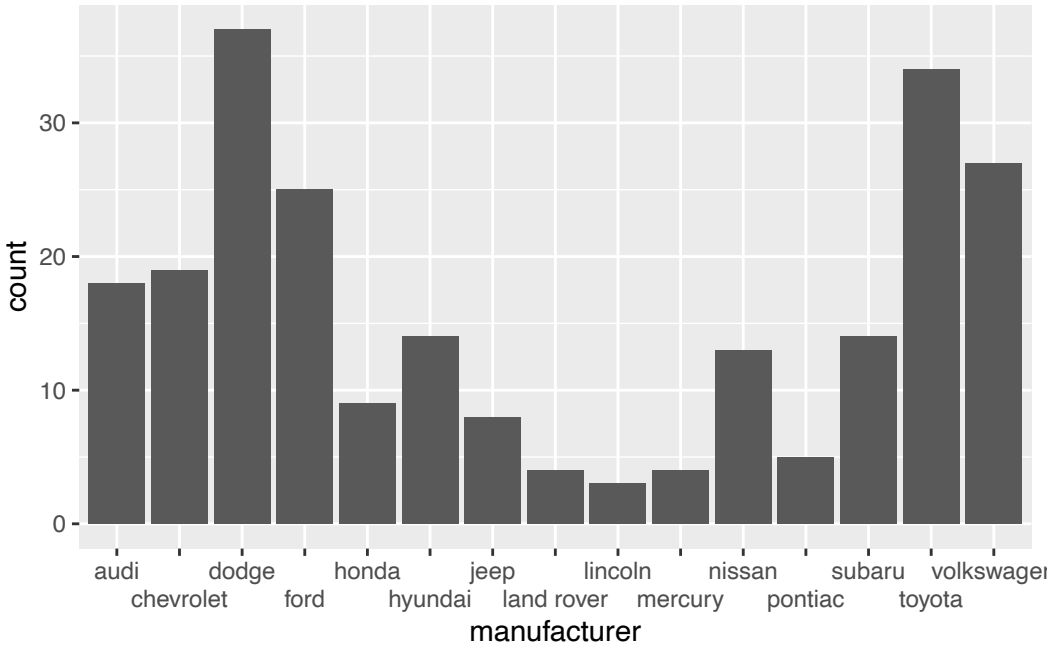
You can't really order the levels if they don't differ in frequency. So let's find a data frame where the factor levels differ. Let us take the data frame `mpg` with the `manufacturer` column:

```
mpg %>%
  count(manufacturer)
```

```
# A tibble: 15 x 2
   manufacturer     n
   <chr>        <int>
 1 audi            18
 2 chevrolet       19
 3 dodge           37
 4 ford            25
 5 honda            9
 6 hyundai         14
 7 jeep             8
 8 land rover       4
 9 lincoln          3
10 mercury          4
11 nissan          13
12 pontiac          5
13 subaru          14
14 toyota          34
15 volkswagen      27
```

Now suppose we want to show how many times each manufacturer appears in the data frame:

```
mpg %>%
  ggplot(aes(x = manufacturer)) +
  geom_bar() +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
```



Again, let's first illustrate the new function with a simple example. Here again is our column manufacturer factor:

```
manufacturer_factor <- as.factor(mpg$manufacturer)
levels(manufacturer_factor)
```

```
 [1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"
 [6] "hyundai"    "jeep"       "land rover" "lincoln"    "mercury"
[11] "nissan"     "pontiac"    "subaru"     "toyota"     "volkswagen"
```

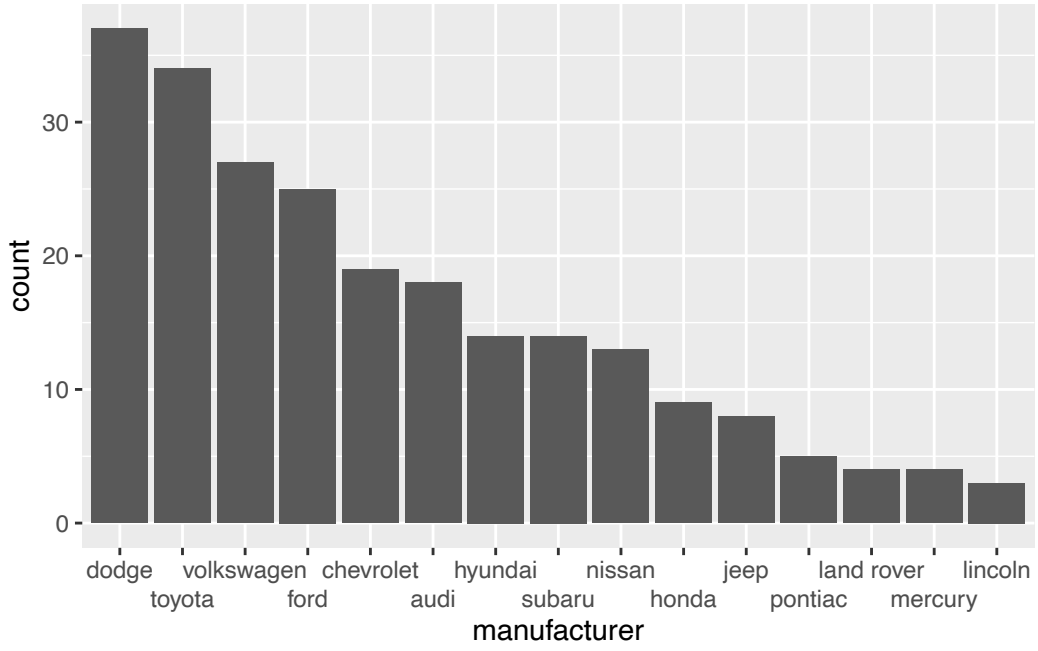
Again, you can see that they are in alphabetical order. With the function `fct_infreq` we can change the order according to how frequent each level occurs:

```
manufacturer_factor %>% fct_infreq %>%
  levels
```

```
 [1] "dodge"      "toyota"     "volkswagen" "ford"       "chevrolet"
 [6] "audi"       "hyundai"    "subaru"     "nissan"     "honda"
[11] "jeep"       "pontiac"    "land rover" "mercury"    "lincoln"
```

