10 Dplyr tricks

for @WeAreRLadies

Suzan Baert

Tip 1: Are you often selecting the same columns over and over again?

You can make a vector of pre-identified columns once, and then refer to them using `one_of()` or even shorter with `!!`.

```
library(dplyr)

cols <- c("mpg", "cyl", "gear")

mtcars %>%
  select(!!cols)
```

```
##                      mpg cyl gear
## Mazda RX4           21.0   6    4
## Mazda RX4 Wag       21.0   6    4
## Datsun 710          22.8   4    4
## Hornet 4 Drive      21.4   6    3
## Hornet Sportabout   18.7   8    3
## Valiant             18.1   6    3
## Duster 360          14.3   8    3
## Merc 240D           24.4   4    4
## Merc 230            22.8   4    4
## Merc 280            19.2   6    4
## Merc 280C           17.8   6    4
## Merc 450SE          16.4   8    3
## Merc 450SL          17.3   8    3
## Merc 450SLC         15.2   8    3
## Cadillac Fleetwood  10.4   8    3
## Lincoln Continental 10.4   8    3
## Chrysler Imperial   14.7   8    3
## Fiat 128            32.4   4    4
```

Tip 2: Select columns via regex

If you have matching patterns you can use `starts_with()`, `contains()` or `ends_with()`. But what if your pattern isn't thàt exact? Simple: enter regex into `matches()`

```
library(dplyr)

iris %>%
  select(matches("S.+th"))
```

```
##     Sepal.Length Sepal.Width
## 1            5.1         3.5
## 2            4.9         3.0
## 3            4.7         3.2
## 4            4.6         3.1
## 5            5.0         3.6
## 6            5.4         3.9
## 7            4.6         3.4
## 8            5.0         3.4
## 9            4.4         2.9
## 10           4.9         3.1
## 11           5.4         3.7
## 12           4.8         3.4
## 13           4.8         3.0
## 14           4.3         3.0
## 15           5.8         4.0
## 16           5.7         4.4
## 17           5.4         3.9
## 18           5.1         3.5
## 19           5.7         3.8
## 20           5.1         3.8
```

Tip 3: Reordering your columns

If you just want to bring one or more columns to the front, you can use `everything()` to add all the remaining columns.

```
library(dplyr)

iris %>%
  select(Species, everything())
```

```
##        Species Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1       setosa          5.1         3.5          1.4         0.2
## 2       setosa          4.9         3.0          1.4         0.2
## 3       setosa          4.7         3.2          1.3         0.2
## 4       setosa          4.6         3.1          1.5         0.2
## 5       setosa          5.0         3.6          1.4         0.2
## 6       setosa          5.4         3.9          1.7         0.4
## 7       setosa          4.6         3.4          1.4         0.3
## 8       setosa          5.0         3.4          1.5         0.2
## 9       setosa          4.4         2.9          1.4         0.2
## 10      setosa          4.9         3.1          1.5         0.1
## 11      setosa          5.4         3.7          1.5         0.2
## 12      setosa          4.8         3.4          1.6         0.2
## 13      setosa          4.8         3.0          1.4         0.1
## 14      setosa          4.3         3.0          1.1         0.1
## 15      setosa          5.8         4.0          1.2         0.2
## 16      setosa          5.7         4.4          1.5         0.4
## 17      setosa          5.4         3.9          1.3         0.4
## 18      setosa          5.1         3.5          1.4         0.3
## 19      setosa          5.7         3.8          1.7         0.3
## 20      setosa          5.1         3.8          1.5         0.3
```

Tip 4: Renaming all variables in one go

One command to get them all in lower case. And one more to replace those dots with underscores...

```
library(dplyr)
library(stringr)

iris %>%
  rename_all(tolower) %>%
  rename_all(~str_replace_all(., "\\.", "_"))
```

```
##     sepal_length sepal_width petal_length petal_width    species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
```

Tip 5: Cleaning up your observations in one go

The `select_all/if/at` and `rename_all/if/at` functions will only modify the variable names, not the observations. If you want to change those, use the `mutate` variant!

```
library(dplyr)
library(stringr)

storms %>%
  select(name, year, status) %>%
  mutate_all(tolower) %>%
  mutate_all(~str_replace_all(., " ", "_"))
```

```
## # A tibble: 10,010 x 3
##    name  year  status
##    <chr> <chr> <chr>
##  1 amy   1975  tropical_depression
##  2 amy   1975  tropical_depression
##  3 amy   1975  tropical_depression
##  4 amy   1975  tropical_depression
##  5 amy   1975  tropical_depression
##  6 amy   1975  tropical_depression
##  7 amy   1975  tropical_depression
##  8 amy   1975  tropical_depression
##  9 amy   1975  tropical_storm
## 10 amy   1975  tropical_storm
## # ... with 10,000 more rows
```

