Handout 05: Data Manipulation

Taylor Arnold

So far we have primarily been working with a single dataset as it was directly given to us. In many cases it is advantageous to produce new datasets from our original one. This can be as simple as selecting a subset of the original columns or rows, or as involved as changing the *level of analysis* of the entire dataset. This handout will introduce several functions called *verbs* that assist in manipulating datasets. These all come from the package **dplyr**, contained in **tidymodels**.

In this handout, as with the others so far, I will use the `msleep` dataset in order to show various numerical summaries. To read it in use the following:

```
data(msleep)
```

A grammar for data manipulation

The *verbs* in **dplyr** take either one or two existing datasets and return a new dataset. There are roughly 30 different verbs, of which we'll use only about 4 in this course:[1]

- `filter`
- `mutate`
- `group_summarize`
- `left_join`

[1] Actually, `group_summarize` is a verb that I wrote specifically for this course and comes from the **smodels** package. It combines several **dplyr** verbs in a particularly useful way.

These verbs on their own are relatively straightforward, but can be tied together to produce suprisingly complex new datasets.

Filtering rows

The filter function takes a dataset followed by a logical statements. It returns a dataset that has any rows in the input data that match the filtering statements. For example, the following returns a dataset for all mammals that are awake more than 20 hours per day:

```
filter(msleep, awake > 20)
```

```
## # A tibble: 9 × 11
##               name         genus  vore
##              <chr>         <chr> <chr>
## 1         Roe deer     Capreolus herbi
## 2   Asian elephant       Elephas herbi
## 3            Horse         Equus herbi
```

```
## 4           Donkey         Equus herbi
## 5          Giraffe       Giraffa herbi
## 6      Pilot whale Globicephalus carni
## 7 African elephant     Loxodonta herbi
## 8            Sheep          Ovis herbi
## 9     Caspian seal         Phoca carni
## # ... with 8 more variables: order <chr>,
## #   conservation <chr>, sleep_total <dbl>,
## #   sleep_rem <dbl>, sleep_cycle <dbl>,
## #   awake <dbl>, brainwt <dbl>, bodywt <dbl>
```

It is very important to notice that the original `msleep` dataset has not been altered here. If we want to actually work with the filtered data, we need to save it using the assignment operator `<-` and given it a name. To pull out just the *sleepy* mammals, we could do the following:

```
sleepy <- filter(msleep, awake < 6)
```

You should see now that a new dataset appears in your workspace named `sleepy`. It is possible to work with this new dataset in all of the ways we have plotted and (now) filtered the original data.

Constructing new variables

The `mutate` function preserves all rows of the original dataset, unlike `filter`, but adds a new variables. For example, to add hours asleep into the dataset we can do this:

```
msleep <- mutate(msleep, asleep = 24 - awake)
```

Here, I used the assignment command to save the result back into the `msleep` dataset. If you look in your environment window, you'll still see just a single version of `msleep` but this one will have one extra variables. This is a relatively safe pratice with the mutate function, as we are simply adding information, but should be generally avoided when using filter.[2]

[2] It is possible to use mutate to redefine an existing variable by giving mutate a variable name that already exists. Be careful of this, particularly if you are overwritting the original dataset.

Summarizing data

The `group_summarize` function is, in my opinion, the most complex verb that we will use this semester. If we use the function on a dataset without any other options it gives the mean, median, standard deviation, and sum for every numeric variable in the dataset. An overall count is also included. Let's apply it to `msleep`:

```
group_summarize(msleep)
```

```
## # A tibble: 1 × 29
##   sleep_total_mean sleep_rem_mean
##              <dbl>          <dbl>
## 1         10.43373             NA
## # ... with 27 more variables:
## #   sleep_cycle_mean <dbl>,
## #   awake_mean <dbl>, brainwt_mean <dbl>,
## #   bodywt_mean <dbl>, asleep_mean <dbl>,
## #   sleep_total_median <dbl>,
## #   sleep_rem_median <dbl>,
## #   sleep_cycle_median <dbl>,
## #   awake_median <dbl>,
## #   brainwt_median <dbl>,
## #   bodywt_median <dbl>,
## #   asleep_median <dbl>,
## #   sleep_total_sd <dbl>,
## #   sleep_rem_sd <dbl>,
## #   sleep_cycle_sd <dbl>, awake_sd <dbl>,
## #   brainwt_sd <dbl>, bodywt_sd <dbl>,
## #   asleep_sd <dbl>, sleep_total_sum <dbl>,
## #   sleep_rem_sum <dbl>,
## #   sleep_cycle_sum <dbl>, awake_sum <dbl>,
## #   brainwt_sum <dbl>, bodywt_sum <dbl>,
## #   asleep_sum <dbl>, n <int>
```

