dplyr package

## dplyr package

Summarized by SY Ohn

## References

-> <https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html> -> <https://dplyr.tidyverse.org/index.html>

dplyr is a powerful R-package to preprocessing data written by H. Wickham.

## functions in dplyr package:

### Handling rows

filter(): Filter rows with condition = {base} subset slice(): Filter rows with position = {base} subset arrange(): Re-order or arrange rows = {base} order

### Handling columns

select(): Select columns ={base} subset

* select(df, starts\_with()): Select columns that start with a prefix
* select(df, ends\_with()): Select columns that end with a prefix
* select(df, contains()): Select columns that contain a character string
* select(df, matchs()): Select columns that match a regular expression
* select(df, one\_of()): Select columns that are from a group of names
* select(df, num\_range()): Select columns from num\_range a to n with a prefix

rename(): Rename column name = {reshape} rename

distinct(): Extract distinct(unique) rows = {base} unique

### Creating and Adding new columns

mutate(): Create(add) new columns.mutate() allows you to refer to columns that you’ve just created. = {base} transform  
transmute(): Create(add) new columns. transmute() only keeps the new columns. = {base} transform

### Summarise

summarise(): Summarise values = {base} summary

### Sampling

sample\_n(): Random sample rows for a fixed number = {base} sample

sample\_frac(): Random sample rows for a fixed fraction = {base} sample

We will use the dataset starwars to demonsrate dplyr package.

This dataset contains 87 characters and comes from the Star Wars API, and is documented in ?starwars.

Note that starwars is a **tibble, a modern reimagining of the data frame.** -> <https://tibble.tidyverse.org> You can convert data frames to tibbles with as\_tibble()

library(dplyr)

## Warning: 패키지 'dplyr'는 R 버전 4.1.3에서 작성되었습니다

##   
## 다음의 패키지를 부착합니다: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

data(starwars)  
  
sw1 <- starwars  
# str(sw1)  
anyNA(sw1)

## [1] TRUE

# head(sw1)  
dim(sw1)

## [1] 87 14

### Filter rows with filter()

Subsetting rows

temp <- filter(sw1, skin\_color == "light", eye\_color == "brown")  
temp <- filter(sw1, (skin\_color == "light" | eye\_color == "brown") & height >= 170)  
# == starwars[starwars$skin\_color == "light" & starwars$eye\_color == "brown", ]  
head(temp)

## # A tibble: 6 x 14  
## name height mass hair\_color skin\_color eye\_color birth\_year sex gender  
## <chr> <int> <dbl> <chr> <chr> <chr> <dbl> <chr> <chr>   
## 1 Owen Lars 178 120 brown, gr~ light blue 52 male mascu~  
## 2 Biggs Da~ 183 84 black light brown 24 male mascu~  
## 3 Han Solo 180 80 brown fair brown 29 male mascu~  
## 4 Boba Fett 183 78.2 black fair brown 31.5 male mascu~  
## 5 Lando Ca~ 177 79 black dark brown 31 male mascu~  
## 6 Lobot 175 79 none light blue 37 male mascu~  
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,  
## # vehicles <list>, starships <list>

### Arrange rows with arrange()

Reordering rows

temp <- arrange(sw1, height, desc(mass))   
# sorting key height(ascending), mass(descending)

Choose rows using their position with slice()

temp <- slice(sw1, 5:10)  
temp <- slice\_head(sw1, n = 3) # slice\_tail()  
temp <- slice\_sample(sw1, n = 5) # random sampling, record number  
temp <- slice\_sample(sw1, prop = 0.1) # set proportion  
  
temp <- slice\_sample(sw1, n = 10, replace = TRUE) # boot strapping  
temp <- slice\_max(sw1, height, n = 3)   
# 3 samples with max height, slice\_min()

### Select columns with select()

temp <- select(sw1, hair\_color, skin\_color, eye\_color)  
temp <- select(sw1, hair\_color:eye\_color)  
temp <- select(sw1, !(hair\_color:eye\_color))  
temp <- select(sw1, ends\_with("color"))   
# Also, starts\_with(), ends\_with(), matches() and contains()  
temp <- select(sw1, home\_world = homeworld)

But because select() drops all the variables not explicitly mentioned, it’s not that useful. Instead, use rename():

temp <- rename(sw1, home\_world = homeworld)

### Add new columns with mutate()

temp <- mutate(sw1, height\_m = height / 100) # cm -> m conversion  
temp <- select(temp, height\_m, height, everything())   
# everything(): matches all

dplyr::mutate() is similar to the base transform(), but allows you to refer to columns that you’ve just created:

temp <- mutate(  
 temp,  
 height\_m = height / 100,  
 BMI = mass / (height\_m^2)  
 )  
select(temp, BMI, everything())

