dplyr and pipes: the basics

Sean C. Anderson, sean@seananderson.ca

September 16, 2014

Contents

Introduction

The dplyr R package is awesome. Pipes from the magrittr R package are awesome. Put the two together and you have one of the most exciting things to happen to R in a long time.

dplyr is Hadley Wickham's re-imagined plyr package (with underlying C++ secret sauce co-written by Romain Francois). plyr 2.0 if you will. It does less than dplyr, but what it does it does more elegantly and much more quickly.

dplyr is built around 5 verbs. These verbs make up the majority of the data manipulation you tend to do. You might need to:

Select certain columns of data.

Filter your data to select specific rows.

Arrange the rows of your data into an order.

Mutate your data frame to contain new columns.

Summarise chunks of you data in some way.

Let's look at how those work.

The data

We're going to work with a dataset of mammal life-history, geography, and ecology traits from the PanTHERIA database:

Jones, K.E., *et al.* PanTHERIA: a species-level database of life history, ecology, and geography of extant and recently extinct mammals. Ecology 90:2648. http://esapubs.org/archive/ecol/E090/184/

First we'll download the data:

```r
pantheria <-
  "http://esapubs.org/archive/ecol/E090/184/PanTHERIA_1-0_WR05_Aug2008.txt"
download.file(pantheria, destfile = "mammals.txt")
```

Next we'll read it in and simplify it. This gets a bit ugly, but you can safely just run this code chunk and ignore the details:

```r
mammals <- read.table("mammals.txt", sep = "\t", header = TRUE,
  stringsAsFactors = FALSE)
names(mammals) <- sub("X[0-9._]+", "", names(mammals))
names(mammals) <- sub("MSW05_", "", names(mammals))
mammals <- dplyr::select(mammals, Order, Binomial, AdultBodyMass_g,
  AdultHeadBodyLen_mm, HomeRange_km2, LitterSize)
names(mammals) <- gsub("([A-Z])", "_\\L\\1", names(mammals), perl = TRUE)
names(mammals) <- gsub("^_", "", names(mammals), perl = TRUE)
mammals[mammals == -999] <- NA
names(mammals)[names(mammals) == "binomial"] <- "species"
mammals <- dplyr::tbl_df(mammals) # for prettier printing
```

Next we'll load the dplyr package:

```r
library(dplyr)
```

Looking at the data

Data frames look a bit different in dplyr. Above, I called the `tbl_df()` function on our data. This provides more useful printing of data frames in the console. Ever accidentally printed a massive data frame in the console before? Yeah… this avoids that. You don't need to change your data to a data frame tbl first — the dplyr functions will automatically convert your data when you call them. This is what the data look like on the console:

```r
mammals
```

```
## Source: local data frame [5,416 x 6]
##
```

```
##           order             species adult_body_mass_g
## 1  Artiodactyla Camelus dromedarius            492714
## 2     Carnivora       Canis adustus             10392
## 3     Carnivora        Canis aureus              9659
## 4     Carnivora       Canis latrans             11989
## 5     Carnivora         Canis lupus             31757
## 6  Artiodactyla       Bos frontalis            800143
## 7  Artiodactyla       Bos grunniens            500000
## 8  Artiodactyla       Bos javanicus            635974
## 9      Primates  Callicebus cupreus              1117
## 10     Primates Callicebus discolor                NA
## ..          ...                 ...               ...
## Variables not shown: adult_head_body_len_mm (dbl), home_range_km2 (dbl),
##   litter_size (dbl)
```

dplyr also provides a function `glimpse()` that makes it easy to look at our data in a transposed view. It's similar to the `str()` (structure) function, but has a few advantages (see `?glimpse`).

```
glimpse(mammals)
```

```
## Variables:
## $ order                  (chr) "Artiodactyla", "Carnivora", "Carnivora...
## $ species                (chr) "Camelus dromedarius", "Canis adustus",...
## $ adult_body_mass_g      (dbl) 492714.5, 10392.5, 9658.7, 11989.1, 317...
## $ adult_head_body_len_mm (dbl) NA, 745.3, 827.5, 872.4, 1055.0, 2700.0...
## $ home_range_km2         (dbl) 196.32000, 1.01000, 2.95000, 18.88000, ...
## $ litter_size            (dbl) 0.98, 4.50, 3.74, 5.72, 4.98, 1.22, 1.0...
```

