Lecture 06: Variable Types in R

Taylor Arnold

Data Types

We have already discussed that variables in R have different *data types*, most likely one of the following:

- `int` stands for integers.
- `dbl` stands for doubles, or real numbers.
- `chr` stands for character vectors, or strings.

Example

An example with the mammals sleep data:

```
data(msleep)
msleep
```

```
## # A tibble: 83 x 11
##                          name      genus  vore         order
##                         <chr>      <chr> <chr>         <chr>
## 1                     Cheetah   Acinonyx carni     Carnivora
## 2                  Owl monkey      Aotus  omni      Primates
## 3           Mountain beaver Aplodontia herbi      Rodentia
## 4 Greater short-tailed shrew    Blarina  omni Soricomorpha
## 5                         Cow        Bos herbi Artiodactyla
## 6           Three-toed sloth   Bradypus herbi       Pilosa
## # ... with 77 more rows, and 7 more variables: conservation <chr>
## #   sleep_total <dbl>, sleep_rem <dbl>, sleep_cycle <dbl>,
## #   awake <dbl>, brainwt <dbl>, bodywt <dbl>
```

Example

The names and genus are characters whereas numbers such as `awake` are doubles. These slides shows how to convert numeric variables into categorical ones and way to manipulate categorical variables. There is generally no straightforward way to convert a categorical variable into a number, so we will not discuss that here.

Numeric to Categorical

With factor()

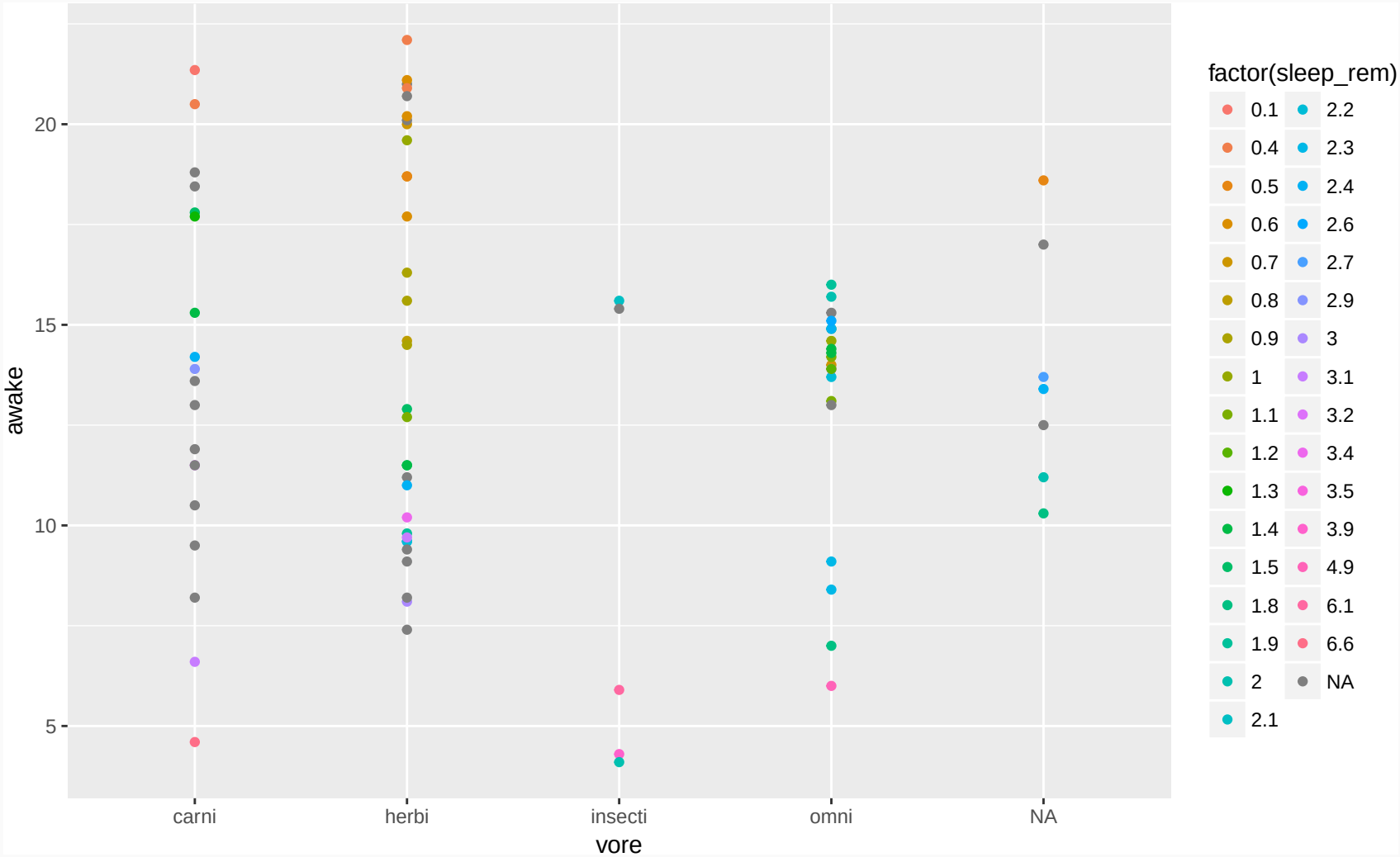
To convert every unique values of a numeric variable to a category in a new categorical variable, we simply use the function `factor`. For example, we can convert `sleep_rem` into categories. This changes, amongst other things, the way color is used in a plot:

```
qplot(vore, awake, data = msleep,
        color = factor(sleep_rem))
```

Notice that missing values became there own category. This is often very useful.