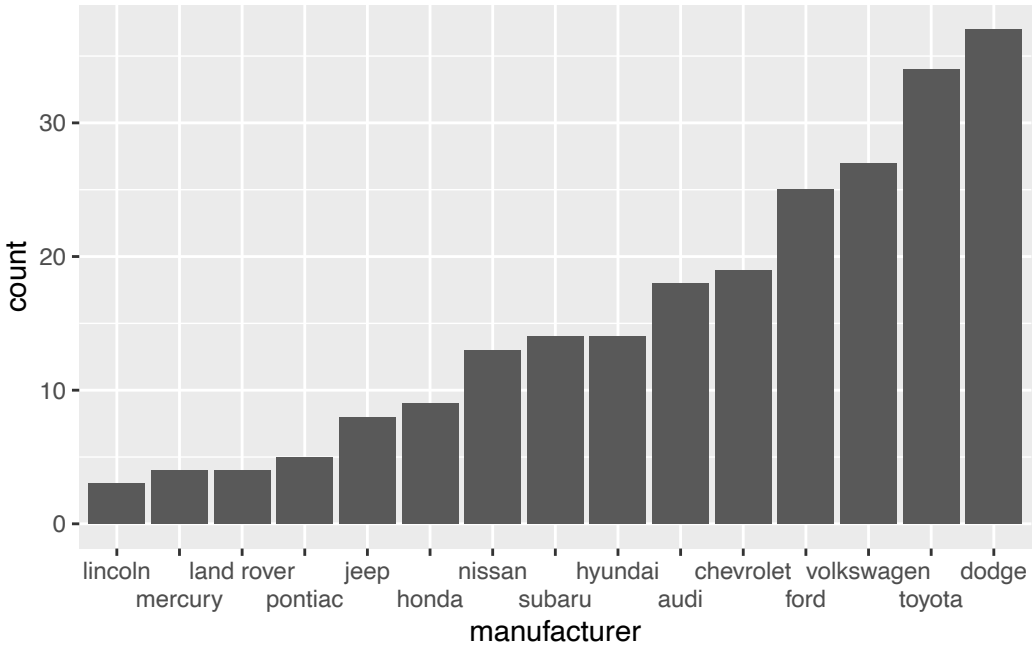
To check this, we can visualize the bar chart again:

```
mpg %>%
  mutate(manufacturer = fct_infreq(manufacturer)) %>%
  ggplot(aes(x = manufacturer)) +
  geom_bar() +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
```



If you want to reverse the order of the levels, you can use the `fct_rev` function:

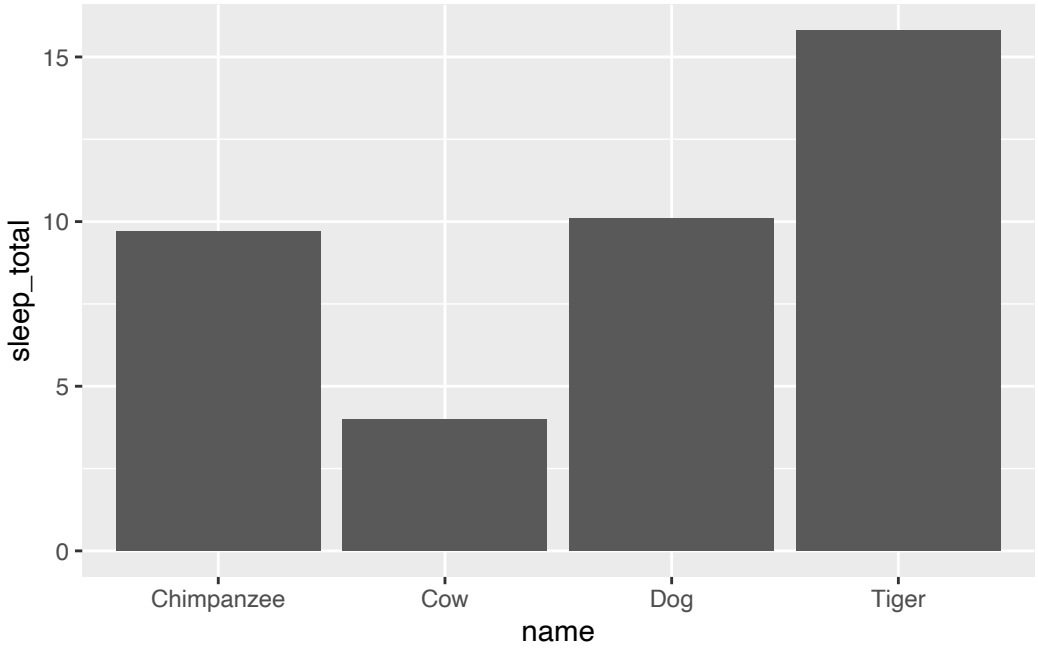
```
mpg %>%
  mutate(manufacturer = fct_infreq(manufacturer) %>% fct_rev) %>%
  ggplot(aes(x = manufacturer)) +
  geom_bar() +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
```



9.3 How to order the levels based on the values of a numeric variable

So far, we have changed the order of the levels based only on the information from the factor column. Next, let's discuss how we can order the levels of a factor column based on another numeric variable. Here is a typical use case for this: you create a bar chart with a discrete variable on the x-axis and a continuous variable on the y-axis:

```
msleep %>%
  filter(name %in% c("Cow", "Dog", "Tiger", "Chimpanzee")) %>%
  ggplot(aes(x = name, y = sleep_total)) +
  geom_col()
```



