# Herding cats with dplyr

*UA R Users Group*

*May 14, 2019*

## Contents



---

## dplyr

80% of the work involved with data analysis involves cleaning and shaping the data until it's in the state you need. Bracket subsetting is handy, but it can be cumbersome and difficult to read, especially for complicated operations. Enter `dplyr`!

`dplyr` is a package for making data manipulation easier. (It does a lot more too, but this is what we'll focus on).

Unlike the subsetting commands we've already worked on, `dplyr` is designed to be highly expressive, and highly readable. It's structured around a set of verbs, or grammar of data manipulation. The core functions we'll talk about are below:

- `select`
- `arrange`
- `filter`
- `group_by`
- `mutate`
- `summarise/summarize`

### Getting the package

```
install.packages("dplyr")
```

Again, you may be asked to choose a CRAN mirror; RStudio is a good choice.

Unlike the `rmarkdown` package, we'll be using `dplyr` in the console. In order to do that, we need to *load* the package into our environment so we can access functions from `dplyr`. We do this with the `library` command:

```
library("dplyr")
```

You only need to install a package once per computer, but you need to load it every time you open a new R session and want to use that package.

## Subsetting Data

The first two `dplyr` commands we'll use help us to subset our data by rows and columns.

### `select`

The first command we'll use is `select`, which allows us to choose columns from our dataset. Let's use our `cats` dataset and select only the `coat` column; we did this previously with

```
cats[, "coat"]
```

```
##  [1] tabby   maltese brown   black   calico  tabby   brown   brown
##  [9] black   maltese
## Levels: black brown calico maltese tabby
```

With dplyr, we don't need to enclose our column names in quotes

```
select(cats, coat)
```

```
##       coat
## 1    tabby
## 2  maltese
## 3    brown
## 4    black
## 5   calico
## 6    tabby
## 7    brown
## 8    brown
## 9    black
## 10 maltese
```

Notice how the output differs slightly; all the main `dplyr` verbs behave consistently in that their inputs and outputs are both `data.frame`s, rather than returning a simple vector as the bracket-indexing method did. All of the main "verbs" we'll talk about will return a data.frame as their result.

We can select more columns by giving `select` additional arguments, and our output `data.frame` will have columns according to the order of our arguments

```
select(cats, coat, cat_id)
```

```
##       coat cat_id
## 1    tabby    321
## 2  maltese    250
## 3    brown    219
## 4    black    182
## 5   calico    107
## 6    tabby    234
## 7    brown    196
## 8    brown    311
## 9    black    130
## 10 maltese    349
```

### filter

So where `select` allowed us to select *columns*, `filter` operated on *rows*. Say we want to see the all the cats with black coats; we saw earlier how to use that using bracket-indexing:

```
cats[cats$coat == "black", ]
```

```
##             street  coat    sex   age weight fixed wander_dist roamer
## 4    140 Robin Way black female 7.172  8.053     1       0.030     no
## 9 16982 Kennedy Rd black female 2.851  3.291     0       0.065     no
##   cat_id
## 4    182
## 9    130
```

In dplyr, this looks like

```
filter(cats, coat == "black")
```

```
##             street  coat    sex   age weight fixed wander_dist roamer
## 1    140 Robin Way black female 7.172  8.053     1       0.030     no
## 2 16982 Kennedy Rd black female 2.851  3.291     0       0.065     no
##   cat_id
## 1    182
## 2    130
```

Notice we don't have to use the `$` operator to tell `filter` where the `coat` column is; it's smart enough to assume we want the `coat` column from the `data.frame` we passed in.