With factor()

```
qplot(vore, awake, data = msleep,
      color = factor(sleep_rem))
```



With cut() and bin()

Often we do not want each numeric variable to be its own category but instead wish to group values that are close into a single category. There are two related functions for doing this: `cut` and `bin`. Both take the variable followed by the number of bins to use. The function `cut` splits the range of the variable into equally spaced buckets where as `bin` breaks the range up so that each bin has (roughly) the same proportion of data points.

For most variables the difference between these two functions is small; I usually use `cut` because the interval cutoffs print out nicer. However, when a variable is fairly skewed, I find that `bin` works better.

With cut() and bin()

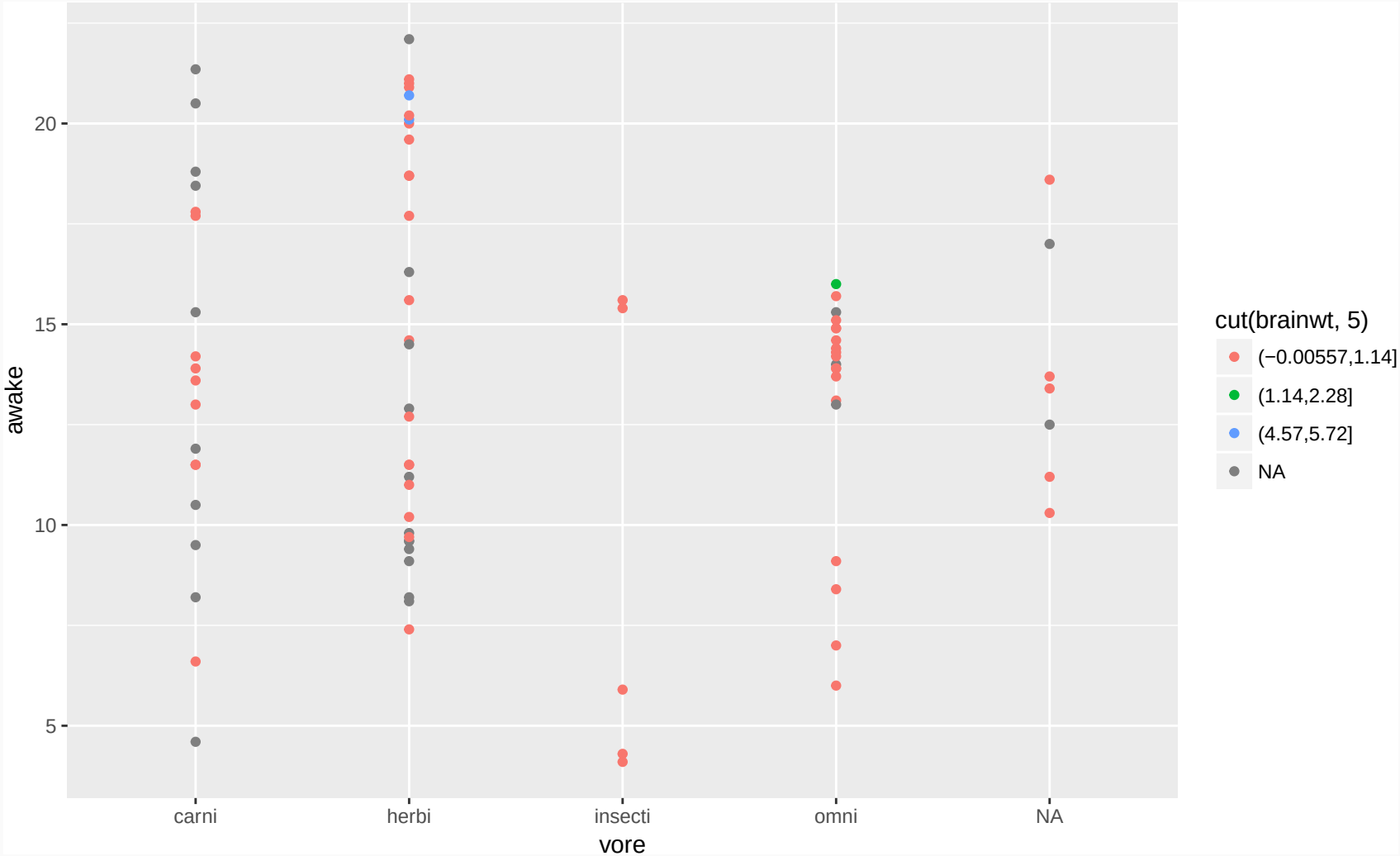
For example, notice how different cutting and binning the `brainwt` variable is:

```
qplot(vore, awake, data = msleep,
        color = cut(brainwt, 5))
qplot(vore, awake, data = msleep,
        color = bin(brainwt, 5))
```

This variable is very skewed and the cutting algorithm puts almost all of the mammals in the same category.