Selecting columns

`select()` lets you subset by columns. This is similar to `subset()` in base R, but it also allows for some fancy use of helper functions such as `contains()`, `starts_with()` and, `ends_with()`. I think these examples are self explanatory, so I'll just include them here:

```
select(mammals, adult_head_body_len_mm)
```

```
## Source: local data frame [5,416 x 1]
##
##    adult_head_body_len_mm
## 1                      NA
## 2                   745.3
## 3                   827.5
## 4                   872.4
## 5                  1055.0
## 6                  2700.0
## 7                      NA
## 8                  2075.0
## 9                   355.0
## 10                     NA
## ..                    ...
```

```r
select(mammals, adult_head_body_len_mm, litter_size)
```

```
## Source: local data frame [5,416 x 2]
##
##    adult_head_body_len_mm litter_size
## 1                      NA        0.98
## 2                   745.3        4.50
## 3                   827.5        3.74
## 4                   872.4        5.72
## 5                  1055.0        4.98
## 6                  2700.0        1.22
## 7                      NA        1.00
## 8                  2075.0        1.22
## 9                   355.0        1.01
## 10                     NA          NA
## ..                    ...         ...
```

```r
select(mammals, adult_head_body_len_mm:litter_size)
```

```
## Source: local data frame [5,416 x 3]
##
##    adult_head_body_len_mm home_range_km2 litter_size
## 1                      NA         196.32        0.98
## 2                   745.3           1.01        4.50
## 3                   827.5           2.95        3.74
## 4                   872.4          18.88        5.72
## 5                  1055.0         159.86        4.98
## 6                  2700.0             NA        1.22
## 7                      NA             NA        1.00
## 8                  2075.0             NA        1.22
## 9                   355.0             NA        1.01
## 10                     NA             NA          NA
## ..                    ...            ...         ...
```

```r
select(mammals, -adult_head_body_len_mm)
```

```
## Source: local data frame [5,416 x 5]
##
##           order             species adult_body_mass_g home_range_km2
## 1  Artiodactyla Camelus dromedarius            492714         196.32
## 2     Carnivora       Canis adustus             10392           1.01
## 3     Carnivora        Canis aureus              9659           2.95
## 4     Carnivora       Canis latrans             11989          18.88
## 5     Carnivora         Canis lupus             31757         159.86
## 6  Artiodactyla       Bos frontalis            800143             NA
## 7  Artiodactyla       Bos grunniens            500000             NA
## 8  Artiodactyla       Bos javanicus            635974             NA
## 9      Primates  Callicebus cupreus              1117             NA
## 10     Primates Callicebus discolor                NA             NA
## ..          ...                 ...               ...            ...
## Variables not shown: litter_size (dbl)
```

```r
select(mammals, contains("body"))
```

```
## Source: local data frame [5,416 x 2]
##
##    adult_body_mass_g adult_head_body_len_mm
## 1             492714                     NA
## 2              10392                  745.3
## 3               9659                  827.5
## 4              11989                  872.4
## 5              31757                 1055.0
## 6             800143                 2700.0
## 7             500000                     NA
## 8             635974                 2075.0
## 9               1117                  355.0
## 10                NA                     NA
## ..               ...                    ...
```

```r
select(mammals, starts_with("adult"))
```

```
## Source: local data frame [5,416 x 2]
##
##    adult_body_mass_g adult_head_body_len_mm
## 1             492714                     NA
## 2              10392                  745.3
## 3               9659                  827.5
## 4              11989                  872.4
## 5              31757                 1055.0
## 6             800143                 2700.0
## 7             500000                     NA
## 8             635974                 2075.0
## 9               1117                  355.0
## 10                NA                     NA
## ..               ...                    ...
```

```r
select(mammals, ends_with("g"))
```

```
## Source: local data frame [5,416 x 1]
##
##    adult_body_mass_g
## 1             492714
## 2              10392
## 3               9659
## 4              11989
## 5              31757
## 6             800143
## 7             500000
## 8             635974
## 9               1117
## 10                NA
## ..               ...
```

```r
select(mammals, 1:3)
```

```
## Source: local data frame [5,416 x 3]
##
##           order              species adult_body_mass_g
## 1  Artiodactyla Camelus dromedarius            492714
## 2     Carnivora       Canis adustus             10392
## 3     Carnivora        Canis aureus              9659
## 4     Carnivora       Canis latrans             11989
## 5     Carnivora         Canis lupus             31757
## 6  Artiodactyla       Bos frontalis            800143
## 7  Artiodactyla       Bos grunniens            500000
## 8  Artiodactyla       Bos javanicus            635974
## 9      Primates   Callicebus cupreus             1117
## 10     Primates Callicebus discolor               NA
## ..          ...                 ...               ...
```