### arrange

Maybe you have a set of observations in your data that you want to organize by their value. `arrange` allows us to change the order of rows in our dataset based on their values.

```r
arrange(cats, coat)
```

```
##                 street    coat    sex   age weight fixed wander_dist
## 1        140 Robin Way   black female 7.172  8.053     1        0.030
## 2     16982 Kennedy Rd   black female 2.851  3.291     0        0.065
## 3    201 Hollywood Ave   brown female 4.601  3.947     1        0.076
## 4  115 Via Santa Maria   brown   male 6.917  5.626     1        0.097
## 5      303 Harding Ave   brown   male 3.713  3.982     1        0.033
## 6       135 Charles St  calico   male 4.660  6.193     1        0.085
## 7      242 Harding Ave maltese female 8.234 12.368     1        0.033
## 8   16528 Marchmont Dr maltese female 4.594  6.994     0        0.059
## 9       Los Robles Way   tabby female 3.003  3.993     0        0.040
## 10 130 Vista Del Campo   tabby female 3.796  3.860     1        0.085
##    roamer cat_id
## 1      no    182
## 2      no    130
## 3      no    219
## 4     yes    196
## 5      no    311
## 6     yes    107
## 7      no    250
## 8      no    349
## 9      no    321
## 10     no    234
```

```r
# you can include additional columns to help sort the data
arrange(cats, coat, sex)
```

```
##                 street    coat    sex   age weight fixed wander_dist
## 1        140 Robin Way   black female 7.172  8.053     1        0.030
## 2     16982 Kennedy Rd   black female 2.851  3.291     0        0.065
## 3    201 Hollywood Ave   brown female 4.601  3.947     1        0.076
## 4  115 Via Santa Maria   brown   male 6.917  5.626     1        0.097
## 5      303 Harding Ave   brown   male 3.713  3.982     1        0.033
## 6       135 Charles St  calico   male 4.660  6.193     1        0.085
## 7      242 Harding Ave maltese female 8.234 12.368     1        0.033
## 8   16528 Marchmont Dr maltese female 4.594  6.994     0        0.059
## 9       Los Robles Way   tabby female 3.003  3.993     0        0.040
## 10 130 Vista Del Campo   tabby female 3.796  3.860     1        0.085
##    roamer cat_id
## 1      no    182
## 2      no    130
## 3      no    219
## 4     yes    196
## 5      no    311
```

```
## 6     yes    107
## 7      no    250
## 8      no    349
## 9      no    321
## 10     no    234
```

**mutate**

One common task in working with data is updating/cleaning some of the values in columns. `mutate` allows us to do this relatively easily. Let's say I don't want a lot of decimal places in one of my measurements. I can use `mutate` to update my existing variable:

```
mutate(cats, weight = round(weight, 2))
```

```
##                 street    coat    sex   age weight fixed wander_dist
## 1       Los Robles Way   tabby female 3.003   3.99     0       0.040
## 2      242 Harding Ave maltese female 8.234  12.37     1       0.033
## 3    201 Hollywood Ave   brown female 4.601   3.95     1       0.076
## 4        140 Robin Way   black female 7.172   8.05     1       0.030
## 5       135 Charles St  calico   male 4.660   6.19     1       0.085
## 6  130 Vista Del Campo   tabby female 3.796   3.86     1       0.085
## 7  115 Via Santa Maria   brown   male 6.917   5.63     1       0.097
## 8      303 Harding Ave   brown   male 3.713   3.98     1       0.033
## 9     16982 Kennedy Rd   black female 2.851   3.29     0       0.065
## 10  16528 Marchmont Dr maltese female 4.594   6.99     0       0.059
##    roamer cat_id
## 1      no    321
## 2      no    250
## 3      no    219
## 4      no    182
## 5     yes    107
## 6      no    234
## 7     yes    196
## 8      no    311
## 9      no    130
## 10     no    349
```

