Manipulating Data with dplyr

Raphael Carvalho

02/07/2019

I've created a variable called path2csv, which contains the full file path to the dataset. Call read.csv() with two arguments, path2csv and stringsAsFactors = FALSE, and save the result in a new variable called mydf. Check ?read.csv if you need help.

```
mydf <- read.csv(path2csv, stringsAsFactors = FALSE)
```

The first step of working with data in dplyr is to load the data into what the package authors call a 'data frame tbl' or 'tbl_df'. Use the following code to create a new tbl_df called cran: cran <- tbl_df(mydf).

```
cran <- tbl_df(mydf)
```

As may often be the case, particularly with larger datasets, we are only interested in some of the variables. Use select(cran, ip_id, package, country) to select only the ip_id, package, and country variables from the cran dataset.

```
select(cran, ip_id, package, country)
```

```
## # A tibble: 225,468 x 3
##    ip_id package      country
##    <int> <chr>        <chr>
##  1     1 htmltools    US
##  2     2 tseries      US
##  3     3 party        US
##  4     3 Hmisc        US
##  5     4 digest       CA
##  6     3 randomForest US
##  7     3 plyr         US
##  8     5 whisker      US
##  9     6 Rcpp         CN
## 10     7 hflights     US
## # ... with 225,458 more rows
```

Normally, this notation is reserved for numbers, but select() allows you to specify a sequence of columns this way, which can save a bunch of typing. Use select(cran, r_arch:country) to select all columns starting from r_arch and ending with country.

```
select(cran, r_arch:country)
```

```
## # A tibble: 225,468 x 5
##    r_arch r_os      package      version country
##    <chr>  <chr>     <chr>        <chr>   <chr>
##  1 x86_64 mingw32   htmltools    0.2.4   US
##  2 x86_64 mingw32   tseries      0.10-32 US
##  3 x86_64 linux-gnu party        1.0-15  US
##  4 x86_64 linux-gnu Hmisc        3.14-4  US
##  5 x86_64 linux-gnu digest       0.6.4   CA
##  6 x86_64 linux-gnu randomForest 4.6-7   US
##  7 x86_64 linux-gnu plyr         1.8.1   US
##  8 x86_64 linux-gnu whisker      0.3-2   US
##  9 <NA>   <NA>      Rcpp         0.10.4  CN
```

```
## 10 x86_64 linux-gnu hflights     0.1     US
## # ... with 225,458 more rows
```

Instead of specifying the columns we want to keep, we can also specify the columns we want to throw away. To see how this works, do select(cran, -time) to omit the time column.

```
select(cran, -time)
```

```
## # A tibble: 225,468 x 10
##        X date     size r_version r_arch r_os  package version country ip_id
##    <int> <chr>   <int> <chr>     <chr>  <chr> <chr>   <chr>   <chr>   <int>
##  1     1 2014-~ 8.06e4 3.1.0     x86_64 ming~ htmlto~ 0.2.4   US          1
##  2     2 2014-~ 3.22e5 3.1.0     x86_64 ming~ tseries 0.10-32 US          2
##  3     3 2014-~ 7.48e5 3.1.0     x86_64 linu~ party   1.0-15  US          3
##  4     4 2014-~ 6.06e5 3.1.0     x86_64 linu~ Hmisc   3.14-4  US          3
##  5     5 2014-~ 7.98e4 3.0.2     x86_64 linu~ digest  0.6.4   CA          4
##  6     6 2014-~ 7.77e4 3.1.0     x86_64 linu~ random~ 4.6-7   US          3
##  7     7 2014-~ 3.94e5 3.1.0     x86_64 linu~ plyr    1.8.1   US          3
##  8     8 2014-~ 2.82e4 3.0.2     x86_64 linu~ whisker 0.3-2   US          5
##  9     9 2014-~ 5.93e3 <NA>      <NA>   <NA>  Rcpp    0.10.4  CN          6
## 10    10 2014-~ 2.21e6 3.0.2     x86_64 linu~ hfligh~ 0.1     US          7
## # ... with 225,458 more rows
```