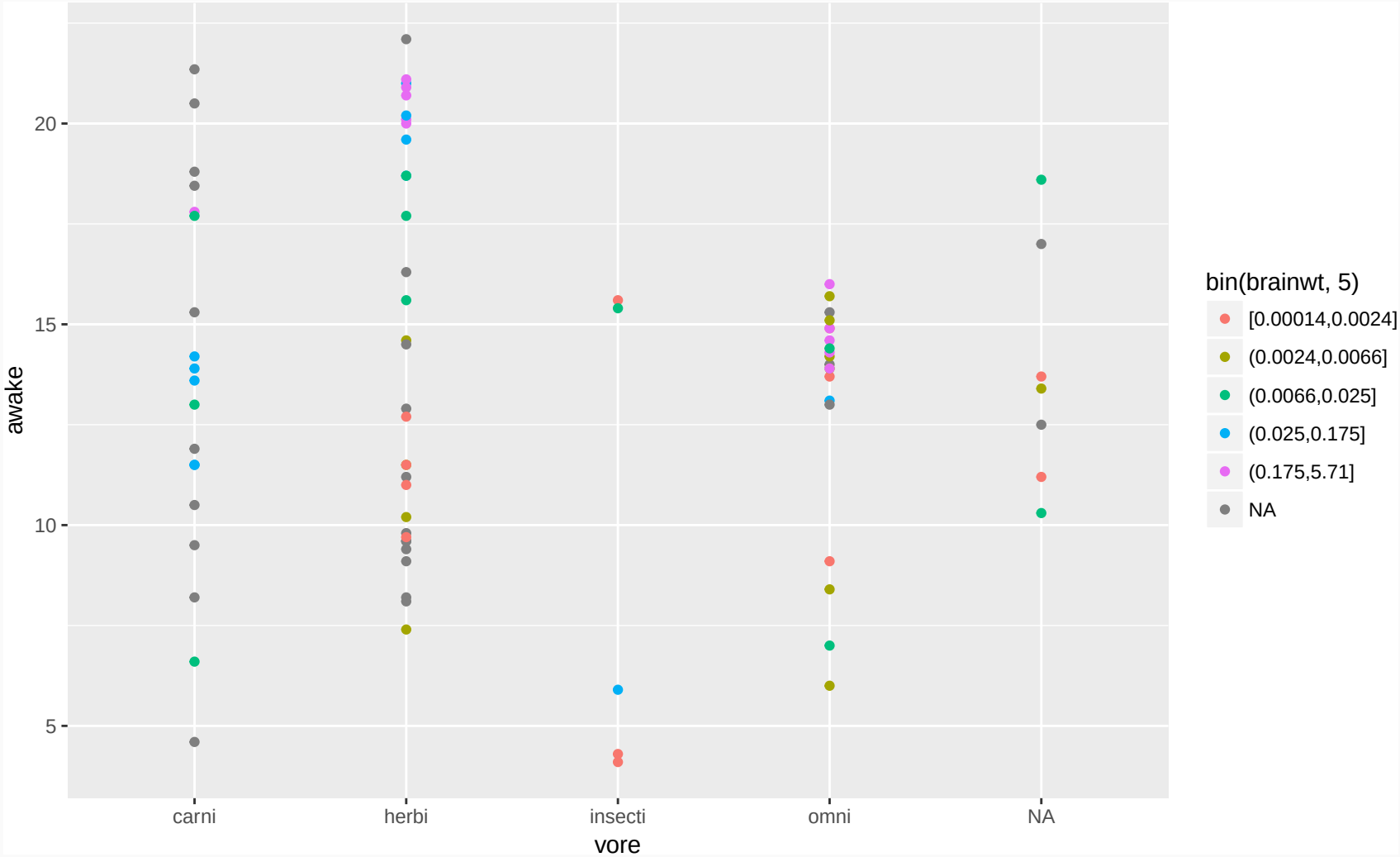
brainwt with cut()

```
qplot(vore, awake, data = msleep,
      color = cut(brainwt, 5))
```



brainwt with bin()

```
qplot(vore, awake, data = msleep,
      color = bin(brainwt, 5))
```



Manipulating Categorical Data

concept

Likewise, it is sometimes useful to convert one categorical variable into another categorical variable by grouping or ordering the categories in a different way. To work with these, we will use the **forcats** package:

```
library(forcats)
```

By default categories are order alphabetically.