Another common task is generating a new column based on values that are already in the dataset you are working on. `mutate` helps us do this, and tacks a new column to the end of our data.frame.

```
# let's say you want to add two variables together
mutate(cats, new_variable = age + weight)
```

```
##                 street    coat    sex   age weight fixed wander_dist
## 1       Los Robles Way   tabby female 3.003  3.993     0       0.040
## 2      242 Harding Ave maltese female 8.234 12.368     1       0.033
```

```
## 3    201 Hollywood Ave   brown female 4.601  3.947     1       0.076
## 4        140 Robin Way   black female 7.172  8.053     1       0.030
## 5       135 Charles St  calico   male 4.660  6.193     1       0.085
## 6  130 Vista Del Campo   tabby female 3.796  3.860     1       0.085
## 7  115 Via Santa Maria   brown   male 6.917  5.626     1       0.097
## 8      303 Harding Ave   brown   male 3.713  3.982     1       0.033
## 9     16982 Kennedy Rd   black female 2.851  3.291     0       0.065
## 10  16528 Marchmont Dr maltese female 4.594  6.994     0       0.059
##    roamer cat_id new_variable
## 1      no    321        6.996
## 2      no    250       20.602
## 3      no    219        8.548
## 4      no    182       15.225
## 5     yes    107       10.853
## 6      no    234        7.656
## 7     yes    196       12.543
## 8      no    311        7.695
## 9      no    130        6.142
## 10     no    349       11.588
```

```
# you can include as many new variables as you want, separated by a comma
mutate(cats, new_var_1 = age + weight, new_var_2 = age * weight)
```

```
##                 street    coat    sex   age weight fixed wander_dist
## 1       Los Robles Way   tabby female 3.003  3.993     0       0.040
## 2      242 Harding Ave maltese female 8.234 12.368     1       0.033
## 3    201 Hollywood Ave   brown female 4.601  3.947     1       0.076
## 4        140 Robin Way   black female 7.172  8.053     1       0.030
## 5       135 Charles St  calico   male 4.660  6.193     1       0.085
## 6  130 Vista Del Campo   tabby female 3.796  3.860     1       0.085
## 7  115 Via Santa Maria   brown   male 6.917  5.626     1       0.097
## 8      303 Harding Ave   brown   male 3.713  3.982     1       0.033
## 9     16982 Kennedy Rd   black female 2.851  3.291     0       0.065
## 10  16528 Marchmont Dr maltese female 4.594  6.994     0       0.059
##    roamer cat_id new_var_1  new_var_2
## 1      no    321     6.996  11.990979
## 2      no    250    20.602 101.838112
## 3      no    219     8.548  18.160147
## 4      no    182    15.225  57.756116
## 5     yes    107    10.853  28.859380
## 6      no    234     7.656  14.652560
## 7     yes    196    12.543  38.915042
## 8      no    311     7.695  14.785166
## 9      no    130     6.142   9.382641
## 10     no    349    11.588  32.130436
```

## Pipes

You'll often find yourself needing to use multiple functions in a row to organize some data that you're working on. This can sometimes lead to dense code that is difficult to read.

```
# for example
sort(round(sqrt(cats$age * 2), 3))
```

```
##  [1] 2.388 2.451 2.725 2.755 3.031 3.033 3.053 3.719 3.787 4.058
```

In the code above, I have multiple steps to get my result, but you have to read what's going on from the inside out. This can be cumbersome, especially if you need to understand how one function's output influences the next operation.

### Using Pipes

`dplyr` includes a special operator designed to make code *flow* and appear more readable.

It's written as `%>%`, and you can call it the "pipe" operator.

Our example above can be re-written as:

```
cats$age * 2 %>%
  sqrt() %>%
  round(3) %>%
  sort()
```

```
##  [1]  4.246242 11.642876  6.505814 10.141208  6.589240  5.367544  9.780638
##  [8]  5.250182  4.031314  6.495916
```

Instead of being nested within a bunch of commands, you can see read the code as a series of statements: 1. Multiply `cats$age` by 2, *then* 2. Take the square-root of these values, *then* 3. Round the result to the 3rd digit, *then* 4. Sort the values in ascending order

I encourage you to think of the `%>%` as short-hand for "then", when reading code that uses it!

"Pipe" operators are found in other languages; they get their name from the idea that your code can be thought of as a "pipeline".

Let's look at another example.

```
round(1.23456789, 3)
```

```
## [1] 1.235
```

We can use a pipe operator to acheive the same thing.

```
1.23456789 %>% round(3)
```

```
## [1] 1.235
```

The pipe takes care of making sure the output of the expression on the left-hand-side (a simple numeric, in this case) is inserted as the first argument of the expressing on the right-hand-side.

We can also pipe into other argument positions by using a period as a placeholder.

```
3 %>% round(1.23456789, .)
```

```
## [1] 1.235
```

These are contrived examples, and I don't suggest using pipes for simple operations like rounding. The pipes really become useful when chaining together multiple operations in sequence, as we'll do with our `dplyr` functions.