The negative sign in front of time tells select() that we DON'T want the time column. Now, let's combine strategies to omit all columns from X through size (X:size).

```
select(cran, -(X:size))
```

```
## # A tibble: 225,468 x 7
##    r_version r_arch r_os      package      version country ip_id
##    <chr>     <chr>  <chr>     <chr>        <chr>   <chr>   <int>
##  1 3.1.0     x86_64 mingw32   htmltools    0.2.4   US          1
##  2 3.1.0     x86_64 mingw32   tseries      0.10-32 US          2
##  3 3.1.0     x86_64 linux-gnu party        1.0-15  US          3
##  4 3.1.0     x86_64 linux-gnu Hmisc        3.14-4  US          3
##  5 3.0.2     x86_64 linux-gnu digest       0.6.4   CA          4
##  6 3.1.0     x86_64 linux-gnu randomForest 4.6-7   US          3
##  7 3.1.0     x86_64 linux-gnu plyr         1.8.1   US          3
##  8 3.0.2     x86_64 linux-gnu whisker      0.3-2   US          5
##  9 <NA>      <NA>   <NA>      Rcpp         0.10.4  CN          6
## 10 3.0.2     x86_64 linux-gnu hflights     0.1     US          7
## # ... with 225,458 more rows
```

Use filter(cran, package == "swirl") to select all rows for which the package variable is equal to "swirl". Be sure to use two equals signs side-by-side!

```
filter(cran, package == "swirl")
```

```
## # A tibble: 820 x 11
##        X date  time    size r_version r_arch r_os  package version country
##    <int> <chr> <chr>  <int> <chr>     <chr>  <chr> <chr>   <chr>   <chr>
##  1    27 2014~ 00:1~ 105350 3.0.2     x86_64 ming~ swirl   2.2.9   US
##  2   156 2014~ 00:2~  41261 3.1.0     x86_64 linu~ swirl   2.2.9   US
##  3   358 2014~ 00:1~ 105335 2.15.2    x86_64 ming~ swirl   2.2.9   CA
##  4   593 2014~ 00:5~ 105465 3.1.0     x86_64 darw~ swirl   2.2.9   MX
##  5   831 2014~ 00:5~ 105335 3.0.3     x86_64 ming~ swirl   2.2.9   US
##  6   997 2014~ 00:3~  41261 3.1.0     x86_64 ming~ swirl   2.2.9   US
##  7  1023 2014~ 00:3~ 106393 3.1.0     x86_64 ming~ swirl   2.2.9   BR
```

```
##  8  1144 2014~ 00:0~ 106534 3.0.2     x86_64 linu~ swirl   2.2.9   US
##  9  1402 2014~ 00:4~  41261 3.1.0     i386   ming~ swirl   2.2.9   US
## 10  1424 2014~ 00:4~ 106393 3.1.0     x86_64 linu~ swirl   2.2.9   US
## # ... with 810 more rows, and 1 more variable: ip_id <int>
```

You can specify as many conditions as you want, separated by commas. For example filter(cran, r_version == "3.1.1", country == "US") will return all rows of cran corresponding to downloads from users in the US running R version 3.1.1. Try it out.

```
knitr::opts_chunk$set(echo = TRUE)
```

You can specify as many conditions as you want, separated by commas. For example filter(cran, r_version == "3.1.1", country == "US") will return all rows of cran corresponding to downloads from users in the US running R version 3.1.1. Try it out.

```
filter(cran, r_version == "3.1.1", country == "US")
```

```
## # A tibble: 1,588 x 11
##        X date  time    size r_version r_arch r_os  package version country
##    <int> <chr> <chr>  <int> <chr>     <chr>  <chr> <chr>   <chr>   <chr>
##  1  2216 2014~ 00:4~ 3.85e5 3.1.1     x86_64 darw~ colors~ 1.2-4   US
##  2 17332 2014~ 03:3~ 1.97e5 3.1.1     x86_64 darw~ httr    0.3     US
##  3 17465 2014~ 03:2~ 2.33e4 3.1.1     x86_64 darw~ snow    0.3-13  US
##  4 18844 2014~ 03:5~ 1.91e5 3.1.1     x86_64 darw~ maxLik  1.2-0   US
##  5 30182 2014~ 04:1~ 7.77e4 3.1.1     i386   ming~ random~ 4.6-7   US
##  6 30193 2014~ 04:0~ 2.35e6 3.1.1     i386   ming~ ggplot2 1.0.0   US
##  7 30195 2014~ 04:0~ 2.99e5 3.1.1     i386   ming~ fExtre~ 3010.81 US
##  8 30217 2014~ 04:3~ 5.68e5 3.1.1     i386   ming~ rJava   0.9-6   US
##  9 30245 2014~ 04:1~ 5.27e5 3.1.1     i386   ming~ LPCM    0.44-8  US
## 10 30354 2014~ 04:3~ 1.76e6 3.1.1     i386   ming~ mgcv    1.8-1   US
## # ... with 1,578 more rows, and 1 more variable: ip_id <int>
```