Filtering rows

`filter()` lets you subset by rows. You can use any valid logical statements:

```r
filter(mammals, adult_body_mass_g > 1e7)[ , 1:3]
```

```
## Source: local data frame [12 x 3]
##
##      order                 species adult_body_mass_g
## 1  Cetacea      Caperea marginata          32000000
## 2  Cetacea  Balaenoptera musculus         154321304
## 3  Cetacea  Balaenoptera physalus          47506008
## 4  Cetacea     Balaena mysticetus          79691179
## 5  Cetacea  Balaenoptera borealis          22106252
## 6  Cetacea     Balaenoptera edeni          20000000
## 7  Cetacea      Berardius bairdii          11380000
## 8  Cetacea  Eschrichtius robustus          27324024
## 9  Cetacea    Eubalaena australis          23000000
## 10 Cetacea    Eubalaena glacialis          23000000
## 11 Cetacea Megaptera novaeangliae          30000000
## 12 Cetacea       Physeter catodon          14540960
```

```r
filter(mammals, species == "Balaena mysticetus")
```

```
## Source: local data frame [1 x 6]
##
##     order            species adult_body_mass_g adult_head_body_len_mm
## 1 Cetacea Balaena mysticetus          79691179                  12187
## Variables not shown: home_range_km2 (dbl), litter_size (dbl)
```

```r
filter(mammals, order == "Carnivora" & adult_body_mass_g < 200)
```

```
## Source: local data frame [3 x 6]
##
##       order         species adult_body_mass_g adult_head_body_len_mm
## 1 Carnivora Mustela altaica            180.24                  243.5
## 2 Carnivora Mustela frenata            190.03                  229.3
## 3 Carnivora Mustela nivalis             78.45                  188.2
## Variables not shown: home_range_km2 (dbl), litter_size (dbl)
```

Arranging rows

`arrange()` lets you order the rows by one or more columns in ascending or descending order. I'm selecting the first three columns only to make the output easier to read:

```
arrange(mammals, adult_body_mass_g)[ , 1:3]
```

```
## Source: local data frame [5,416 x 3]
##
##           order                      species adult_body_mass_g
## 1    Chiroptera Craseonycteris thonglongyai              1.96
## 2    Chiroptera            Kerivoula minuta              2.03
## 3  Soricomorpha             Suncus etruscus              2.26
## 4  Soricomorpha          Sorex minutissimus              2.46
## 5  Soricomorpha      Suncus madagascariensis              2.47
## 6  Soricomorpha          Crocidura lusitania              2.48
## 7  Soricomorpha          Crocidura planiceps              2.50
## 8    Chiroptera        Pipistrellus nanulus              2.51
## 9  Soricomorpha                  Sorex nanus              2.57
## 10 Soricomorpha               Sorex arizonae              2.70
## ..          ...                          ...               ...
```

```
arrange(mammals, desc(adult_body_mass_g))[ , 1:3]
```

```
## Source: local data frame [5,416 x 3]
##
##      order                species adult_body_mass_g
## 1  Cetacea  Balaenoptera musculus         154321304
## 2  Cetacea     Balaena mysticetus          79691179
## 3  Cetacea  Balaenoptera physalus          47506008
## 4  Cetacea      Caperea marginata          32000000
## 5  Cetacea Megaptera novaeangliae          30000000
## 6  Cetacea  Eschrichtius robustus          27324024
## 7  Cetacea    Eubalaena australis          23000000
## 8  Cetacea    Eubalaena glacialis          23000000
## 9  Cetacea  Balaenoptera borealis          22106252
## 10 Cetacea     Balaenoptera edeni          20000000
## ..     ...                    ...               ...
```

```
arrange(mammals, order, adult_body_mass_g)[ , 1:3]
```

```
## Source: local data frame [5,416 x 3]
```

```
##
##           order                 species adult_body_mass_g
## 1  Afrosoricida      Microgale pusilla              3.40
## 2  Afrosoricida      Microgale parvula              3.53
## 3  Afrosoricida         Geogale aurita              6.69
## 4  Afrosoricida   Microgale fotsifotsy              7.70
## 5  Afrosoricida Microgale longicaudata              8.08
## 6  Afrosoricida Microgale brevicaudata              8.99
## 7  Afrosoricida   Microgale principula             10.20
## 8  Afrosoricida    Microgale drouhardi             10.50
## 9  Afrosoricida       Microgale cowani             12.27
## 10 Afrosoricida        Microgale taiva             12.40
## ..           ...                    ...               ...
```

