post02 - A walkthrough of dplyr package functions and why it’s more convenient than base R functions

Avanti Mehrotra

11/26/2017

Introduction

While I was researching a topic to write my post about, I started reading about **dplyr**. Although I thought I knew all things necessary to use the package, I quickly realized there’s much more information to cover and learn, as one would predict. The purpose of my post is to explain why **dplyr** is so useful, how to use some of its useful functions, and why it’s so important in real world applications. We’ll first walk through examples of various dplyr functions. Let’s get started!

What exactly is the dplyr package?

The **dplyr** package is used for a multitude of reasons. Some reasons include its high-speed performance and that it helps us see which are the best tools to manipulate our datasets. **dplyr** is especially useful for data frames.

First, we need to download the packages necessary to read in the CSV file with the data we’d like to look at. We also need to download the **dplyr** package, and later we’ll use the ggplot2 package, so let’s download that too.

```r
# load packages necessary in this post
library(readr)
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr
```

```
## Conflicts with tidy packages ----------------------------------------------
```

```
## filter(): dplyr, stats
## lag():    dplyr, stats
```

```r
library(dplyr)
library(ggplot2)
```

Next, read the CSV file, downloaded from *FiveThirtyEight* (ii), and save it as a dataframe. The CSV file includes “data on surnames from the U.S. Census Bureau”, as stated in the README file describing data in the most-common-name folder (ii).

```r
# downloading data with read_csv()
names_data <- read_csv(file = "../data/surnames.csv")
```

```
## Parsed with column specification:
## cols(
##   name = col_character(),
##   rank = col_integer(),
##   count = col_integer(),
##   prop100k = col_double(),
##   cum_prop100k = col_double(),
##   pctwhite = col_double(),
##   pctblack = col_character(),
##   pctapi = col_double(),
##   pctaian = col_character(),
##   pct2prace = col_double(),
##   pcthispanic = col_double()
## )
```

```
## Warning in rbind(names(probs), probs_f): number of columns of result is not
## a multiple of vector length (arg 1)
```

```
## Warning: 188854 parsing failures.
## row # A tibble: 5 x 5 col     row         col expected actual                   file expected   <int>       <ch
r>    <chr>  <chr>                  <chr> actual 1  2099 pcthispanic a double    (S) '../data/surnames.csv' file 2
3228      pctapi a double    (S) '../data/surnames.csv' row 3  3342   pct2prace a double    (S) '../data/surnames.
csv' col 4  3658      pctapi a double    (S) '../data/surnames.csv' expected 5  3962      pctapi a double    (S) '
../data/surnames.csv'
## ... ................. ... ........................................................... ........ .................
........................................... ...... .......................................................... .... .
......................................................... ... ...................................................
...... ... ........................................................... ........ ..................................
.......................
## See problems(...) for more details.
```

Now that we have a dataframe, let’s get an idea of what our data frame looks like, and then let’s use the **dplyr** package to do some data manipulation.

First, let’s check what the first ten rows of names_data looks like by using the head() function.

```r
# see preview of first ten rows of names_data
head(names_data, 10)
```

```
## # A tibble: 10 x 11
##         name  rank   count prop100k cum_prop100k pctwhite pctblack pctapi
##        <chr> <int>   <int>    <dbl>        <dbl>    <dbl>    <chr>  <dbl>
##  1     SMITH     1 2376206   880.85       880.85    73.35    22.22   0.40
##  2   JOHNSON     2 1857160   688.44      1569.30    61.55     33.8   0.42
##  3  WILLIAMS     3 1534042   568.66      2137.96    48.52    46.72   0.37
##  4     BROWN     4 1380145   511.62      2649.58    60.71    34.54   0.41
##  5     JONES     5 1362755   505.17      3154.75    57.69    37.73   0.35
##  6    MILLER     6 1127803   418.07      3572.82    85.81    10.41   0.42
##  7     DAVIS     7 1072335   397.51      3970.33    64.73    30.77   0.40
##  8    GARCIA     8  858289   318.17      4288.50     6.17     0.49   1.43
##  9 RODRIGUEZ     9  804240   298.13      4586.62     5.52     0.54   0.58
## 10    WILSON    10  783051   290.27      4876.90    69.72    25.32   0.46
## # ... with 3 more variables: pctaian <chr>, pct2prace <dbl>,
## #   pcthispanic <dbl>
```

Since **dplyr** lets us easily manipulate our dataframes, let's decide which columns we want to use for analyzing.