Edit your previous call to filter() to instead return rows corresponding to users in "IN" (India) running an R version that is less than or equal to "3.0.2". The up arrow on your keyboard may come in handy here. Don't forget your double quotes!

```
filter(cran, r_version <= "3.0.2", country == "IN")
```

```
## # A tibble: 4,139 x 11
##        X date  time    size r_version r_arch r_os  package version country
##    <int> <chr> <chr>  <int> <chr>     <chr>  <chr> <chr>   <chr>   <chr>
##  1   348 2014~ 00:4~ 1.02e7 3.0.0     x86_64 ming~ BH      1.54.0~ IN
##  2  9990 2014~ 02:1~ 3.97e5 3.0.2     x86_64 linu~ equate~ 1.1     IN
##  3  9991 2014~ 02:1~ 1.19e5 3.0.2     x86_64 linu~ ggdend~ 0.1-14  IN
##  4  9992 2014~ 02:1~ 8.18e4 3.0.2     x86_64 linu~ dfcrm   0.2-2   IN
##  5 10022 2014~ 02:1~ 1.56e6 2.15.0    x86_64 ming~ RcppAr~ 0.4.32~ IN
##  6 10023 2014~ 02:1~ 1.18e6 2.15.1    i686   linu~ foreca~ 5.4     IN
##  7 10189 2014~ 02:3~ 9.09e5 3.0.2     x86_64 linu~ editru~ 2.7.2   IN
##  8 10199 2014~ 02:3~ 1.78e5 3.0.2     x86_64 linu~ energy  1.6.1   IN
##  9 10200 2014~ 02:3~ 5.18e4 3.0.2     x86_64 linu~ ENmisc  1.2-7   IN
## 10 10201 2014~ 02:3~ 6.52e4 3.0.2     x86_64 linu~ entropy 1.2.0   IN
## # ... with 4,129 more rows, and 1 more variable: ip_id <int>
```

Our last two calls to filter() requested all rows for which some condition AND another condition were TRUE. We can also request rows for which EITHER one condition OR another condition are TRUE. For example, filter(cran, country == "US" | country == "IN") will gives us all rows for which the country variable equals either "US" or "IN". Give it a go.

```
filter(cran, country == "US" | country == "IN")
```

```
## # A tibble: 95,283 x 11
##        X date  time    size r_version r_arch r_os  package version country
##    <int> <chr> <chr>  <int> <chr>     <chr>  <chr> <chr>   <chr>   <chr>
##  1     1 2014~ 00:5~ 8.06e4 3.1.0     x86_64 ming~ htmlto~ 0.2.4   US
##  2     2 2014~ 00:5~ 3.22e5 3.1.0     x86_64 ming~ tseries 0.10-32 US
##  3     3 2014~ 00:4~ 7.48e5 3.1.0     x86_64 linu~ party   1.0-15  US
##  4     4 2014~ 00:4~ 6.06e5 3.1.0     x86_64 linu~ Hmisc   3.14-4  US
##  5     6 2014~ 00:4~ 7.77e4 3.1.0     x86_64 linu~ random~ 4.6-7   US
##  6     7 2014~ 00:4~ 3.94e5 3.1.0     x86_64 linu~ plyr    1.8.1   US
##  7     8 2014~ 00:4~ 2.82e4 3.0.2     x86_64 linu~ whisker 0.3-2   US
##  8    10 2014~ 00:1~ 2.21e6 3.0.2     x86_64 linu~ hfligh~ 0.1     US
##  9    11 2014~ 00:1~ 5.27e5 3.0.2     x86_64 linu~ LPCM    0.44-8  US
## 10    12 2014~ 00:1~ 2.35e6 2.14.1    x86_64 linu~ ggplot2 1.0.0   US
## # ... with 95,273 more rows, and 1 more variable: ip_id <int>
```