Tip 6: Finding the 5 highest/lowest values

You can use `top_n` to find the 5 cars with the highest horsepower without ordering them first.

```
library(dplyr)

mtcars %>%
  top_n(5, hp)
```

```
##    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## 1 14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
## 2 14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
## 3 13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
## 4 15.8   8  351 264 4.22 3.170 14.50  0  1    5    4
## 5 15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
```

Tip 7: Adding the amount of observations

You can add the amount of observations without summarising them yourself.
And if you don't like the default column name `n`, just change it again with a `rename()` statement.

```
library(dplyr)

mtcars %>%
  select(-(drat:vs)) %>%
  add_count(cyl) %>% rename(n_cyl = n) %>%
  add_count(am) %>% rename(n_am = n)
```

```
## # A tibble: 32 x 9
##      mpg   cyl  disp    hp    am  gear  carb n_cyl  n_am
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int> <int>
##  1  21       6  160    110     1     4     4     7    13
##  2  21       6  160    110     1     4     4     7    13
##  3  22.8     4  108     93     1     4     1    11    13
##  4  21.4     6  258    110     0     3     1     7    19
##  5  18.7     8  360    175     0     3     2    14    19
##  6  18.1     6  225    105     0     3     1     7    19
##  7  14.3     8  360    245     0     3     4    14    19
##  8  24.4     4  147.    62     0     4     2    11    19
##  9  22.8     4  141.    95     0     4     2    11    19
## 10  19.2     6  168.   123     0     4     4     7    19
## # ... with 22 more rows
```

Tip 8: Making new discrete variables

`case_when()` can be a very powerful tool to make new discrete variables based on other columns.

```
starwars %>%
  select(name, species, homeworld, birth_year, hair_color) %>%
  mutate(new_group = case_when(
      species == "Droid" ~ "Robot",
      homeworld == "Tatooine" & hair_color == "blond" ~ "Blond Tatooinian",
      homeworld == "Tatooine" ~ "Other Tatooinian",
      hair_color == "blond" ~ "Blond non-Tatooinian",
      TRUE ~ "Other Human"))
```

```
## # A tibble: 87 x 6
##    name               species homeworld birth_year hair_color    new_group
##    <chr>              <chr>   <chr>          <dbl> <chr>         <chr>
##  1 Luke Skywalker     Human   Tatooine        19   blond         Blond Ta~
##  2 C-3PO              Droid   Tatooine       112   <NA>          Robot
##  3 R2-D2              Droid   Naboo           33   <NA>          Robot
##  4 Darth Vader        Human   Tatooine        41.9 none          Other Ta~
##  5 Leia Organa        Human   Alderaan        19   brown         Other Hu~
##  6 Owen Lars          Human   Tatooine        52   brown, grey   Other Ta~
##  7 Beru Whitesun lars Human   Tatooine        47   brown         Other Ta~
##  8 R5-D4              Droid   Tatooine        NA   <NA>          Robot
##  9 Biggs Darklighter  Human   Tatooine        24   black         Other Ta~
## 10 Obi-Wan Kenobi     Human   Stewjon         57   auburn, white Other Hu~
## # ... with 77 more rows
```

Tip 9: going rowwise...

Mutating with aggregate functions by default will take the average/sum/... of the entire column. Via adding `rowwise()` you can aggregate within an observatioin.

```
iris %>%
  select(contains("Length")) %>%
  rowwise() %>%
  mutate(avg_length = mean(c(Petal.Length, Sepal.Length)))
```

```
## Source: local data frame [150 x 3]
## Groups: <by row>
##
## # A tibble: 150 x 3
##    Sepal.Length Petal.Length avg_length
##           <dbl>        <dbl>      <dbl>
##  1          5.1          1.4       3.25
##  2          4.9          1.4       3.15
##  3          4.7          1.3       3
##  4          4.6          1.5       3.05
##  5          5            1.4       3.2
##  6          5.4          1.7       3.55
##  7          4.6          1.4       3
##  8          5            1.5       3.25
##  9          4.4          1.4       2.9
## 10          4.9          1.5       3.2
## # ... with 140 more rows
```

Tip 10: Changing your column names after summarise_if

If you've used the `summarise_all`, `summarise_if` and `summarise_at` variants before, you know that the variable name by default does not get changed.
If you do what a modified name, you can wrap your function inside `funs()` and add a tag that will be added to the variable name.

```
iris %>%
  summarise_if(is.numeric, funs(avg = mean))
```

```
##   Sepal.Length_avg Sepal.Width_avg Petal.Length_avg Petal.Width_avg
## 1         5.843333        3.057333            3.758        1.199333
```