We can see that each factor level is associated with the continuous variable `sleep_total`:

```
(sleep_data <- msleep %>%
  filter(name %in% c("Cow", "Dog", "Tiger", "Chimpanzee")) %>%
  select(name, sleep_total))
```

```
# A tibble: 4 x 2
  name       sleep_total
  <chr>            <dbl>
1 Cow                4
2 Dog               10.1
3 Chimpanzee         9.7
4 Tiger             15.8
```

The function `fct_reorder` allows to order the levels based on another continuous variable. In our case `sleep_total`. Let's try it out:

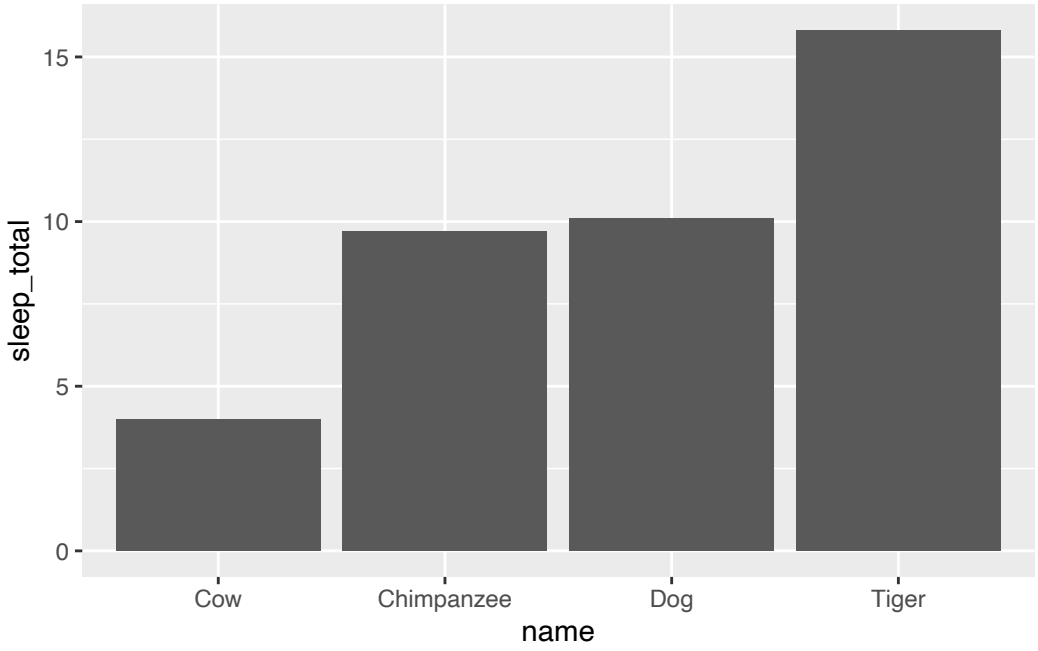
```
sleep_data %>%
  mutate(
    name = as.factor(name) %>% fct_reorder(sleep_total)
  ) %>%
```

```
    pull(name)
```

```
[1] Cow        Dog        Chimpanzee Tiger
Levels: Cow Chimpanzee Dog Tiger
```

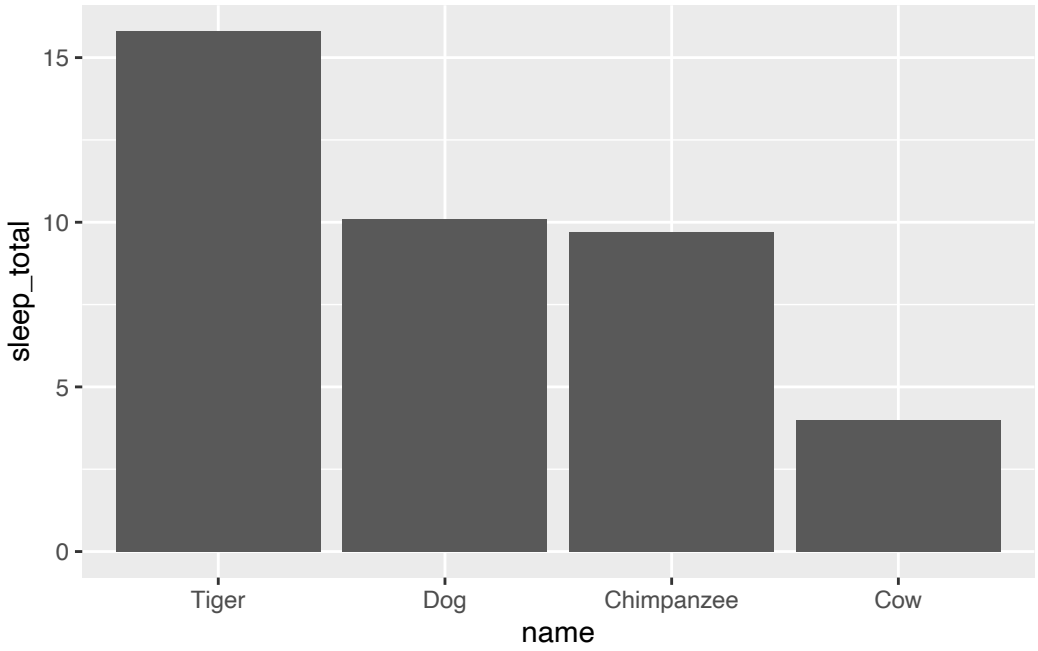
Cows apparently have the fewest hours of sleep, followed by dogs, chimpanzees and tigers. Let's visualize this:

```
msleep %>%
  filter(name %in% c("Cow", "Dog", "Tiger", "Chimpanzee")) %>%
  mutate(
    name = as.factor(name) %>% fct_reorder(sleep_total)
  ) %>%
  ggplot(aes(x = name, y = sleep_total)) +
  geom_col()
```