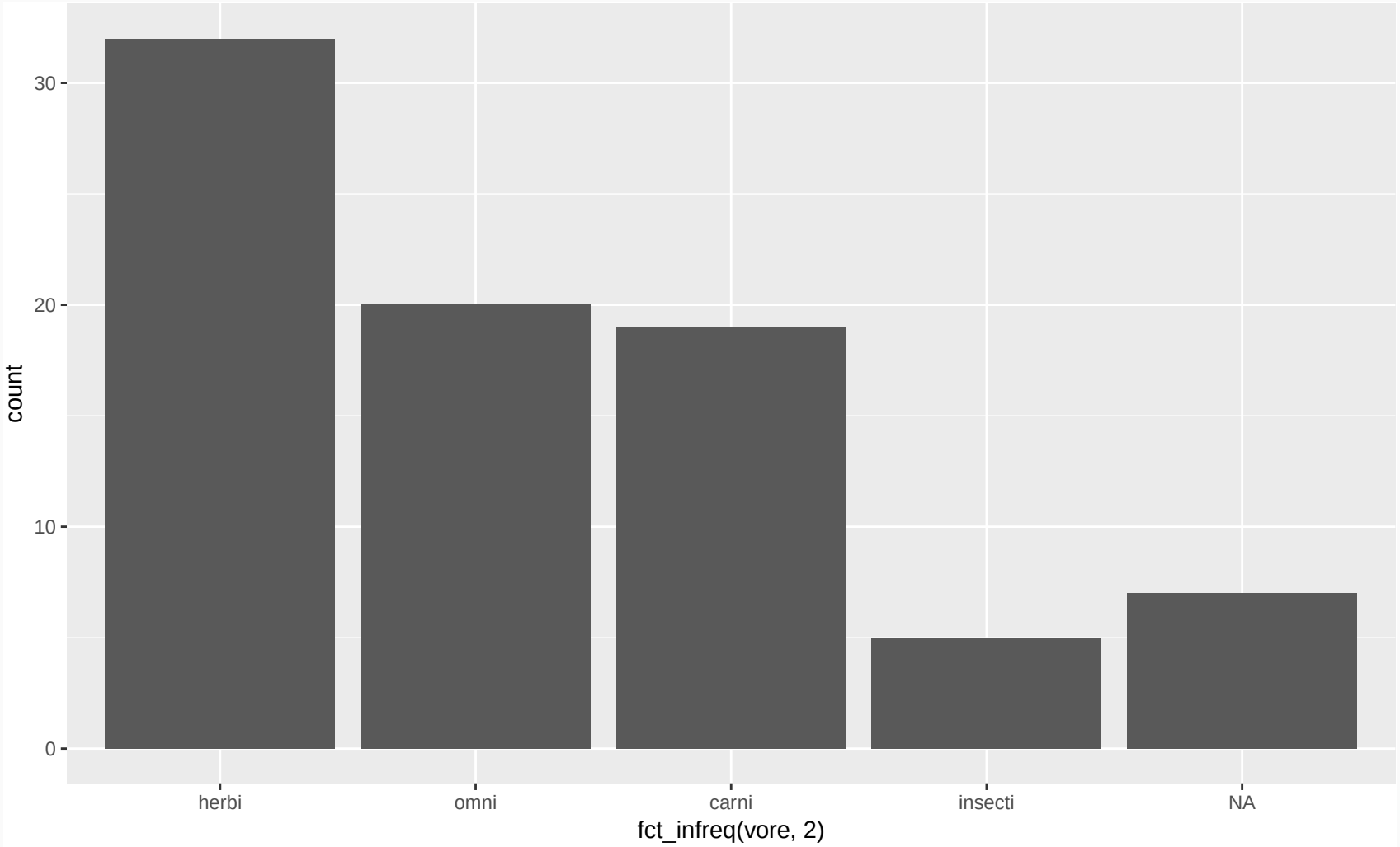
fct_infreq

The function `fct_infreq` order the categories from largest to smallest.

```
qplot(fct_infreq(vore), data = msleep)
```