Mutating columns

`mutate()` lets you add new columns. Notice that the new columns you create can build on each other. I will wrap these in `glimpse()` to make the new columns easy to see:

```
glimpse(mutate(mammals, adult_body_mass_kg = adult_body_mass_g / 1000))
```

```
## Variables:
## $ order                  (chr) "Artiodactyla", "Carnivora", "Carnivora...
## $ species                (chr) "Camelus dromedarius", "Canis adustus",...
## $ adult_body_mass_g      (dbl) 492714.5, 10392.5, 9658.7, 11989.1, 317...
## $ adult_head_body_len_mm (dbl) NA, 745.3, 827.5, 872.4, 1055.0, 2700.0...
## $ home_range_km2         (dbl) 196.32000, 1.01000, 2.95000, 18.88000, ...
## $ litter_size            (dbl) 0.98, 4.50, 3.74, 5.72, 4.98, 1.22, 1.0...
## $ adult_body_mass_kg     (dbl) 492.7145, 10.3925, 9.6587, 11.9891, 31....
```

```
glimpse(mutate(mammals,
    g_per_mm = adult_body_mass_g / adult_head_body_len_mm))
```

```
## Variables:
## $ order                  (chr) "Artiodactyla", "Carnivora", "Carnivora...
## $ species                (chr) "Camelus dromedarius", "Canis adustus",...
## $ adult_body_mass_g      (dbl) 492714.5, 10392.5, 9658.7, 11989.1, 317...
## $ adult_head_body_len_mm (dbl) NA, 745.3, 827.5, 872.4, 1055.0, 2700.0...
## $ home_range_km2         (dbl) 196.32000, 1.01000, 2.95000, 18.88000, ...
## $ litter_size            (dbl) 0.98, 4.50, 3.74, 5.72, 4.98, 1.22, 1.0...
## $ g_per_mm               (dbl) NA, 13.9437, 11.6717, 13.7428, 30.1010,...
```

```
glimpse(mutate(mammals,
    g_per_mm = adult_body_mass_g / adult_head_body_len_mm,
    kg_per_mm = g_per_mm / 1000))
```

```
## Variables:
## $ order                  (chr) "Artiodactyla", "Carnivora", "Carnivora...
## $ species                (chr) "Camelus dromedarius", "Canis adustus",...
## $ adult_body_mass_g      (dbl) 492714.5, 10392.5, 9658.7, 11989.1, 317...
## $ adult_head_body_len_mm (dbl) NA, 745.3, 827.5, 872.4, 1055.0, 2700.0...
## $ home_range_km2         (dbl) 196.32000, 1.01000, 2.95000, 18.88000, ...
## $ litter_size            (dbl) 0.98, 4.50, 3.74, 5.72, 4.98, 1.22, 1.0...
## $ g_per_mm               (dbl) NA, 13.9437, 11.6717, 13.7428, 30.1010,...
## $ kg_per_mm              (dbl) NA, 0.0139437, 0.0116717, 0.0137428, 0....
```

Summarising columns

Finally, `summarise()` lets you calculate summary statistics. On its own `summarise()` isn't that useful, but when combined with `group_by()` you can summarise by chunks of data. This is similar to what you might be familiar with through `ddply()` and `summarise()` from the plyr package:

```r
summarise(mammals, mean_mass = mean(adult_body_mass_g, na.rm = TRUE))
```

```
## Source: local data frame [1 x 1]
##
##   mean_mass
## 1    177810
```

```r
# summarise with group_by:
head(summarise(group_by(mammals, order),
  mean_mass = mean(adult_body_mass_g, na.rm = TRUE)))
```

```
## Source: local data frame [6 x 2]
##
##          order mean_mass
## 1 Afrosoricida 9.476e+01
## 2 Artiodactyla 1.213e+05
## 3    Carnivora 4.739e+04
## 4      Cetacea 7.373e+06
## 5   Chiroptera 5.772e+01
## 6    Cingulata 4.699e+03
```

Piping data

Pipes take the output from one function and feed it to the first argument of the next function. You may have encountered the Unix pipe `|` before.

The magrittr R package contains the pipe function `%>%`. Yes it might look bizarre at first but it makes more sense when you think about it. The R language allows symbols wrapped in `%` to be defined as functions, the `>` helps imply a chain, and you can hit these 2 characters one after the other very quickly on a keyboard by holding down the Shift key. Try it!

Try pronouncing `%>%` "then" whenever you see it. If you want to see the help page, you'll need to wrap it in back ticks like so:

```r
?magrittr::`%>%`
```

A trivial pipe example

Pipes can work with nearly any functions. Let's start with a non-dplyr example:

```
x <- rnorm(10)
x %>% max
```

```
## [1] 1.77
```

```
# is the same thing as:
max(x)
```

```
## [1] 1.77
```

So, we took the value of `x` (what would have been printed on the console), captured it, and fed it to the first argument of `max()`. It's probably not clear why this is cool yet, but hang on.

