The Usefulness of the dplyr Package

Yonas Kbrom

10/31/2017

Motivation

The dplyer library is something I found to be one the most interesting things I have learned so far in this course. Even though I have prior coding experience and have been exposed to many libraries and packages, dplyr is the first thing that I have come to in this class and feel as if I was learning something new, besides the slightly different syntax of R compared to other languages such as python and java. Throughout this post I will dive deeply into what exactly the dplyr package is capable of and I find so fascinating compared to anything else I have learned in this course or even through my previous computer science courses here at UC Berkeley.

Introduction

For those of you who do not already know, the dplyr package is a package that allows programmers to manipulate data in a clean and concise manner. Data manipulation in itself can be a bit complex, but with this package manipulating data is made simple. With dplyr, there are a variety of methods and tools that can be used to better visualize and manipulate the data you are working with, such as “filter”, “select”, “group_by”, “mutate”, “summarize”, “arrange”, and the piping paradigm. Without this package, attempting to manipulate data in any other way would easily prove how necessary and impactful dplyr is and will continue to be.

Examples

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

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
##
## 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("ggplot2")
```

Throughout these examples I will be using the data frame “starwars” that comes built in.

```
starwars
```

```
## # A tibble: 87 x 13
##                  name height  mass    hair_color  skin_color eye_color
##                 <chr>  <int> <dbl>         <chr>       <chr>     <chr>
##  1     Luke Skywalker    172    77         blond        fair      blue
##  2              C-3PO    167    75          <NA>        gold    yellow
##  3              R2-D2     96    32          <NA> white, blue       red
##  4        Darth Vader    202   136          none       white    yellow
##  5        Leia Organa    150    49         brown       light     brown
##  6          Owen Lars    178   120   brown, grey       light      blue
##  7 Beru Whitesun lars    165    75         brown       light      blue
##  8              R5-D4     97    32          <NA>  white, red       red
##  9  Biggs Darklighter    183    84         black       light     brown
## 10     Obi-Wan Kenobi    182    77 auburn, white        fair blue-gray
## # ... with 77 more rows, and 7 more variables: birth_year <dbl>,
## #   gender <chr>, homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

One of the first functions I truly enjoyed learning and utilizing is “union_all”. With union all, you are capable of joining two data frames with the same columns efficiently. The following code segment provides an example:

```
filter(starwars, height==79)
```

```
## # A tibble: 1 x 13
##            name height  mass hair_color skin_color eye_color birth_year
##           <chr>  <int> <dbl>      <chr>      <chr>     <chr>      <dbl>
## 1 Ratts Tyerell     79    15       none grey, blue   unknown         NA
## # ... with 6 more variables: gender <chr>, homeworld <chr>, species <chr>,
## #   films <list>, vehicles <list>, starships <list>
```

```
first = head(starwars, 5)
second = tail(starwars, 5)
union_all(first, second)
```

```
## # A tibble: 10 x 13
##               name height  mass hair_color  skin_color eye_color birth_year
##              <chr>  <int> <dbl>      <chr>       <chr>     <chr>      <dbl>
##  1 Luke Skywalker    172    77      blond        fair      blue       19.0
##  2           C-3PO    167    75       <NA>        gold    yellow      112.0
##  3           R2-D2     96    32       <NA> white, blue       red       33.0
##  4     Darth Vader    202   136       none       white    yellow       41.9
##  5     Leia Organa    150    49      brown       light     brown       19.0
##  6             Rey     NA    NA      brown       light     hazel         NA
##  7     Poe Dameron     NA    NA      brown       light     brown         NA
##  8             BB8     NA    NA       none        none     black         NA
##  9 Captain Phasma     NA    NA    unknown     unknown   unknown         NA
## 10   Padmé Amidala    165    45      brown       light     brown       46.0
## # ... with 6 more variables: gender <chr>, homeworld <chr>, species <chr>,
## #   films <list>, vehicles <list>, starships <list>
```