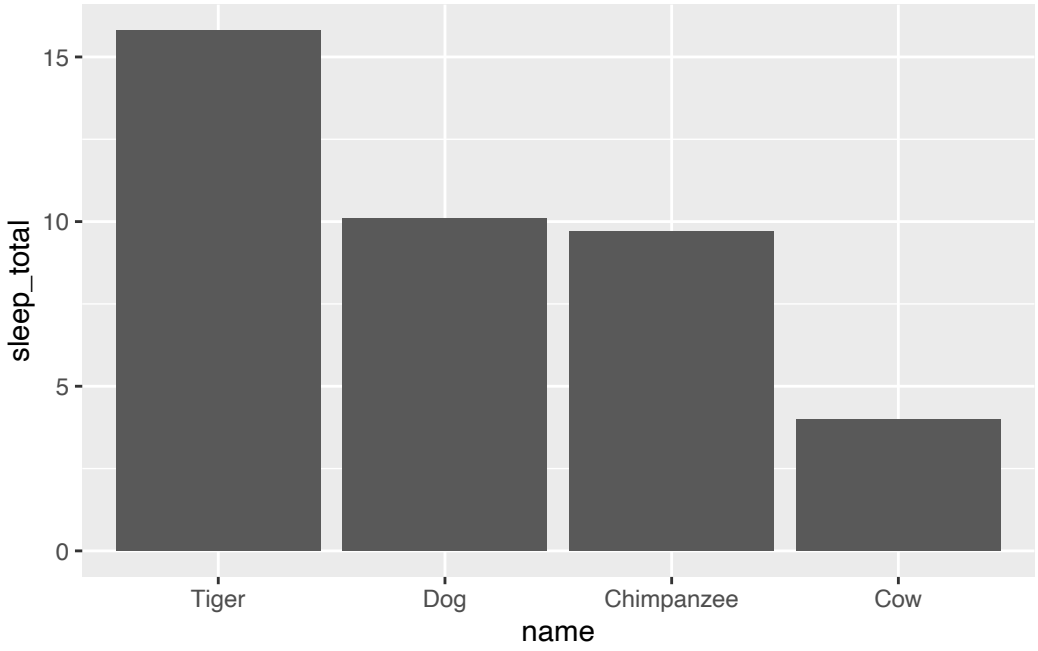
That is correct. We can reverse the order in two ways. First, by setting the `.desc` argument to `TRUE`:

```r
msleep %>%
  filter(name %in% c("Cow", "Dog", "Tiger", "Chimpanzee")) %>%
  mutate(
    name = as.factor(name) %>%
      fct_reorder(sleep_total, .desc = TRUE)
  ) %>%
  ggplot(aes(x = name, y = sleep_total)) +
  geom_col()
```



Then with the function `fct_rev`:

```r
msleep %>%
  filter(name %in% c("Cow", "Dog", "Tiger", "Chimpanzee")) %>%
  mutate(
    name = as.factor(name) %>%
      fct_reorder(sleep_total) %>%
      fct_rev
  ) %>%
  ggplot(aes(x = name, y = sleep_total)) +
  geom_col()
```



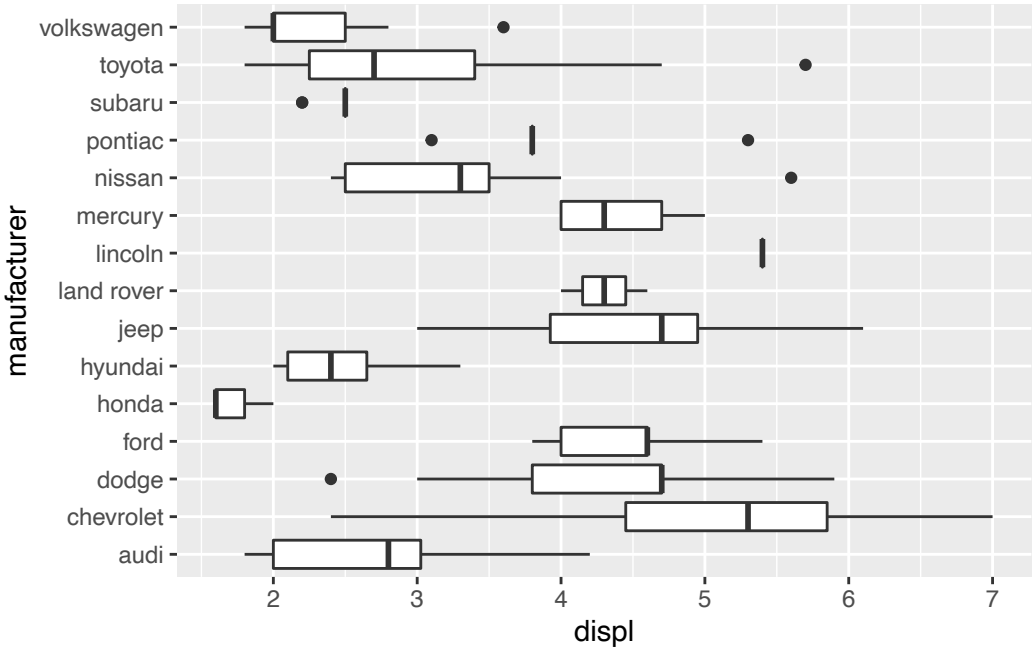
If you take a look at the documentation, you will find the `.fun` argument. The documentation says the following:

> "n summary function. It should take one vector for fct_reorder, and two vectors for fct_reorder2, and return a single value."

By default, this argument is set to the function `median`. Frankly, I didn't understand this argument at first.