fct_infreq

```
qplot(fct_infreq(vore, 2), data = msleep)
```



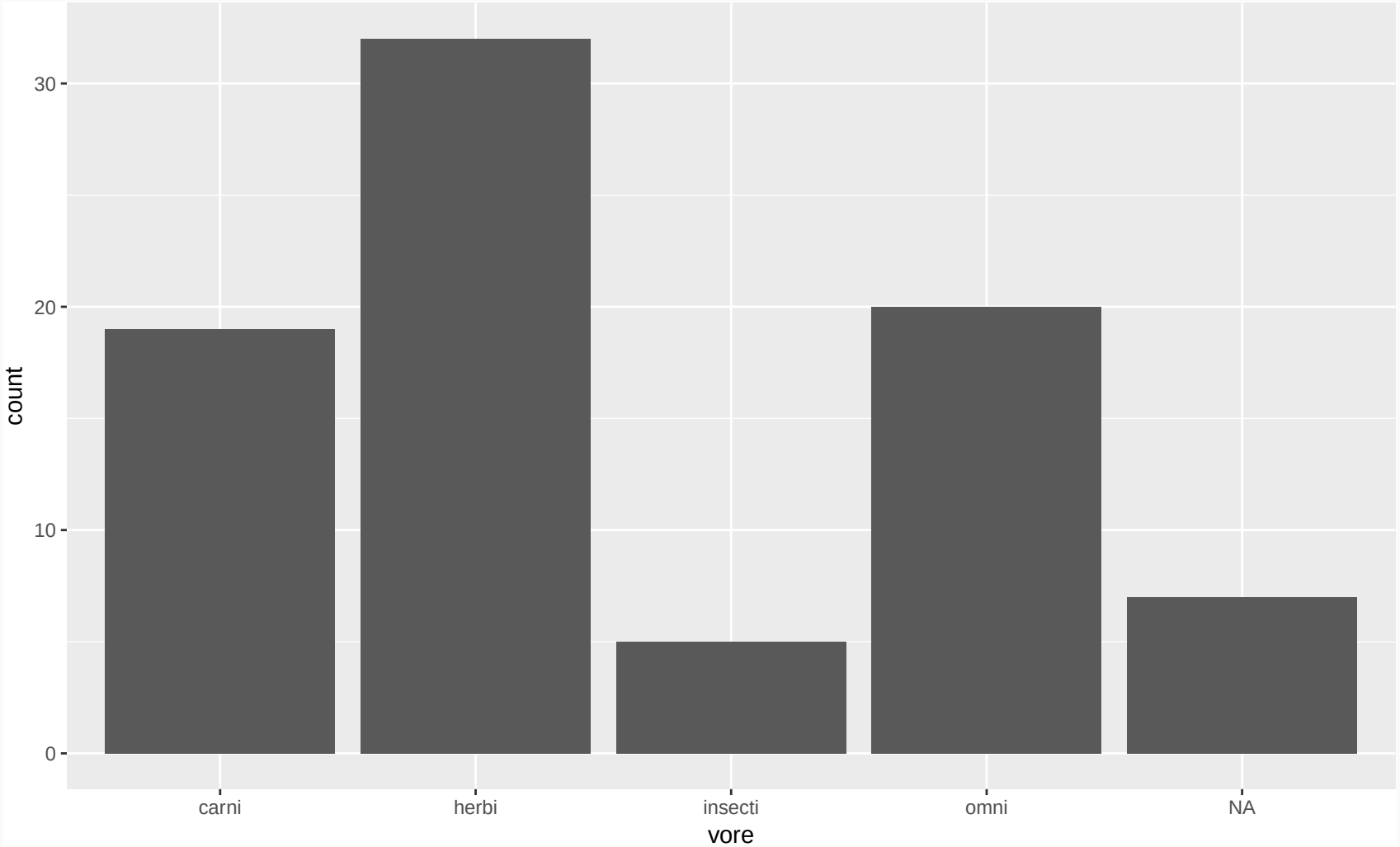
fct_lump

The function `fct_lump` take a second argument that gives the maximum number of categories allowed. The most frequent categories are included and all other categories are lumped into the `other` category.

```
qplot(vore, data = msleep)
qplot(fct_lump(vore, 2), data = msleep)
```