How many rows of data do we have?

```
# count how many rows are in names_data
num_rows <- nrow(names_data)
num_rows
```

```
## [1] 151671
```

Since we have over 150,000 entries in our dataframe, let's look at only the first five columns. With the use of **dplyr**, we can easily **select** the first five columns.

```
# select only name, rank, count, prop100k, cum_prop100k columns from names_data
names_and_props <- select(names_data, name, rank, count, prop100k, cum_prop100k)
names_and_props
```

```
## # A tibble: 151,671 x 5
##         name  rank   count prop100k cum_prop100k
##        <chr> <int>   <int>    <dbl>        <dbl>
##  1     SMITH     1 2376206   880.85       880.85
##  2   JOHNSON     2 1857160   688.44      1569.30
##  3  WILLIAMS     3 1534042   568.66      2137.96
##  4     BROWN     4 1380145   511.62      2649.58
##  5     JONES     5 1362755   505.17      3154.75
##  6    MILLER     6 1127803   418.07      3572.82
##  7     DAVIS     7 1072335   397.51      3970.33
##  8    GARCIA     8  858289   318.17      4288.50
##  9 RODRIGUEZ     9  804240   298.13      4586.62
## 10    WILSON    10  783051   290.27      4876.90
## # ... with 151,661 more rows
```

dplyr's select() allows one to use other helper functions to actually select various columns, which can't be done simply with base R's subset() (iv).

For example, say we wanted to look at the first five columns but don't want to write out the last two column names. select() has a nifty helper function — contains() — that'll allow us to search for columns that have "prop" in the name. Check it out:

```
# example of using select()'s helper function contains()
ex_names_and_props <- select(names_data, name, rank, count, contains("prop"))
ex_names_and_props
```

```
## # A tibble: 151,671 x 5
##         name  rank   count prop100k cum_prop100k
##        <chr> <int>   <int>    <dbl>        <dbl>
##  1     SMITH     1 2376206   880.85       880.85
##  2   JOHNSON     2 1857160   688.44      1569.30
##  3  WILLIAMS     3 1534042   568.66      2137.96
##  4     BROWN     4 1380145   511.62      2649.58
##  5     JONES     5 1362755   505.17      3154.75
##  6    MILLER     6 1127803   418.07      3572.82
##  7     DAVIS     7 1072335   397.51      3970.33
##  8    GARCIA     8  858289   318.17      4288.50
##  9 RODRIGUEZ     9  804240   298.13      4586.62
## 10    WILSON    10  783051   290.27      4876.90
## # ... with 151,661 more rows
```

Isn't that convenient? Columns can similarly be selected with other select() helper functions like ends_with() and starts_with().

Let's create another data frame with the surnames and the corresponding percentage of surnames belonging to Hispanic people, so we can join it to our names_and_props data frame a little later in this post. This will allow us to implement **dplyr**'s join() function.

```
# create new data frame with name and pcthispanic from names_data
pct_hispanic_people <- select(names_data, name, contains("hispanic"))
pct_hispanic_people
```

```
## # A tibble: 151,671 x 2
##         name pcthispanic
##        <chr>       <dbl>
##  1     SMITH        1.56
##  2   JOHNSON        1.50
##  3  WILLIAMS        1.60
##  4     BROWN        1.64
##  5     JONES        1.44
##  6    MILLER        1.43
##  7     DAVIS        1.58
##  8    GARCIA       90.81
##  9 RODRIGUEZ       92.70
## 10    WILSON        1.73
## # ... with 151,661 more rows
```

There are several other functions in **dplyr** that are highly useful. For example, let's say we want to confirm that the only surname with count over 2,000,000 is, indeed, Smith, since the names_data is in descending order according to "count". We can use **dplyr**'s filter() function to filter through the rows of our data frame. If Smith is the only name with more than 2,000,000 occurences, then after filtering through our data frame, our new data frame should only have one row. Let's confirm this fact that our original data frame told us.

```
# filter names_data to only include surnames with count greater than 2000000
check_smith <- filter(names_data, count > 2000000)
check_smith
```

```
## # A tibble: 1 x 11
##    name  rank   count prop100k cum_prop100k pctwhite pctblack pctapi
##   <chr> <int>   <int>    <dbl>        <dbl>    <dbl>    <chr>  <dbl>
## 1 SMITH     1 2376206   880.85       880.85    73.35    22.22    0.4
## # ... with 3 more variables: pctaian <chr>, pct2prace <dbl>,
## #   pcthispanic <dbl>
```

What are the benefits of using **dplyr**'s filter() function? filter() is more efficient than base R's subset() function for large data sets, although it can be slow for filtering just one or a few elements (v).