## Combining Select & Filter

The pipe is really helpful when combined with the data-manipulation of `dplyr`. Remember how we used `filter` to select only the black cats? What if we only want to see the ID's of those cats, rather than all the info about them? We've already seen we can use `select` to pick out certain columns. We can use that to select the `cat_id` column from our `filter`ed dataset like so

```
# reading from the inside out
select(filter(cats, coat == "black"), cat_id)
```

```
##   cat_id
## 1    182
## 2    130
```

That might not look too bad now, but what if we wanted to do another operation on that output? We'd add another layer of nesting, and having to read that line from the inside-out can quickly become annoying. We can use the pipe operator to clean that up.

```
# reading from left to right
filter(cats, coat == "black") %>% select(cat_id)
```

```
##   cat_id
## 1    182
## 2    130
```

We could even add another pipe to feed `cats` into `filter`; it isn't necessary, but it makes it even easier to see what we're operating on in this chain of commands. We'll combine this with some line breaks to really make this easy to read:

```
cats %>%
  filter(coat == "black") %>%
  select(cat_id)
```

```
##   cat_id
## 1    182
## 2    130
```

### summarize

While `mutate` creates new columns, it's often useful to summarize multiple rows into a single value. Say we want to find the mean weight of all these cats; enter `summarize`! Like `mutate`, the arguments to summarize (after the `data.frame` we want to operate on) are expressions. We can combine `summarize` with the `mean` function to get a mean weight for our collection of cats like so:

```
cats %>% summarize(mean_weight = mean(weight))
```

```
##   mean_weight
## 1      5.8307
```

Notice how we have only a single value returned, but it's still in a `data.frame` format. This is subtle, but important; all these basic `dplyr` verbs take in `data.frame`s and also return `data.frame`s. This consistency helps make long chains of `dplyr` operations possible.

### group_by

A very common data analysis task is to do operations like we did above, but to do them on a group-by-group basis. To do this with `dplyr`, we'll use the `group_by` function.

Let's look at the mean weights of our cats, grouping up by coat. This will give us the mean weight of the black cats, mean weight of the calico cats, etc. We can do this by inserting a `group_by` function into our earlier expression for computing mean weight:

```
cats %>%
  group_by(coat) %>%
  summarize(mean_weight = mean(weight))
```

```
## # A tibble: 5 x 2
##   coat    mean_weight
##   <fct>         <dbl>
## 1 black          5.67
## 2 brown          4.52
## 3 calico         6.19
## 4 maltese        9.68
## 5 tabby          3.93
```

Ta-da!

We can also use `mutate` on a per-group basis. Let's make a new column which centers our weights around zero; this can be done by subtracting the group's mean weight from each cat's weight:

```
cats %>%
  group_by(coat) %>%
  mutate(centered_weight = weight - mean(weight))
```

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

```
## # Groups:   coat [5]
##    street coat  sex     age weight fixed wander_dist roamer cat_id
##    <fct>  <fct> <fct> <dbl>  <dbl> <int>       <dbl> <fct>   <int>
##  1 Los R~ tabby fema~  3.00   3.99     0       0.04  no        321
##  2 242 H~ malt~ fema~  8.23  12.4      1       0.033 no        250
##  3 201 H~ brown fema~  4.60   3.95     1       0.076 no        219
##  4 140 R~ black fema~  7.17   8.05     1       0.03  no        182
##  5 135 C~ cali~ male   4.66   6.19     1       0.085 yes       107
##  6 130 V~ tabby fema~  3.80   3.86     1       0.085 no        234
##  7 115 V~ brown male   6.92   5.63     1       0.097 yes       196
##  8 303 H~ brown male   3.71   3.98     1       0.033 no        311
##  9 16982~ black fema~  2.85   3.29     0       0.065 no        130
## 10 16528~ malt~ fema~  4.59   6.99     0       0.059 no        349
## # ... with 1 more variable: centered_weight <dbl>
```

# Going further

This is an introductory look at `dplyr`, just enough to make you dangerous. As you continue your R journey I suggest looking into the other awesome things you can do with this package!