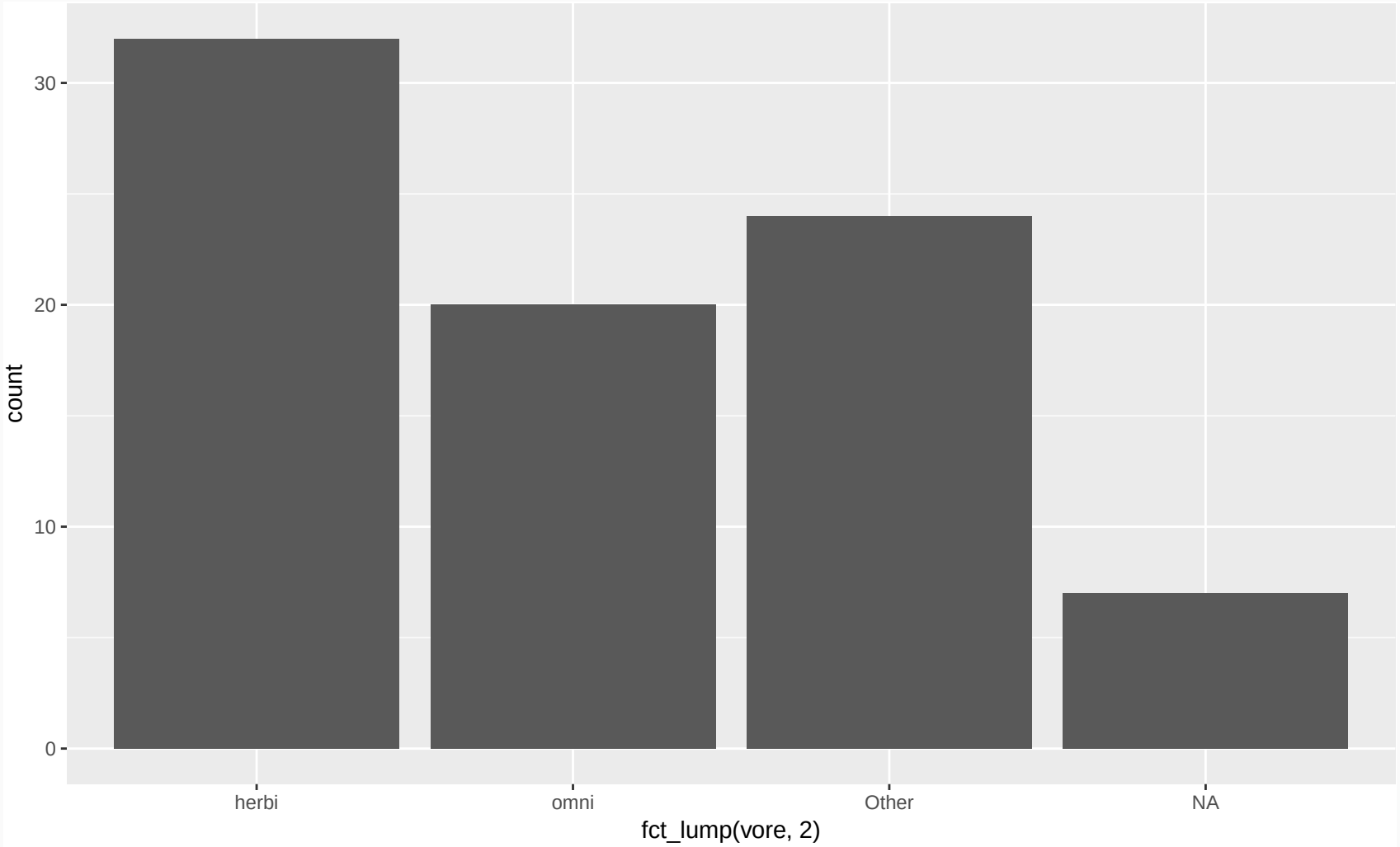
fct_lump

```
qplot(vore, data = msleep)
```