Let's take a look at **dplyr**'s join() function which allows us to join two tables, usually from the same main source of data (vi). **dplyr** has different types of join() functions, including **inner_join()**, **left_join()**, **right_join()**, **semi_join()**, **anti_join()**, and **full_join()**. Let's merge our names_and_props table with our pct_hispanic_people table, with the pct_hispanic_people table joined to the right of names_and_props, and we'll join the two tables by the "name" column.

```
# join the names_and_props table with the pct_hispanic_people using the name column
names_and_hispanic_pct <- right_join(names_and_props, pct_hispanic_people, by = "name")
names_and_hispanic_pct
```

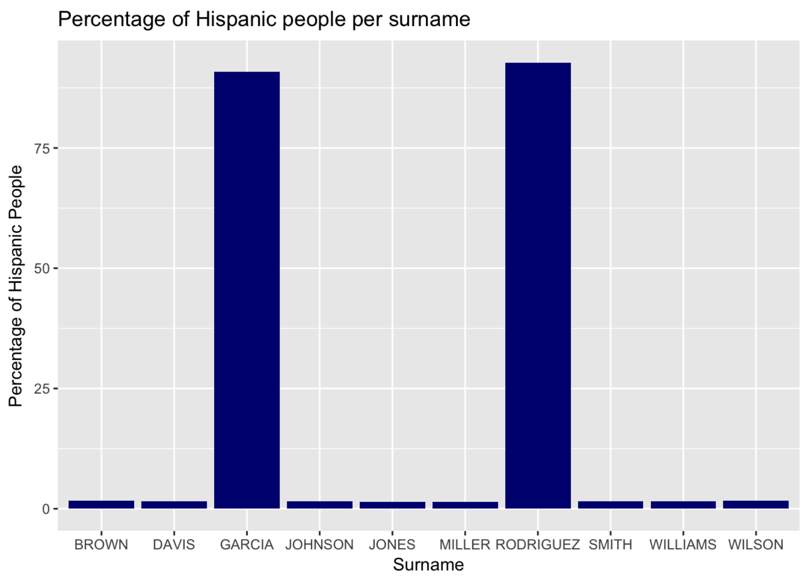
```
## # A tibble: 151,671 x 6
##         name  rank   count prop100k cum_prop100k pcthispanic
##        <chr> <int>   <int>    <dbl>        <dbl>       <dbl>
##  1     SMITH     1 2376206   880.85       880.85        1.56
##  2   JOHNSON     2 1857160   688.44      1569.30        1.50
##  3  WILLIAMS     3 1534042   568.66      2137.96        1.60
##  4     BROWN     4 1380145   511.62      2649.58        1.64
##  5     JONES     5 1362755   505.17      3154.75        1.44
##  6    MILLER     6 1127803   418.07      3572.82        1.43
##  7     DAVIS     7 1072335   397.51      3970.33        1.58
##  8    GARCIA     8  858289   318.17      4288.50       90.81
##  9 RODRIGUEZ     9  804240   298.13      4586.62       92.70
## 10    WILSON    10  783051   290.27      4876.90        1.73
## # ... with 151,661 more rows
```

Pretty cool, right? One might reasonably wonder what the benefit of using **dplyr**'s join() over base R's merge() function is. Again, join() is simply much faster than the similar base R function, the rows are kept in the same order, and even if you don't specify which column you're joining the tables by, join() will tell you what was used to join them (vii).

Another nice thing about the **dplyr** package is that after manipulating data with it, the manipulated data works really well with the **ggplot2** package (viii). This is because we've easily manipulated our data into a dataset that we want to visualize. Let's do a simple visualization of our dataframe names_and_hispanic_pct.

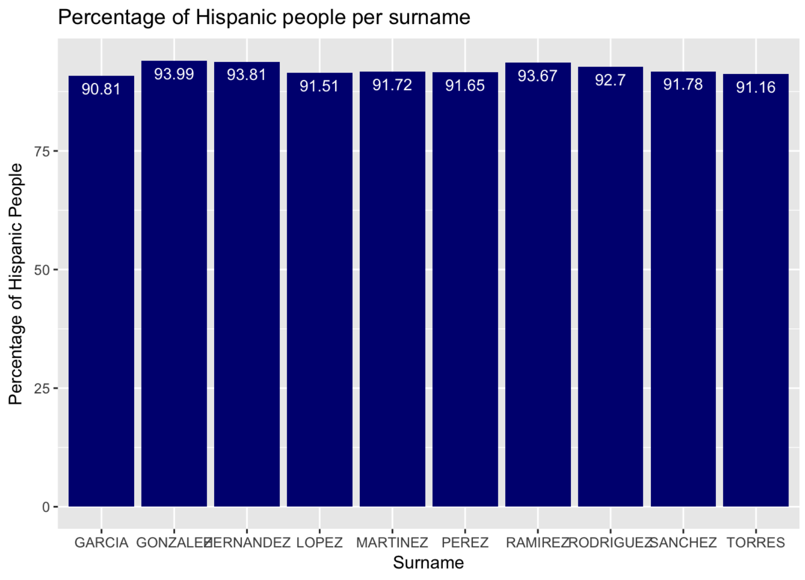
Just looking at the top ten names in our dataframe, the percentage of Hispanic people with surnames Garcia and Rodriguez is over 90 percent. It would be interesting to plot this against the other surnames that make up the top ten surnames. Since we have the percentages of hispanic people for each surname, let's create a barplot of the top ten most common surnames, where we see the percentage of surnames belonging to Hispanic people (ix).

```
# create a barplot for the top ten ranked surnames and the
# percentage of Hispanic people per surname
pct_barplot <- ggplot(data = filter(names_and_hispanic_pct, rank <= 10), aes(x = name, y = pcthispanic)) + geom_ba
r(stat = "identity", fill = "navy") + labs(title = "Percentage of Hispanic people per surname", x = "Surname", y =
"Percentage of Hispanic People")
pct_barplot
```