Now, use filter() to fetch all rows for which size is strictly greater than (>) 100500 (no quotes, since size is numeric) AND r_os equals "linux-gnu". Hint: You are passing three arguments to filter(): the name of the dataset, the first condition, and the second condition.

```
filter(cran, !is.na(r_version))
```

```
## # A tibble: 207,205 x 11
##        X date  time    size r_version r_arch r_os  package version country
##    <int> <chr> <chr>  <int> <chr>     <chr>  <chr> <chr>   <chr>   <chr>
##  1     1 2014~ 00:5~ 8.06e4 3.1.0     x86_64 ming~ htmlto~ 0.2.4   US
##  2     2 2014~ 00:5~ 3.22e5 3.1.0     x86_64 ming~ tseries 0.10-32 US
##  3     3 2014~ 00:4~ 7.48e5 3.1.0     x86_64 linu~ party   1.0-15  US
##  4     4 2014~ 00:4~ 6.06e5 3.1.0     x86_64 linu~ Hmisc   3.14-4  US
##  5     5 2014~ 00:4~ 7.98e4 3.0.2     x86_64 linu~ digest  0.6.4   CA
##  6     6 2014~ 00:4~ 7.77e4 3.1.0     x86_64 linu~ random~ 4.6-7   US
##  7     7 2014~ 00:4~ 3.94e5 3.1.0     x86_64 linu~ plyr    1.8.1   US
##  8     8 2014~ 00:4~ 2.82e4 3.0.2     x86_64 linu~ whisker 0.3-2   US
##  9    10 2014~ 00:1~ 2.21e6 3.0.2     x86_64 linu~ hfligh~ 0.1     US
## 10    11 2014~ 00:1~ 5.27e5 3.0.2     x86_64 linu~ LPCM    0.44-8  US
## # ... with 207,195 more rows, and 1 more variable: ip_id <int>
```

Sometimes we want to order the rows of a dataset according to the values of a particular variable. This is the job of arrange().

To see how arrange() works, let's first take a subset of cran. select() all columns from size through ip_id and store the result in cran2.

```
cran2 <- select(cran, size:ip_id)
```

Now, to order the ROWS of cran2 so that ip_id is in ascending order (from small to large), type arrange(cran2, ip_id). You may want to make your console wide enough so that you can see ip_id, which is the last column.

```
arrange(cran2, ip_id)
```

```
## # A tibble: 225,468 x 8
##      size r_version r_arch r_os    package     version country ip_id
##     <int> <chr>     <chr>  <chr>   <chr>       <chr>   <chr>   <int>
##  1  80589 3.1.0     x86_64 mingw32 htmltools   0.2.4   US          1
##  2 180562 3.0.2     x86_64 mingw32 yaml        2.1.13  US          1
##  3 190120 3.1.0     i386   mingw32 babel       0.2-6   US          1
```

```
##  4 321767 3.1.0     x86_64 mingw32      tseries     0.10-32 US          2
##  5  52281 3.0.3     x86_64 darwin10.8.0 quadprog    1.5-5   US          2
##  6 876702 3.1.0     x86_64 linux-gnu    zoo         1.7-11  US          2
##  7 321764 3.0.2     x86_64 linux-gnu    tseries     0.10-32 US          2
##  8 876702 3.1.0     x86_64 linux-gnu    zoo         1.7-11  US          2
##  9 321768 3.1.0     x86_64 mingw32      tseries     0.10-32 US          2
## 10 784093 3.1.0     x86_64 linux-gnu    strucchange 1.5-0   US          2
## # ... with 225,458 more rows
```