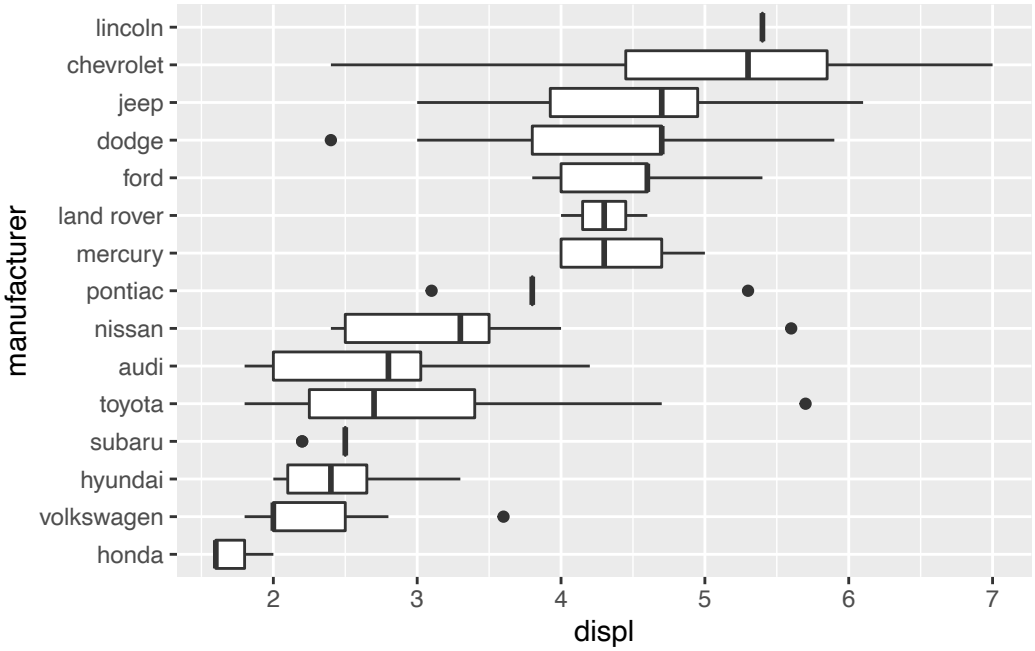
The first thing you need to know is that the `.fun` argument is useful when you have many values for each factor level. This is always the case when you create a boxplot or a violin diagram. Like this:

```
mpg %>%
  ggplot(aes(x = displ, y = manufacturer)) +
  geom_boxplot()
```



Before we explain the argument in detail, let's order the levels with the function `fct_reorder`:

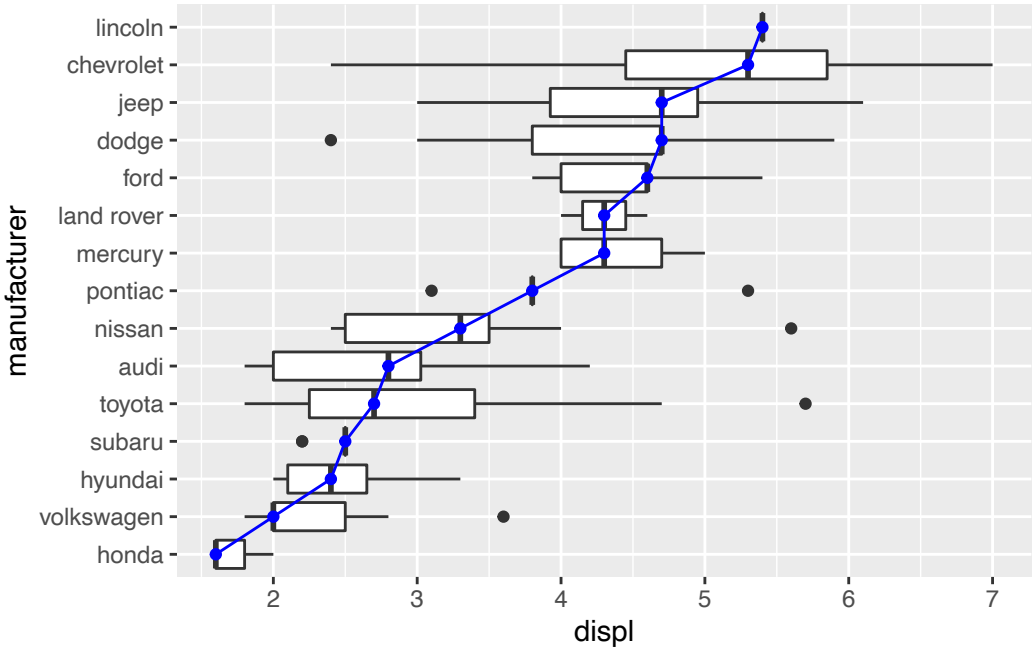
```
mpg %>%
  mutate(
    manufacturer = fct_reorder(as.factor(manufacturer),
                               displ)
  ) %>%
  ggplot(aes(x = displ, y = manufacturer)) +
  geom_boxplot()
```



Clearly, the levels have been ordered. But how? The higher the level, the larger the value of `displ`. I told you that by default the levels are ordered by the median values of the continuous variable.

We can illustrate this by adding a line chart to the plot:

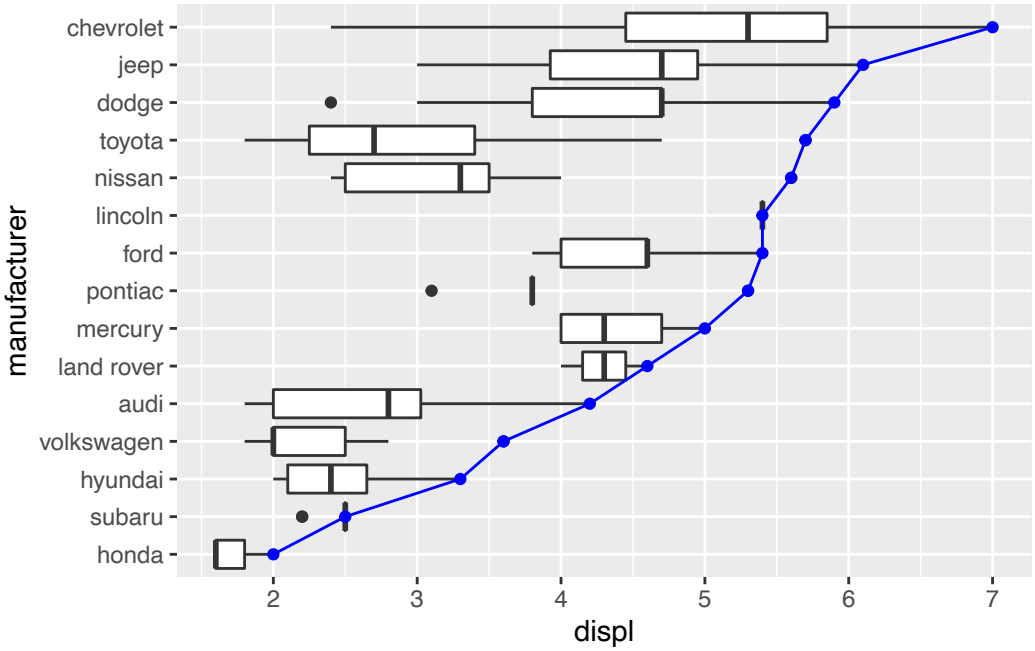
```
mpg %>%
  mutate(
    manufacturer = fct_reorder(as.factor(manufacturer),
                               displ)
  ) %>%
  ggplot(aes(x = displ, y = manufacturer)) +
  geom_boxplot() +
  stat_summary(geom = "point", fun = "median", color = "blue") +
  stat_summary(geom = "line", fun = "median", color = "blue", group = 1)
```



Some of you may know that the middle line in boxplots represents the median value. This is exactly what we see here.

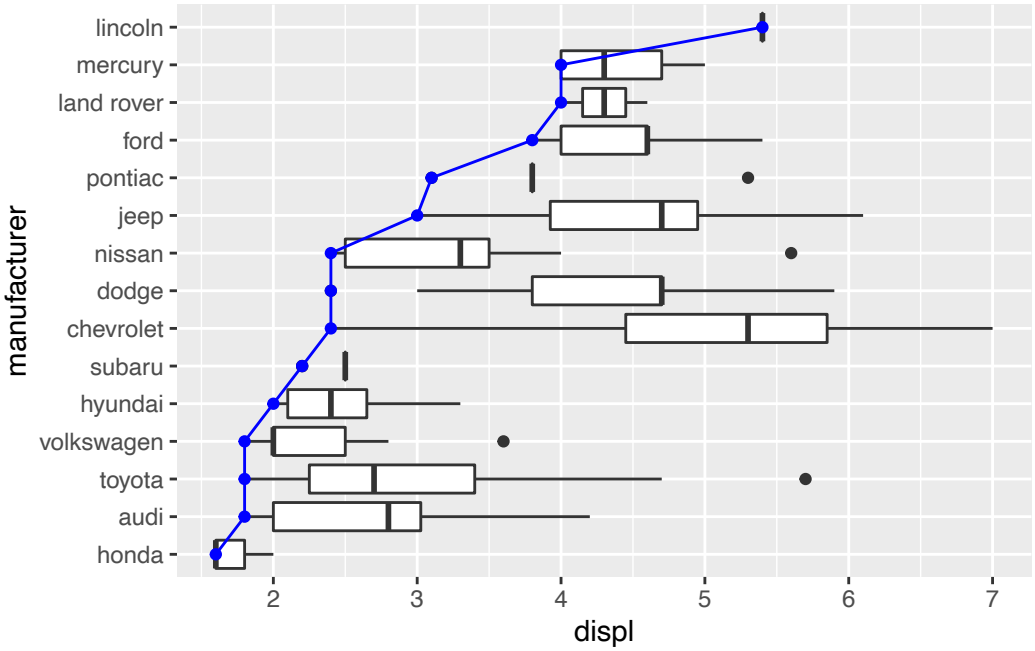
Say we want to change the order according to the maximum value of the continuous variable (see `.fun = max`):

```
mpg %>%
  mutate(
    manufacturer = fct_reorder(as.factor(manufacturer),
                               displ, .fun = max)
  ) %>%
  ggplot(aes(x = displ, y = manufacturer)) +
  geom_boxplot() +
  stat_summary(geom = "point", fun = "max", color = "blue") +
  stat_summary(geom = "line", fun = "max", color = "blue", group = 1)
```



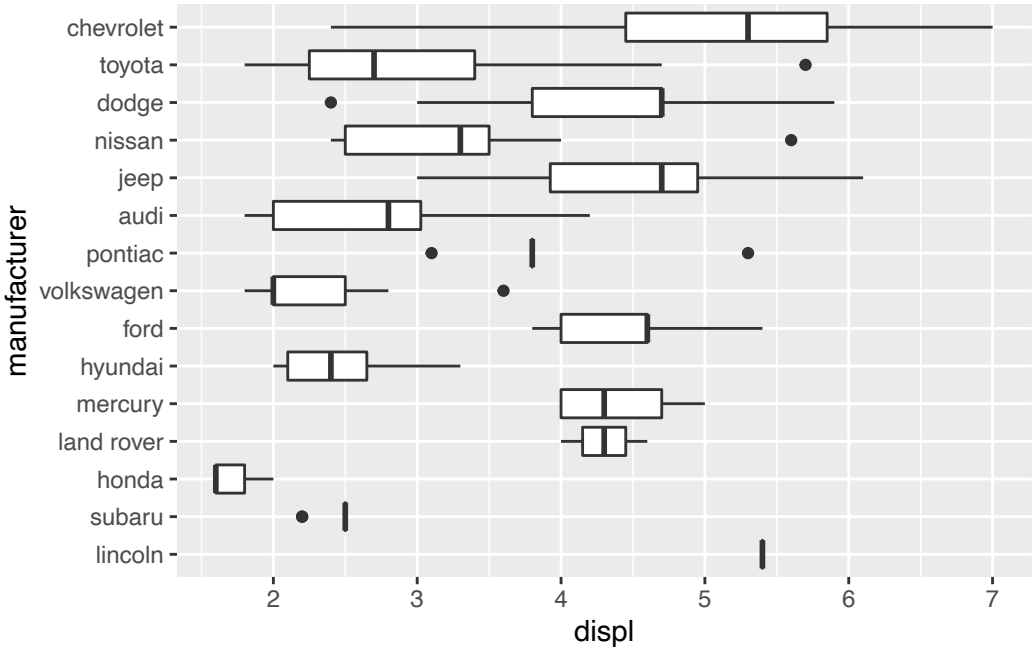
Or the minimal value:

```
mpg %>%
  mutate(
    manufacturer = fct_reorder(as.factor(manufacturer),
                               displ, .fun = min)
  ) %>%
  ggplot(aes(x = displ, y = manufacturer)) +
  geom_boxplot() +
  stat_summary(geom = "point", fun = "min", color = "blue") +
  stat_summary(geom = "line", fun = "min", color = "blue", group = 1)
```