Along with this is a function group_by, which although discussed in labs, many students seemed very unsure of how this worked or how to use it. The main idea of group_by is to group a number of rows together based on a certain criteria, or column, and perform some operation with those groups. The group_by method itself does not change the data frame, it only allows you to insert it as an argument to another method for varying reasons. Here is an example of how the group_by operator works with the summarize method:

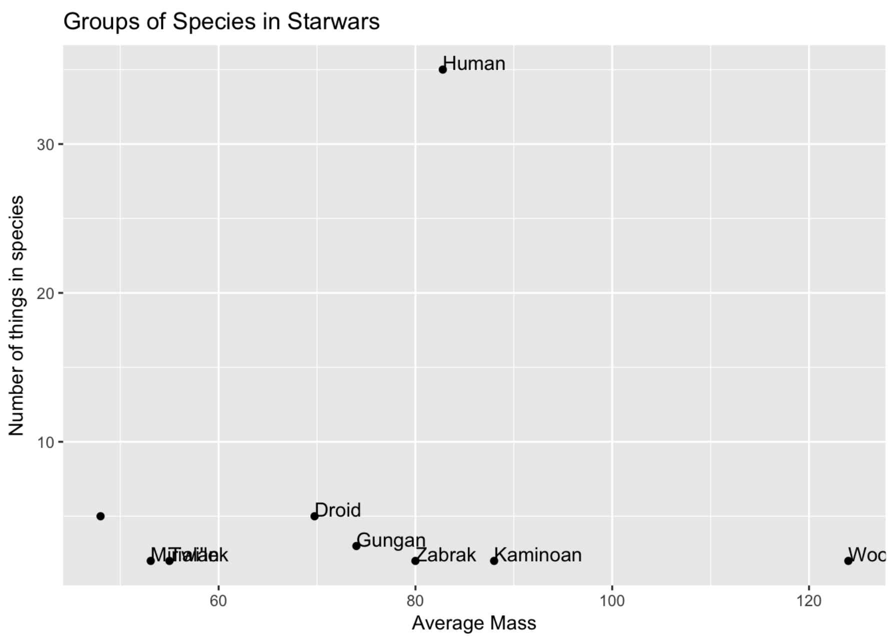
```
species1 = filter(summarise(group_by(starwars, species), n = n(), mass = mean(mass, na.rm = TRUE)), n > 1)
species1
```

```
## # A tibble: 9 x 3
##    species     n      mass
##      <chr> <int>     <dbl>
## 1    Droid     5  69.75000
## 2   Gungan     3  74.00000
## 3    Human    35  82.78182
## 4 Kaminoan     2  88.00000
## 5 Mirialan     2  53.10000
## 6  Twi'lek     2  55.00000
## 7  Wookiee     2 124.00000
## 8   Zabrak     2  80.00000
## 9     <NA>     5  48.00000
```

As you can see here, I have grouped the data frame starwars by species and printed out the number of things are categorized as that specific species. You can also see how once they are grouped, we also find the average mass contained within that species, making sure that each species has some mass that can be accounted for and is not some massless entity. If we would actually like to see this data visually, we could just use ggplot, a handy library used to create beautiful graphs when given data.

```
ggplot(data = species1, aes(x = mass, y = n)) +
  geom_point() + xlab("Average Mass") + ylab("Number of things in species") + geom_text(aes(label = species), hjus
t=0, vjust=0) + ggtitle("Groups of Species in Starwars")
```

```
## Warning: Removed 1 rows containing missing values (geom_text).
```



One final method that I found extremely useful in terms of what I use consistently since I have learned it was piping, a concept that is used often in R languages using dplyr and Linux based systems. With piping, one is easily able to put the previous output into another method without dealing with potentially long, nested functions that come with terrible readability. An example of this can be done on the previous code segment, which used multiple nested functions.