fct_lump

```
qplot(fct_lump(vore, 2), data = msleep)
```



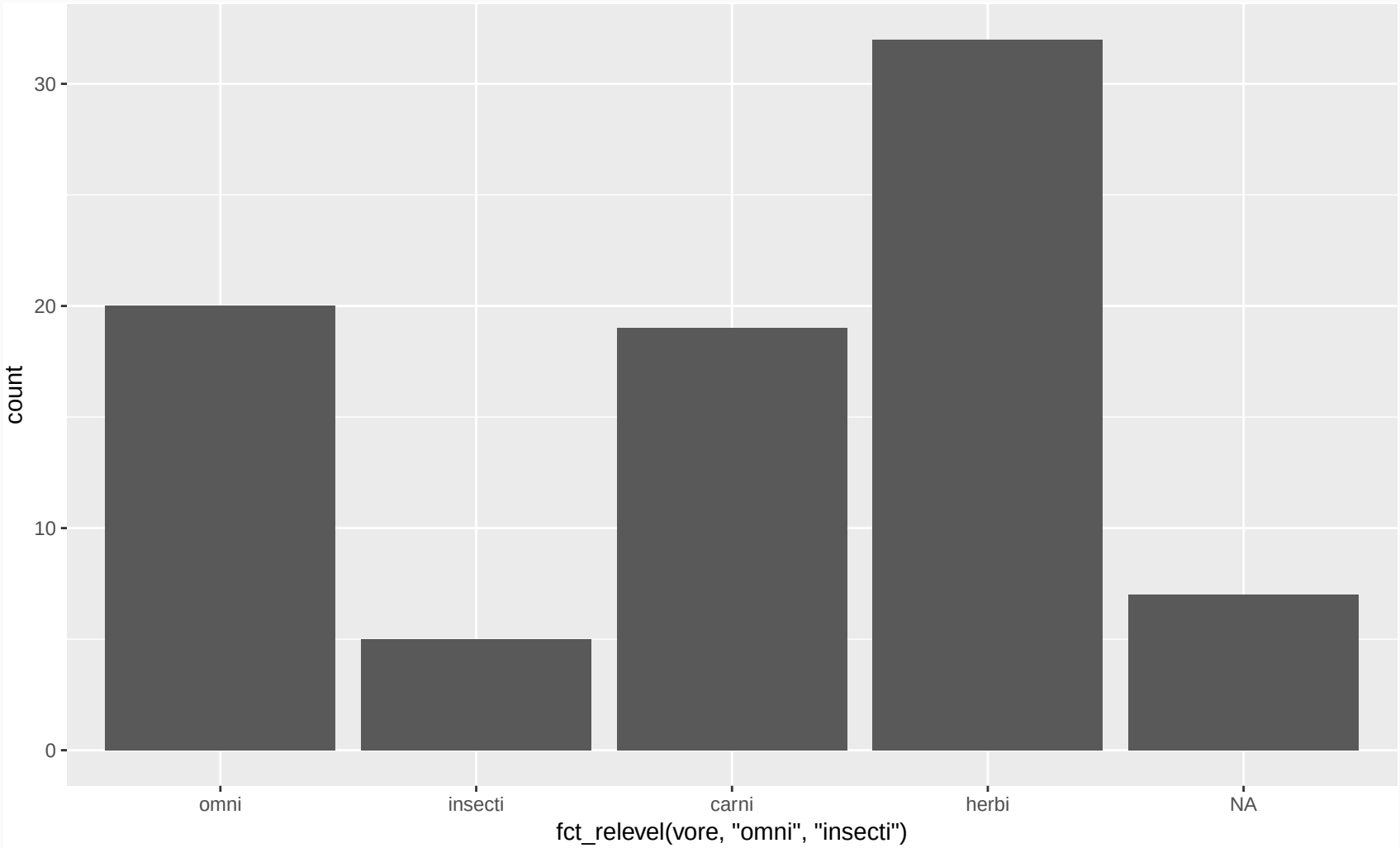
fct_relevel

Finally, the function `fct_relevel` takes multiple arguments that gives the categories that should be first, second, and so on. Unmentioned categories are left in their original order:

```
qplot(fct_relevel(vore, "omni", "insecti"),
        data = msleep)
```

fct_relevel

```
qplot(fct_relevel(vore, "omni", "insecti"),
      data = msleep)
```



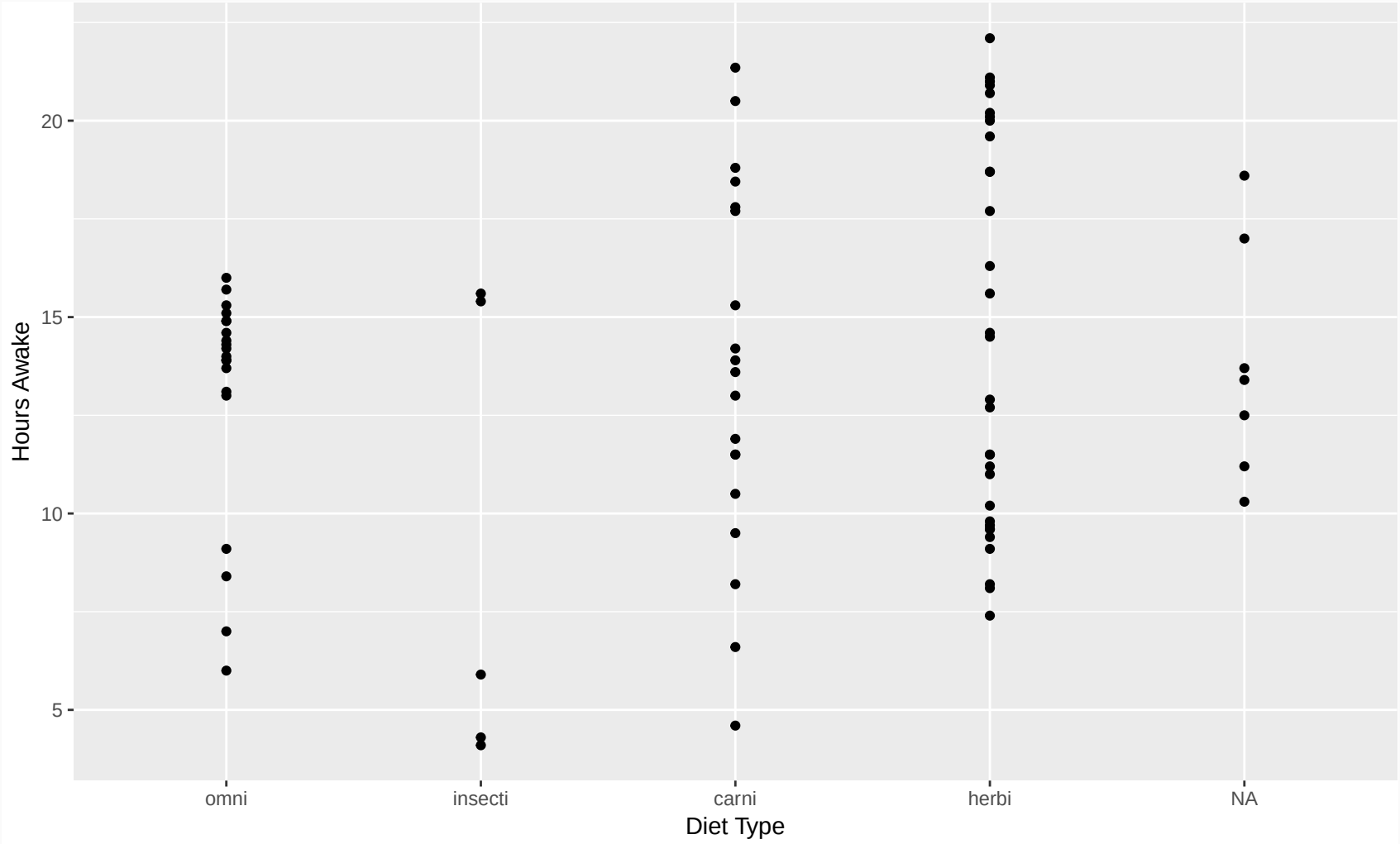
Plot Labels

Axes

All this messing with the input variables makes the default axis labels become quite messy. To modify these, simply add the functions `xlab()` and/or `ylab()` to the plot:

```
qplot(fct_relevel(vore, "omni", "insecti"), awake,
        data = msleep) + xlab("Diet Type") +
        ylab("Hours Awake")
```