To do the same, but in descending order, change the second argument to desc(ip_id), where desc() stands for 'descending'. Go ahead.

```
arrange(cran2, desc(ip_id))
```

```
## # A tibble: 225,468 x 8
##       size r_version r_arch r_os         package      version country ip_id
##      <int> <chr>     <chr>  <chr>        <chr>        <chr>   <chr>   <int>
##  1    5933 <NA>      <NA>   <NA>         CPE          1.4.2   CN      13859
##  2  569241 3.1.0     x86_64 mingw32      multcompView 0.1-5   US      13858
##  3  228444 3.1.0     x86_64 mingw32      tourr        0.5.3   NZ      13857
##  4  308962 3.1.0     x86_64 darwin13.1.0 ctv          0.7-9   CN      13856
##  5  950964 3.0.3     i386   mingw32      knitr        1.6     CA      13855
##  6   80185 3.0.3     i386   mingw32      htmltools    0.2.4   CA      13855
##  7 1431750 3.0.3     i386   mingw32      shiny        0.10.0  CA      13855
##  8 2189695 3.1.0     x86_64 mingw32      RMySQL       0.9-3   US      13854
##  9 4818024 3.1.0     i386   mingw32      igraph       0.7.1   US      13853
## 10  197495 3.1.0     x86_64 mingw32      coda         0.16-1  US      13852
## # ... with 225,458 more rows
```

We can also arrange the data according to the values of multiple variables. For example, arrange(cran2, package, ip_id) will first arrange by package names (ascending alphabetically), then by ip_id. This means that if there are multiple rows with the same value for package, they will be sorted by ip_id (ascending numerically). Try arrange(cran2, package, ip_id) now.

```
arrange(cran2, package, ip_id)
```

```
## # A tibble: 225,468 x 8
##     size r_version r_arch r_os         package version country ip_id
##    <int> <chr>     <chr>  <chr>        <chr>   <chr>   <chr>   <int>
##  1 71677 3.0.3     x86_64 darwin10.8.0 A3      0.9.2   CN       1003
##  2 71672 3.1.0     x86_64 linux-gnu    A3      0.9.2   US       1015
##  3 71677 3.1.0     x86_64 mingw32      A3      0.9.2   IN       1054
##  4 70438 3.0.1     x86_64 darwin10.8.0 A3      0.9.2   CN       1513
##  5 71677 <NA>      <NA>   <NA>         A3      0.9.2   BR       1526
##  6 71892 3.0.2     x86_64 linux-gnu    A3      0.9.2   IN       1542
##  7 71677 3.1.0     x86_64 linux-gnu    A3      0.9.2   ZA       2925
##  8 71672 3.1.0     x86_64 mingw32      A3      0.9.2   IL       3889
##  9 71677 3.0.3     x86_64 mingw32      A3      0.9.2   DE       3917
## 10 71672 3.1.0     x86_64 mingw32      A3      0.9.2   US       4219
## # ... with 225,458 more rows
```

Arrange cran2 by the following three variables, in this order: country (ascending), r_version (descending), and ip_id (ascending).

```
arrange(cran2, country, desc(r_version), ip_id)
```

```
## # A tibble: 225,468 x 8
##       size r_version r_arch r_os      package       version   country ip_id
```

```
##      <int> <chr>     <chr>  <chr>     <chr>         <chr>     <chr>   <int>
##  1 1556858 3.1.1     i386   mingw32   RcppArmadillo 0.4.320.0 A1       2843
##  2 1823512 3.1.0     x86_64 linux-gnu mgcv          1.8-1     A1       2843
##  3   15732 3.1.0     i686   linux-gnu grnn          0.1.0     A1       3146
##  4 3014840 3.1.0     x86_64 mingw32   Rcpp          0.11.2    A1       3146
##  5  660087 3.1.0     i386   mingw32   xts           0.9-7     A1       3146
##  6  522261 3.1.0     i386   mingw32   FNN           1.1       A1       3146
##  7  522263 3.1.0     i386   mingw32   FNN           1.1       A1       3146
##  8 1676627 3.1.0     x86_64 linux-gnu rgeos         0.3-5     A1       3146
##  9 2118530 3.1.0     x86_64 linux-gnu spacetime     1.1-0     A1       3146
## 10 2217180 3.1.0     x86_64 mingw32   gstat         1.0-19    A1       3146
## # ... with 225,458 more rows
```