A silly dplyr example with pipes

Let's try a single-pipe dplyr example. We'll pipe the `mammals` data frame to the arrange function's first argument, and choose to arrange by the `adult_body_mass_g` column:

```
mammals %>% arrange(adult_body_mass_g)
```

```
## Source: local data frame [5,416 x 6]
##
##           order                     species adult_body_mass_g
## 1    Chiroptera Craseonycteris thonglongyai              1.96
## 2    Chiroptera            Kerivoula minuta              2.03
## 3  Soricomorpha             Suncus etruscus              2.26
## 4  Soricomorpha          Sorex minutissimus              2.46
## 5  Soricomorpha     Suncus madagascariensis              2.47
## 6  Soricomorpha         Crocidura lusitania              2.48
## 7  Soricomorpha         Crocidura planiceps              2.50
## 8    Chiroptera         Pipistrellus nanulus              2.51
## 9  Soricomorpha                 Sorex nanus              2.57
## 10 Soricomorpha              Sorex arizonae              2.70
## ..          ...                         ...               ...
## Variables not shown: adult_head_body_len_mm (dbl), home_range_km2 (dbl),
##   litter_size (dbl)
```

An awesome example

OK, here's where it gets cool. We can chain dplyr functions in succession. This lets us write data manipulation steps in the order we think of them and avoid creating temporary variables in the middle to capture the output. This works because the output from every dplyr function is a data frame and the first argument of every dplyr function is a data frame.

Say we wanted to find the species with the highest body-mass-to-length ratio:

```
mammals %>%
  mutate(mass_to_length = adult_body_mass_g / adult_head_body_len_mm) %>%
  arrange(desc(mass_to_length)) %>%
  select(species, mass_to_length)
```

```
## Source: local data frame [5,416 x 2]
##
##                   species mass_to_length
## 1      Balaena mysticetus           6539
## 2   Balaenoptera musculus           5063
## 3  Megaptera novaeangliae           2334
## 4   Eschrichtius robustus           2309
## 5   Balaenoptera physalus           2302
## 6         Elephas maximus           1704
## 7     Eubalaena glacialis           1654
## 8     Eubalaena australis           1625
## 9      Balaenoptera edeni           1444
## 10  Balaenoptera borealis           1203
## ..                    ...            ...
```

So, we took `mammals`, fed it to `mutate()` to create a mass-length ratio column, arranged the resulting data frame in descending order by that ratio, and selected the columns we wanted to see. This is just the beginning. If you can imagine it, you can string it together. If you want to debug your code, just pull a pipe off the end and run the code down to that step. Or build your analysis up and add successive pipes.

The above is equivalent to:

```
select(
  arrange(
    mutate(mammals,
      mass_to_length = adult_body_mass_g / adult_head_body_len_mm),
    desc(mass_to_length)),
  species, mass_to_length)
```

```
## Source: local data frame [5,416 x 2]
##
##                   species mass_to_length
## 1      Balaena mysticetus           6539
## 2   Balaenoptera musculus           5063
## 3  Megaptera novaeangliae           2334
## 4   Eschrichtius robustus           2309
## 5   Balaenoptera physalus           2302
## 6         Elephas maximus           1704
## 7     Eubalaena glacialis           1654
## 8     Eubalaena australis           1625
## 9      Balaenoptera edeni           1444
## 10  Balaenoptera borealis           1203
## ..                    ...            ...
```

But the problem here is that you have to read it inside out, it's easy to miss a bracket, and the arguments get separated from the function (e.g. see `mutate()` and `desc(mass_to_length)))`. Plus, this is a rather trivial example. Chain together even more steps and it quickly gets out of hand.

Here's one more example. Let's ask what taxonomic orders have a median litter size greater than 3.

```
mammals %>% group_by(order) %>%
  summarise(median_litter = median(litter_size, na.rm = TRUE)) %>%
  filter(median_litter > 3) %>%
  arrange(desc(median_litter)) %>%
  select(order, median_litter)
```

```
## Source: local data frame [5 x 2]
##
##             order median_litter
## 1 Didelphimorphia         6.595
## 2  Dasyuromorphia         6.190
## 3  Erinaceomorpha         3.870
## 4    Soricomorpha         3.660
## 5        Rodentia         3.280
```

These examples don't even highlight one of the best things about dplyr. It's *really* fast. The internal C++ code makes quick work of massive data frames that would make plyr slow to a crawl.

dplyr can do much more, but the above are the basics of the 5 verbs and pipes. Try them for a bit. Once they click I think they'll revolutionize your data analysis.