Axes

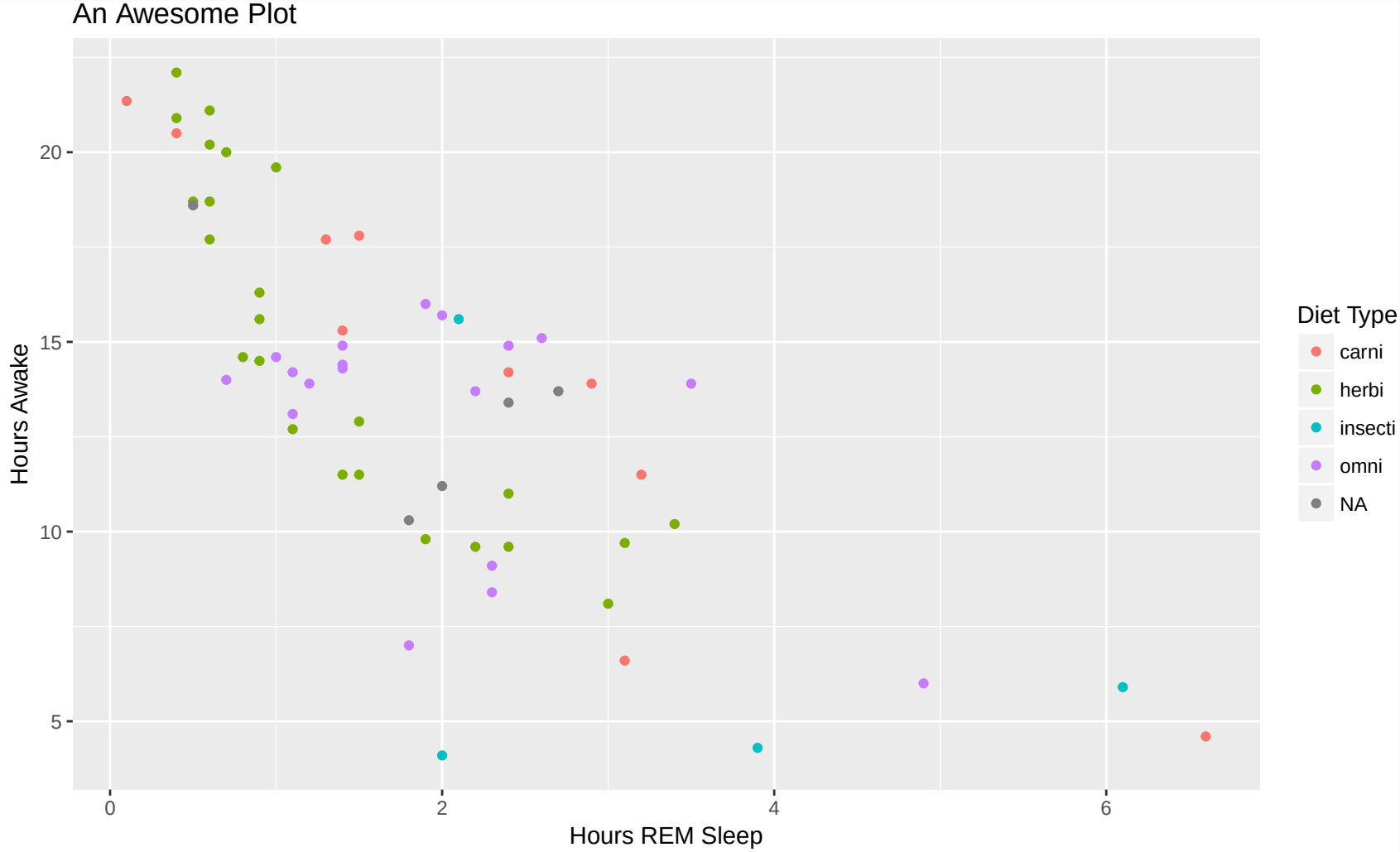


Titles

Similarly, titles are added with `ggtitle()` and labels for legends with the `labs()` function:

```
qplot(sleep_rem, awake, color = vore, data = msleep) +
    ggtitle("An Awesome Plot") +
    labs(color = "Diet Type") +
    xlab("Hours REM Sleep") +
    ylab("Hours Awake")
```

Titles


An Awesome Plot
Hours Awake
Hours REM Sleep
20
15
10
5
0
2
4
6
Diet Type
carni
herbi
insecti
omni
NA