To illustrate the next major function in dplyr, let's take another subset of our original data. Use select() to grab 3 columns from cran – ip_id, package, and size (in that order) – and store the result in a new variable called cran3.

```
cran3 <- select(cran, ip_id, package, size)
```

It's common to create a new variable based on the value of one or more variables already in a dataset. The mutate() function does exactly this. The size variable represents the download size in bytes, which are units of computer memory. These days, megabytes (MB) are a more common unit of measurement. One megabyte is equal to 2^20 bytes. That's 2 to the power of 20, which is approximately one million bytes!

We want to add a column called size_mb that contains the download size in megabytes. Here's the code to do it: mutate(cran3, size_mb = size / 2^20)

```
mutate(cran3, size_mb = size / 2^20)
```

```
## # A tibble: 225,468 x 4
##    ip_id package         size size_mb
##    <int> <chr>          <int>   <dbl>
##  1     1 htmltools      80589 0.0769
##  2     2 tseries       321767 0.307
##  3     3 party         748063 0.713
##  4     3 Hmisc         606104 0.578
##  5     4 digest         79825 0.0761
##  6     3 randomForest   77681 0.0741
##  7     3 plyr          393754 0.376
##  8     5 whisker        28216 0.0269
##  9     6 Rcpp            5928 0.00565
## 10     7 hflights     2206029 2.10
## # ... with 225,458 more rows
```

One very nice feature of mutate() is that you can use the value computed for your second column (size_mb) to create a third column, all in the same line of code. To see this in action, repeat the exact same command as above, except add a third argument creating a column that is named size_gb and equal to size_mb / 2^10.

```
mutate(cran3, size_mb = size / 2^20, size_gb = size_mb / 2^10)
```

```
## # A tibble: 225,468 x 5
##    ip_id package         size size_mb    size_gb
##    <int> <chr>          <int>   <dbl>      <dbl>
##  1     1 htmltools      80589 0.0769  0.0000751
##  2     2 tseries       321767 0.307   0.000300
##  3     3 party         748063 0.713   0.000697
##  4     3 Hmisc         606104 0.578   0.000564
##  5     4 digest         79825 0.0761  0.0000743
```

```
##  6     3 randomForest   77681 0.0741  0.0000723
##  7     3 plyr          393754 0.376   0.000367
##  8     5 whisker        28216 0.0269  0.0000263
##  9     6 Rcpp            5928 0.00565 0.00000552
## 10     7 hflights     2206029 2.10    0.00205
## # ... with 225,458 more rows
```

Let's try one more for practice. Pretend we discovered a glitch in the system that provided the original values for the size variable. All of the values in cran3 are 1000 bytes less than they should be. Using cran3, create just one new column called correct_size that contains the correct size.

```
mutate(cran3, correct_size = size + 1000)
```

```
## # A tibble: 225,468 x 4
##    ip_id package         size correct_size
##    <int> <chr>          <int>        <dbl>
##  1     1 htmltools      80589        81589
##  2     2 tseries       321767       322767
##  3     3 party         748063       749063
##  4     3 Hmisc         606104       607104
##  5     4 digest         79825        80825
##  6     3 randomForest   77681        78681
##  7     3 plyr          393754       394754
##  8     5 whisker        28216        29216
##  9     6 Rcpp            5928         6928
## 10     7 hflights     2206029      2207029
## # ... with 225,458 more rows
```

The last of the five core dplyr verbs, summarize(), collapses the dataset to a single row. Let's say we're interested in knowing the average download size. summarize(cran, avg_bytes = mean(size)) will yield the mean value of the size variable. Here we've chosen to label the result 'avg_bytes', but we could have named it anything. Give it a try.

```
summarize(cran, avg_bytes = mean(size))
```

```
## # A tibble: 1 x 1
##   avg_bytes
##       <dbl>
## 1   844086.
```