These variables could have easily been computed by calling the respective functions indvidually in R. The group summarize function becomes more interesting when we pass it a second input giving a variable to group by. For example, here is the summary *grouped by vore*

```
group_summarize(msleep, vore)
```

```
## # A tibble: 5 × 30
##      vore sleep_total_mean sleep_rem_mean
##     <chr>            <dbl>          <dbl>
## 1   carni        10.378947             NA
## 2   herbi         9.509375             NA
## 3 insecti        14.940000             NA
## 4    omni        10.925000             NA
## 5    <NA>        10.185714             NA
## # ... with 27 more variables:
## #   sleep_cycle_mean <dbl>,
## #   awake_mean <dbl>, brainwt_mean <dbl>,
## #   bodywt_mean <dbl>, asleep_mean <dbl>,
## #   sleep_total_median <dbl>,
```

```
## #   sleep_rem_median <lgl>,
## #   sleep_cycle_median <lgl>,
## #   awake_median <dbl>,
## #   brainwt_median <dbl>,
## #   bodywt_median <dbl>,
## #   asleep_median <dbl>,
## #   sleep_total_sd <dbl>,
## #   sleep_rem_sd <dbl>,
## #   sleep_cycle_sd <dbl>, awake_sd <dbl>,
## #   brainwt_sd <dbl>, bodywt_sd <dbl>,
## #   asleep_sd <dbl>, sleep_total_sum <dbl>,
## #   sleep_rem_sum <dbl>,
## #   sleep_cycle_sum <dbl>, awake_sum <dbl>,
## #   brainwt_sum <dbl>, bodywt_sum <dbl>,
## #   asleep_sum <dbl>, n <int>
```

Notice that the result now provides these summaries for each group. It is possible to summarize by multiple groups at once, which produces summaries for each unique combination of the those variables. For instance, we could summarize by both genus and vore:

```
group_summarize(msleep, vore, genus)
```

```
## # A tibble: 77 × 31
##     vore         genus sleep_total_mean
##    <chr>         <chr>            <dbl>
## 1  carni      Acinonyx             12.1
## 2  carni   Callorhinus              8.7
## 3  carni         Canis             10.1
## 4  carni       Dasypus             17.4
## 5  carni         Felis             12.5
## 6  carni       Genetta              6.3
## 7  carni Globicephalus              2.7
## 8  carni  Haliochoerus              6.2
## 9  carni    Lutreolina             19.4
## 10 carni     Nyctibeus             11.0
## # ... with 67 more rows, and 28 more
## #   variables: sleep_rem_mean <dbl>,
## #   sleep_cycle_mean <dbl>,
## #   awake_mean <dbl>, brainwt_mean <dbl>,
## #   bodywt_mean <dbl>, asleep_mean <dbl>,
## #   sleep_total_median <dbl>,
## #   sleep_rem_median <dbl>,
## #   sleep_cycle_median <dbl>,
## #   awake_median <dbl>,
```

```
## #   brainwt_median <dbl>,
## #   bodywt_median <dbl>,
## #   asleep_median <dbl>,
## #   sleep_total_sd <dbl>,
## #   sleep_rem_sd <dbl>,
## #   sleep_cycle_sd <dbl>, awake_sd <dbl>,
## #   brainwt_sd <dbl>, bodywt_sd <dbl>,
## #   asleep_sd <dbl>, sleep_total_sum <dbl>,
## #   sleep_rem_sum <dbl>,
## #   sleep_cycle_sum <dbl>, awake_sum <dbl>,
## #   brainwt_sum <dbl>, bodywt_sum <dbl>,
## #   asleep_sum <dbl>, n <int>
```

And there is now a row for each combination of `vore` and `genus`.

Combine datasets

The final verb that we will use this semester is also the only two-table verb that we will need. It will be used to combine a dataset with metadata about one or more of its variables. To illustrate, let's make a small dataframe that contains the full name for the short-hand abbreviations given in the variable `vore`:

```
meta <- data_frame(vore = c("carni", "omni", "herbi", "insecti"),
                   full_name = c("carnivore", "omnivore", "herbivore",
                                 "insectivore"))
meta
```

```
## # A tibble: 4 × 2
##      vore   full_name
##     <chr>       <chr>
## 1   carni   carnivore
## 2    omni    omnivore
## 3   herbi   herbivore
## 4 insecti insectivore
```

To combine these with the original dataset, we use the `left_join` function, giving the larger dataset first:

```
msleep <- left_join(msleep, meta)
msleep
```

```
## # A tibble: 83 × 13
##                          name       genus
##                         <chr>       <chr>
## 1                     Cheetah    Acinonyx
```

```
## 2                  Owl monkey       Aotus
## 3             Mountain beaver  Aplodontia
## 4  Greater short-tailed shrew     Blarina
## 5                          Cow         Bos
## 6            Three-toed sloth    Bradypus
## 7           Northern fur seal Callorhinus
## 8                Vesper mouse     Calomys
## 9                          Dog       Canis
## 10                   Roe deer   Capreolus
## # ... with 73 more rows, and 11 more
## #   variables: vore <chr>, order <chr>,
## #   conservation <chr>, sleep_total <dbl>,
## #   sleep_rem <dbl>, sleep_cycle <dbl>,
## #   awake <dbl>, brainwt <dbl>,
## #   bodywt <dbl>, asleep <dbl>,
## #   full_name <chr>
```

Notice that there is a new variable `full_name` that now displays the full name for each `vore` type.