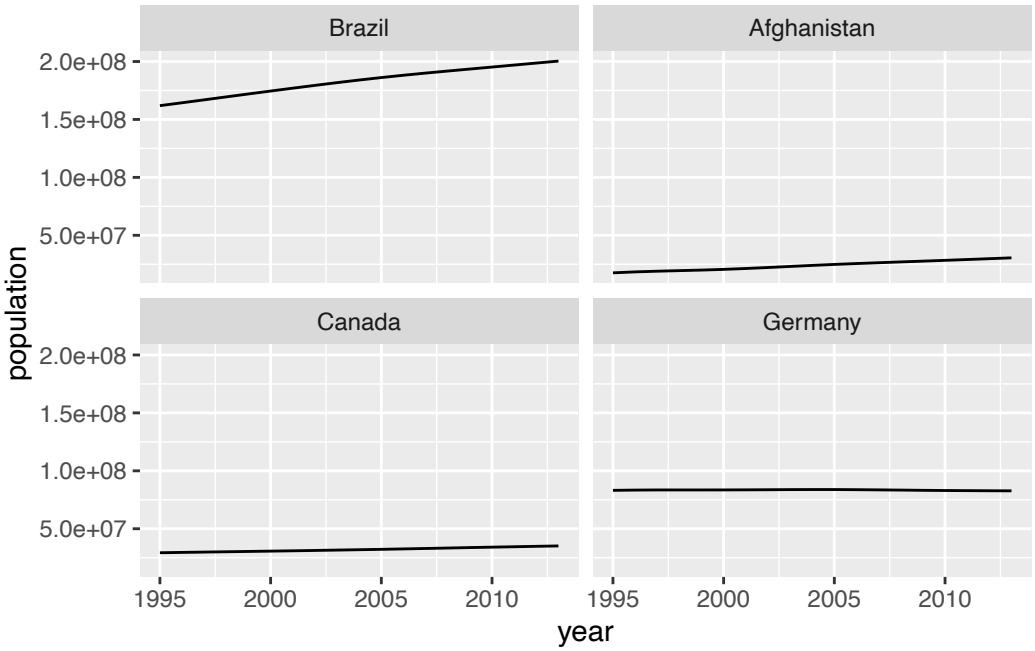
Similarly, you could order the levels by the range between the maximum and minimum values of each level:

```
mpg %>%
  mutate(
    manufacturer = fct_reorder(as.factor(manufacturer),
                               displ,
                               .fun = function(x) max(x) - min(x))
  ) %>%
  ggplot(aes(x = displ, y = manufacturer)) +
  geom_boxplot()
```



We could do something similar with facets. For example, suppose we want to rank countries by the amount of population growth from 1995 to 2013:

```
population %>%
  filter(country %in% c("Afghanistan", "Germany", "Brazil",
                        "Canada")) %>%
  mutate(country = fct_reorder(country,
                               population,
                               .fun = function(x) min(x) - max(x))
  ) %>%
  ggplot(aes(x = year, y = population)) +
  geom_line() +
  facet_wrap(vars(country))
```



Obviously, Brazil had the largest increase compared to the other countries.

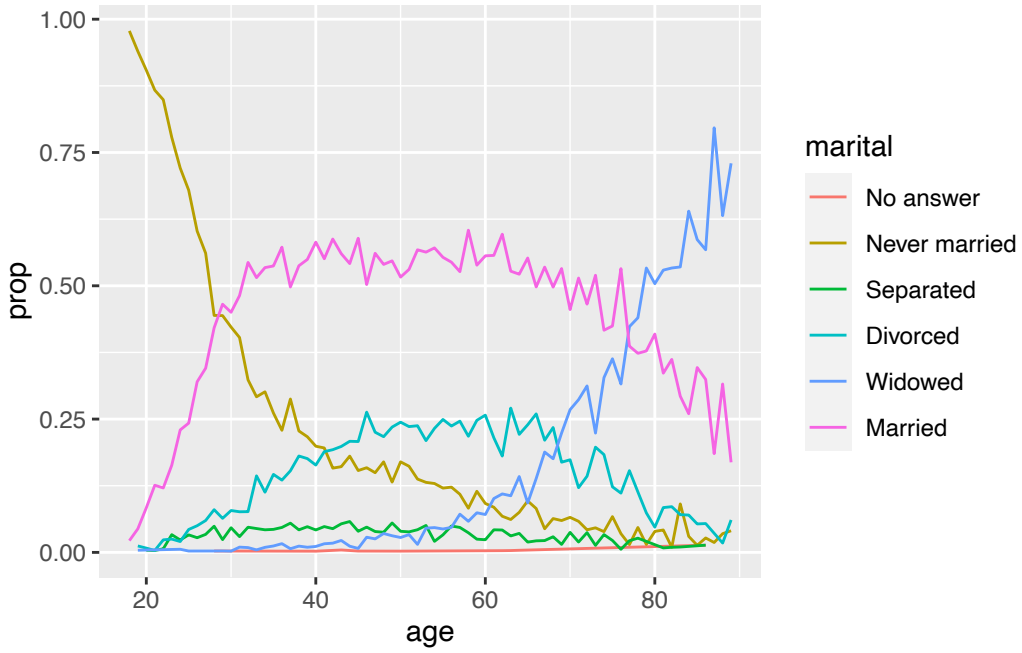
9.4 How to order levels based on the values of two numeric variables