Now, let's try creating a barplot of all of the surnames where the percentage of Hispanic people is greater than 50%. First, we'll have to filter() our data so that it only includes surnames that fit our criteria. Then, we can use ggplot again to create a barplot. Since we have thousands of data, let's again narrow the number of rows we use. Let's only use the top ten surnames with percentage of Hispanic people exceeding 50%. We'll also label each bar in our barplot with the exact percentage of Hispanic people per surname — this will help us easily differentiate the bars from each other since they may all seem very close in height to each other (ix).

```
# create a barplot for the top ten ranked surnames where the
# percentage of Hispanic people per surname is greater than 50%
more_than_50 <- head(filter(names_and_hispanic_pct, pcthispanic > 50), 10)
pct_barplot2 <- ggplot(data = more_than_50, aes(x = name, y = pcthispanic)) + geom_bar(stat = "identity", fill = "
navy") + labs(title = "Percentage of Hispanic people per surname", x = "Surname", y = "Percentage of Hispanic Peop
le") + geom_text(aes(label = pcthispanic), vjust=1.6, color="white", position = position_dodge(0.9), size = 3.5)
pct_barplot2
```



Conclusion

As we've seen through our detailed walkthrough of **dplyr** and some of its key functions, the package is highly useful, especially when dealing with large sets of data (x). *Why* is the package especially useful? It makes analyzing and manipulating data much faster, and it also helps us actually manipulate and clean our sets of data easily; in other words, **dplyr** is useful for *data processing* (xi). **dplyr** is is also useful in its real world applications. For example, as detailed in the book *AngularJS: Maintaining Web Applications*, calling a dataframe created with **dplyr** is user-friendly in that it doesn't display rows and rows of data, which is something that would happen with a dataframe created with base R functions. Moreover, by implementing **dplyr** in your data processing, you can connect to various databases, including MySQL and SQLite, which in turn can allow us to create even larger datasets and allows for easy interaction. In this post, we walked through various **dplyr** functions, learned why its functions are often more useful than those from base R, and understood some of its real world applications.

Resources Used

- https://www.r-project.org/nosvn/pandoc/dplyr.html (i)
- https://github.com/fivethirtyeight/data/tree/master/most-common-name (ii)
- http://dplyr.tidyverse.org/ (iii)
- http://seananderson.ca/2014/09/13/dplyr-intro.html (iv)
- https://stackoverflow.com/questions/39882463/difference-between-subset-and-filter-from-dplyr (v)
- http://dplyr.tidyverse.org/reference/join.html (vi)
- http://zevross.com/blog/2014/04/30/mini-post-for-large-tables-in-r-dplyrs-function-inner_join-is-much-faster-than-merge/ (vii)
- http://sharpsightlabs.com/blog/data-analysis-example-r-supercars-part2/ (vii)
- http://www.sthda.com/english/wiki/ggplot2-barplots-quick-start-guide-r-software-and-data-visualization (ix)
- http://zevross.com/blog/2014/03/26/four-reasons-why-you-should-check-out-the-r-package-dplyr-3/ (x)

- https://csgillespie.github.io/efficientR/data-carpentry.html (xi)
- https://books.google.com/books?id=ADrgDAAAQBAJ&pg=PA1730&lpg=PA1730&dq=real+world+applications+of+dplyr&source=bl&ots=KVwpOwWT3R&sig=q98veNvXT-UsFn9uVZTD7Qv81Mk&hl=en&sa=X&ved=0ahUKEwiOg9CUwO3XAhXHllQKHdGCBNEQ6AEISDAF#v=onepage&q=real%20world%20applications%20of%20dpl
  (xii)