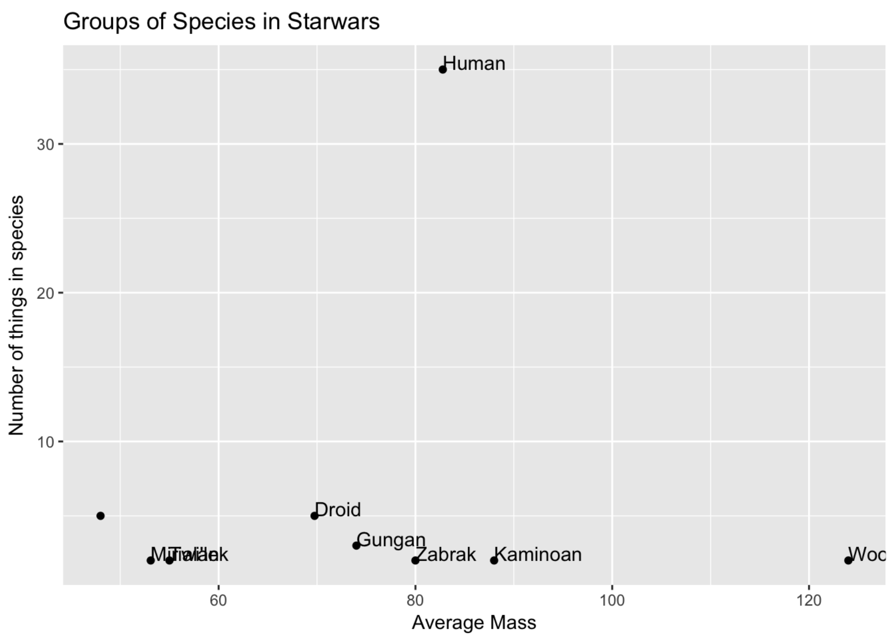
```
species1 = starwars %>% group_by(species) %>% summarise(n = n(), mass = mean(mass, na.rm = TRUE)) %>% filter(n > 1
)
species1
```

```
## # A tibble: 9 x 3
##    species     n      mass
##      <chr> <int>     <dbl>
## 1    Droid     5  69.75000
## 2   Gungan     3  74.00000
## 3    Human    35  82.78182
## 4 Kaminoan     2  88.00000
## 5 Mirialan     2  53.10000
## 6  Twi'lek     2  55.00000
## 7  Wookiee     2 124.00000
## 8   Zabrak     2  80.00000
## 9     <NA>     5  48.00000
```

With this, we get a much more concise version and one that is much easier to read. You can also see how the graph is still similar to the one with nesting.

```
ggplot(data = species1, aes(x = mass, y = n)) +
  geom_point() + xlab("Average Mass") + ylab("Number of things in species") + geom_text(aes(label = species), hjus
t=0, vjust=0) + ggtitle("Groups of Species in Starwars")
```

```
## Warning: Removed 1 rows containing missing values (geom_text).
```



Discussion

The dplyr package is something that is definitely worth looking more into and experimenting more with. Learning more about how to aggregate data frames with “group_by”" and filter out certain rows with “filter” are things that are truly useful no matter what type of data analytic job you are into. Another useful thing I found about dplyer that was not mentioned either in lecture or lab was how diverse the package is behind just the couple of methods mentioned. As explained previously, the “union_all” method is very useful and can have a real impact on how beginners learn to manipulate data effectively. There are also others such as “summarize_if”“,”top_n“, and”coalesce" just to name a few. The impact dplyr can have on both beginners and experts is immense and as I said earlier, is something that is definitely worth playing with and looking more into.

Conclusions

All in all, the dplyr package is one that everyone should at least play with, specifically those who intend to work with many data bases and hope to one day serve as data analysts. Even if one is just curious about how data manipulation can be done on certain data frames and data in general, the dplyr package is the best way to go.

References

- http://dplyr.tidyverse.org/
- https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html
- https://www.r-bloggers.com/data-manipulation-with-dplyr/
- https://spark.rstudio.com/dplyr.html
- https://www.rdocumentation.org/packages/dplyr/versions/0.5.0
- https://rpubs.com/williamsurles/292547
- http://bioconnector.org/workshops/r-dplyr-yeast.html

- https://www3.nd.edu/~steve/computing_with_data/24_dplyr/dplyr.html