Data manipulation using dplyr

Dave Millar

dave.millar@uwyo.edu

R for Ecologists

11/15/17

This R notebook contains a VERY brief overview of **dplyr**, using several examples with sample data. The five primary functions for **dplyr** include:

1. **mutate()**
2. **select()**
3. **filter()**
4. **summarize()**
5. **arrange()**

First, let's clear everything out of memory, call dplyr and create a data frame full of example data to work with. Note that the **gather()** function from tidyr is used create the long format version of the data frame.

```
rm(list=ls())

library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)

var1 <- rnorm(100,5.3,1.2)
var2 <- rnorm(100,6.1,1.7)
var3 <- rnorm(100,9.5,1.4)
var4 <- rnorm(100,1.2,0.5)
var5 <- rnorm(100,8.5,1.2)

# wide format
cat_data <- cbind.data.frame(var1, var2, var3, var4, var5)

# long format
cat_data_gg <- cat_data %>% gather("var1", "var2", "var3", "var4", "var5",
                                   key = "variable", value = "value")
```

1. Mutate

The is function allows you create a new variable (column within a data frame or tibble) as a function of existing variables within the same data frame.

```
cat_data_mut <- cat_data %>% mutate(
  newCol = var1 * var3 + 5 / var5
)

head(cat_data_mut)
```

```
##       var1     var2      var3      var4      var5   newCol
## 1 7.372450 5.303861 11.380109 0.9072361 10.301038 84.38467
## 2 7.906802 6.955676 10.644481 0.6713332  8.800822 84.73193
## 3 4.662313 6.552846  8.745033 2.3995936  8.606519 41.35304
## 4 5.106382 5.179034 10.829731 1.3312174  7.790877 55.94252
## 5 5.530915 6.440444  8.859528 1.7964493  8.091930 49.61920
## 6 4.619204 8.751095 11.316290 1.2165730  8.147556 52.88593
```

2. Select

The is function allows you to 'select' specific variables within a data frame or tibble.

```
cat_data_sel <- select(cat_data, var2:var4)
head(cat_data_sel)
```

```
##       var2      var3      var4
## 1 5.303861 11.380109 0.9072361
## 2 6.955676 10.644481 0.6713332
## 3 6.552846  8.745033 2.3995936
## 4 5.179034 10.829731 1.3312174
## 5 6.440444  8.859528 1.7964493
## 6 8.751095 11.316290 1.2165730
```

3. Filter

The is function allows you to 'filter' out specific rows of a data frame based on variable values. Below is a simple example using arithmetic operators.

```
cat_data_filt <- filter(cat_data, var1 > 3 & var5 <= 7.5)
head(cat_data_filt)
```

```
##       var1     var2      var3      var4     var5
## 1 5.516575 5.315791  5.731391 1.2284772 7.242523
## 2 4.151058 6.845226 10.488892 1.0621613 7.396103
## 3 4.109315 4.820092  8.873219 0.6578831 6.669857
## 4 6.354532 8.271579  9.129220 1.4220195 7.401424
## 5 6.160718 6.379978  8.506927 0.8643888 7.223538
## 6 6.349512 6.796242  6.605058 0.6208120 6.666399
```

4. Summarize

The is function allows you to summarize variables in a data frame using various functions. Here, we'll summarize each variable by calculating its mean and standard deviation. Note that the summary statistics

match those used to create each vector of data for each variable using **rnorm()** above.

```
cat_data_summarized <- cat_data_gg %>%
               group_by(variable) %>%
               summarize(mean = mean(value),sd = sd(value))
cat_data_summarized
```

```
## # A tibble: 5 x 3
##   variable     mean        sd
##      <chr>    <dbl>     <dbl>
## 1     var1 5.361234 1.1865436
## 2     var2 6.143396 1.3308338
## 3     var3 9.533878 1.4884773
## 4     var4 1.257298 0.4950592
## 5     var5 8.626354 1.1480597
```

5. Arrange

The is function allows you to arrange (or rearrange) data in a data frame based on a user-defined condition. Here let's just rearrange the rows in **cat_data** so that the values for **var3** are in ascending order, then again in descending order.

```
cd_var3_asc <- arrange(cat_data, var3)
head(cd_var3_asc)
```

```
##       var1     var2     var3      var4      var5
## 1 5.516575 5.315791 5.731391 1.2284772  7.242523
## 2 6.692652 8.547499 6.221199 1.2294192  8.290528
## 3 5.753244 4.505669 6.477495 1.1847041  8.086052
## 4 6.349512 6.796242 6.605058 0.6208120  6.666399
## 5 4.577141 4.864652 6.801107 0.1431171  9.697820
## 6 4.750319 6.073066 7.388587 1.2082650 10.421952
```

```
cd_var3_desc <- arrange(cat_data, desc(var3))
head(cd_var3_desc)
```

```
##       var1     var2     var3      var4      var5
## 1 3.909182 6.322508 12.46571 1.2678527  9.684912
## 2 7.458066 4.789463 12.40116 1.7959900  8.151913
## 3 4.525869 5.721359 12.28436 1.0073642  8.443806
## 4 5.857802 7.388104 12.15557 1.1742794 10.232968
## 5 5.041876 7.575533 12.11332 0.9245203  9.960598
## 6 3.252482 4.138323 12.10519 1.0561521  9.046338
```

Hopefully this very brief overview was helpful in better understanding some of the new tidyverse methods for data manipulation in R. At the very least, it hopefully provides an efficient alternative for doing this type of data *wrangling* in Excel. Lastly, for more examples and detailed info, definitely check out the dplyr link mentioned above.