Have you ever created a line chart and found that the end of your lines are not aligned with your legend text? Here is an example of this (which I adapted from this stackoverflow post):

```
marital_status_per_age <- gss_cat %>%
  count(age, marital) %>%
  group_by(age) %>%
  mutate(
    prop = n / sum(n)
  ) %>%
  ungroup()

marital_status_per_age %>%
  ggplot(aes(x = age, y = prop, color = marital)) +
  stat_summary(geom = "line", fun = mean)
```

```
Warning: Removed 6 rows containing non-finite values (stat_summary).
```



As you can see the blue line ends at the top but the first item in the legend is the "No answer" category. We can improve the order of the legend (aka the levels) by using the function `fct_reorder2`.

In essence, the function does the following: It finds the largest values of one variable at the largest value of another variable. In this case, we are looking for the largest value of `prop` within the largest value of `age`:

```
marital_status_per_age %>%
  group_by(marital) %>%
  slice_max(age) %>%
  ungroup() %>%
  arrange(desc(prop))
```

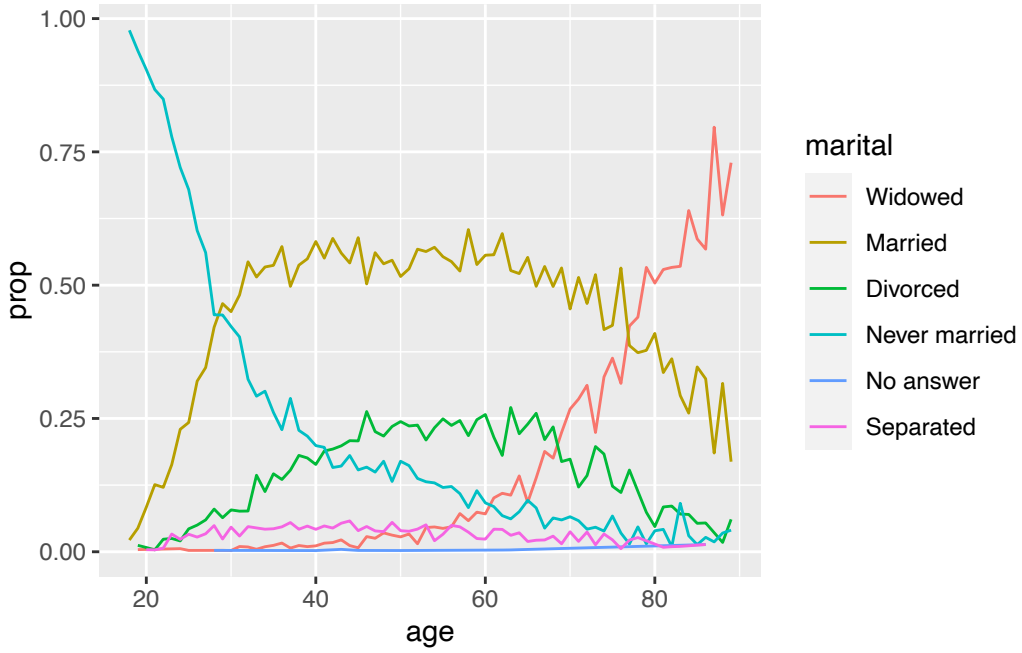
```
# A tibble: 6 x 4
    age marital           n   prop
  <int> <fct>         <int>  <dbl>
1    89 Widowed         108 0.730
2    89 Married          25 0.169
```

```
3    89 Divorced          9 0.0608
4    89 Never married     6 0.0405
5    86 No answer         1 0.0135
6    86 Separated         1 0.0135
```

For example, the largest value of `prop` for the largest value of `age` is 0.73. This level should, therefore, be at the top of our legend. Let's see how this works:

```
marital_status_per_age %>%
  mutate(
    marital = as.factor(marital) %>%
      fct_reorder2(age, prop)
  ) %>%
  ggplot(aes(x = age, y = prop, color = marital)) +
  stat_summary(geom = "line", fun = mean)
```

```
Warning: Removed 6 rows containing non-finite values (stat_summary).
```



The first column is the reference variable (i.e., `age`). The second column determines the order of the levels.

We can reverse the order by setting the `.fun` argument to `first2`:

```
marital_status_per_age %>%
  mutate(
    marital = as.factor(marital) %>%
      fct_reorder2(age, prop, .fun = first2)
  ) %>%
  ggplot(aes(x = age, y = prop, color = marital)) +
  stat_summary(geom = "line", fun = mean) +
  theme(
    legend.position = "left"
  )
```

```
Warning: Removed 6 rows containing non-finite values (stat_summary).
```

And this concludes our tour through ordering factor levels.

Summary

Here's what you can take away from this tutorial.

- The functions `fct_relevel` and `fct_infreq` order the levels only according to the

information provided by the factor column itself.

- The functions `fct_reorder` and `fct_reorder2` order levels based on other continuous variables.
- Use `fct_reorder2` if you want to align your legend text to the endpoints of your lines in a line chart.
- Use the `.fun` argument in `fct_reorder` whenever you want to order geometric objects that visualize many values (e.g. boxplots or violinplots)