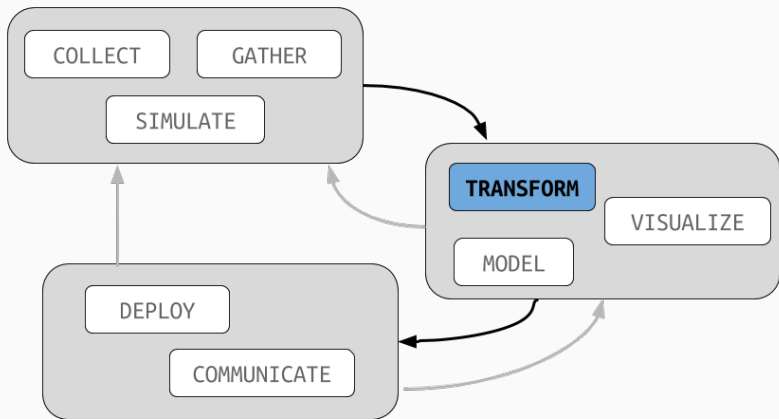


# Lecture 07: Data Manipulation

---

Taylor Arnold



# Data manipulation

- ▶ 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.
- ▶ 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.
- ▶ we will introduce several functions called *verbs* that assist in manipulating datasets. These all come from the package **dplyr**, supplemented with functions in **models**.

In this handout, as with the others so far, I will use the `msleep` dataset in order to show various numerical summaries. Rather than the entire dataset, it will be helpful to take just a subset of the columns. This gives us the first verb we'll see: `select`:

```
msleep <- select(msleep, awake, vore, genus)
```

# A grammar for data manipulation

---

## 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)
```

## Filtering rows

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

```
## # A tibble: 9 x 3
##   awake vore      genus
##   <dbl> <chr>    <chr>
## 1 21.00 herbi  Capreolus
## 2 20.10 herbi  Elephas
## 3 21.10 herbi  Equus
## 4 20.90 herbi  Equus
## 5 22.10 herbi  Giraffa
## 6 21.35 carni Globicephalus
## # ... with 3 more rows
```

## Filtering rows

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.

## Constructing new variables

If you look in your environment window, you'll still see just a single version of `msleep` but this one will have one extra variable. This is a relatively safe practice with the `mutate` function, as we are simply adding information, but should be generally avoided when using `filter`.

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 overwriting the original dataset.

## Constructing new variables

Of course, we could also do the same thing with the following:

```
msleep$asleep <- 24 - msleep$awake
```

It is just nice to use the `mutate` function as it is consistent with all of the other **dplyr** verbs.

## 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)
```

## Summarizing data

```
group_summarize(msleep)
```

```
## # A tibble: 1 x 9
##   awake_mean asleep_mean awake_median asleep_median
##   <dbl>         <dbl>         <dbl>         <dbl>
## 1   13.56747    10.43253          13.9          10.1
## # ... with 5 more variables: awake_sd <dbl>,
## #   asleep_sd <dbl>, awake_sum <dbl>, asleep_sum <dbl>,
## #   n <int>
```

## Summarizing data

These variables could have easily been computed by calling the respective functions individually 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)
```

## Summarizing data

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

```
## # A tibble: 5 x 10
```

```
##   vore awake_mean asleep_mean awake_median asleep_median
```

```
##   <chr>      <dbl>      <dbl>      <dbl>      <dbl>
```

```
## 1  carni    13.62632    10.373684     13.6      10.4
```

```
## 2  herbi    14.49062     9.509375     13.7      10.3
```

```
## 3 insecti   9.06000    14.940000      5.9      18.1
```

```
## 4  omni    13.07500    10.925000     14.1       9.9
```

```
## 5   <NA>    13.81429    10.185714     13.4      10.6
```

```
## # ... with 5 more variables: awake_sd <dbl>,
```

```
## #   asleep_sd <dbl>, awake_sum <dbl>, asleep_sum <dbl>,
```

```
## #     n <int>
```



## Summarizing data

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)
```

Which returns a row for each unique combination of vore and genus.

## Summarizing data

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

```
## # A tibble: 77 x 11
```

```
##   vore      genus awake_mean asleep_mean awake_median
```

```
##   <chr>      <chr>      <dbl>      <dbl>      <dbl>
```

```
## 1 carni    Acinonyx      11.9       12.1       11.9
```

```
## 2 carni Callorhinus    15.3       8.7       15.3
```

```
## 3 carni     Canis      13.9      10.1      13.9
```

```
## 4 carni   Dasypus       6.6      17.4       6.6
```

```
## 5 carni     Felis      11.5      12.5      11.5
```

```
## 6 carni   Genetta      17.7       6.3      17.7
```

```
## # ... with 71 more rows, and 6 more variables:
```

```
## #   asleep_median <dbl>, awake_sd <dbl>, asleep_sd <dbl>,
```

```
## #   awake_sum <dbl>, asleep_sum <dbl>, n <int>
```

## Counting data

The summarized data also includes a column called `n` giving the total number of observations within a group. The `count` function functions similarly, but does not returns any of the other summary functions. This is convenient if all you need are counts:

```
count(msleep, vore)
```

## Counting data

```
count(msleep, vore)
```

```
## # A tibble: 5 x 2
##       vore      n
##   <chr> <int>
## 1  carní    19
## 2  herbi    32
## 3 insecti    5
## 4   omni    20
## 5   <NA>     7
```

# Counting data

The `sort` option gives us the ability to order to data at the same time as it is counted:

```
count(msleep, vore, sort = TRUE)
```

## Counting data

```
count(msleep, vore, sort = TRUE)
```

```
## # A tibble: 5 x 2
##       vore      n
##   <chr> <int>
## 1 herbi    32
## 2 omni     20
## 3 carni    19
## 4 <NA>      7
## 5 insecti   5
```

## Combining 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 data frame 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 x 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)
```

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

```
msleep
```

```
## # A tibble: 83 x 5
##   awake vore      genus asleep full_name
##   <dbl> <chr>    <chr>  <dbl>    <chr>
## 1  11.9 carni  Acinonyx  12.1 carnivore
## 2   7.0 omni    Aotus    17.0 omnivore
## 3   9.6 herbi Aplodontia  14.4 herbivore
## 4   9.1 omni    Blarina  14.9 omnivore
## 5  20.0 herbi      Bos     4.0 herbivore
## 6   9.6 herbi  Bradypus  14.4 herbivore
## # ... with 77 more rows
```