## # A tibble: 87 x 16  
## BMI height\_m height name mass hair\_color skin\_color eye\_color birth\_year  
## <dbl> <dbl> <int> <chr> <dbl> <chr> <chr> <chr> <dbl>  
## 1 26.0 1.72 172 Luke ~ 77 blond fair blue 19   
## 2 26.9 1.67 167 C-3PO 75 <NA> gold yellow 112   
## 3 34.7 0.96 96 R2-D2 32 <NA> white, bl~ red 33   
## 4 33.3 2.02 202 Darth~ 136 none white yellow 41.9  
## 5 21.8 1.5 150 Leia ~ 49 brown light brown 19   
## 6 37.9 1.78 178 Owen ~ 120 brown, gr~ light blue 52   
## 7 27.5 1.65 165 Beru ~ 75 brown light blue 47   
## 8 34.0 0.97 97 R5-D4 32 <NA> white, red red NA   
## 9 25.1 1.83 183 Biggs~ 84 black light brown 24   
## 10 23.2 1.82 182 Obi-W~ 77 auburn, w~ fair blue-gray 57   
## # ... with 77 more rows, and 7 more variables: sex <chr>, gender <chr>,  
## # homeworld <chr>, species <chr>, films <list>, vehicles <list>,  
## # starships <list>

If you only want to keep the new variables, use transmute():

transmute(temp,   
 height\_m = height / 100,  
 BMI = mass / (height\_m^2)  
 )

## # A tibble: 87 x 2  
## height\_m BMI  
## <dbl> <dbl>  
## 1 1.72 26.0  
## 2 1.67 26.9  
## 3 0.96 34.7  
## 4 2.02 33.3  
## 5 1.5 21.8  
## 6 1.78 37.9  
## 7 1.65 27.5  
## 8 0.97 34.0  
## 9 1.83 25.1  
## 10 1.82 23.2  
## # ... with 77 more rows

### Change column order with relocate()

Use a similar syntax as select() to move blocks of columns at once

temp <- relocate(sw1, sex:homeworld, .before = height) # .after

Summarise values with summarise()

temp <- summarise(sw1, height = mean(height, na.rm = TRUE))

### Group rows with group\_by()

Most data operations are done on groups defined by variables. group\_by() takes an existing tbl and converts it into a grouped tbl where operations are performed “by group”. ungroup() removes grouping.

a1 <- group\_by(sw1, species, sex) # group by species and sex values   
a2 <- select(a1, height, mass)

## Adding missing grouping variables: `species`, `sex`

a3 <- summarise(a2,  
 height = mean(height, na.rm = TRUE),  
 mass = mean(mass, na.rm = TRUE)  
)

## `summarise()` has grouped output by 'species'. You can override using the  
## `.groups` argument.

### Commonalities

he dplyr API is functional in the sense that function calls don’t have side-effects. You must always save their results. You either have to do it step-by-step:

a1 <- group\_by(sw1, species, sex)  
a2 <- select(a1, height, mass)

## Adding missing grouping variables: `species`, `sex`

a3 <- summarise(a2,  
 height = mean(height, na.rm = TRUE),  
 mass = mean(mass, na.rm = TRUE)  
 )

## `summarise()` has grouped output by 'species'. You can override using the  
## `.groups` argument.

Or if you don’t want to name the intermediate results, you need to wrap the function calls inside each other:

summarise(  
 select(  
 group\_by(starwars, species, sex),  
 height, mass  
 ),  
 height = mean(height, na.rm = TRUE),  
 mass = mean(mass, na.rm = TRUE)  
)

## Adding missing grouping variables: `species`, `sex`  
## `summarise()` has grouped output by 'species'. You can override using the  
## `.groups` argument.

## # A tibble: 41 x 4  
## # Groups: species [38]  
## species sex height mass  
## <chr> <chr> <dbl> <dbl>  
## 1 Aleena male 79 15   
## 2 Besalisk male 198 102   
## 3 Cerean male 198 82   
## 4 Chagrian male 196 NaN   
## 5 Clawdite female 168 55   
## 6 Droid none 131. 69.8  
## 7 Dug male 112 40   
## 8 Ewok male 88 20   
## 9 Geonosian male 183 80   
## 10 Gungan male 209. 74   
## # ... with 31 more rows

### Using the pipe:

dplyr provides the %>% operator (pipe) from magrittr. x %>% f(y) turns into f(x, y) (reading the pipe operator as “then”)

starwars %>%  
 group\_by(species, sex) %>%  
 select(height, mass) %>%  
 summarise(  
 height = mean(height, na.rm = TRUE),  
 mass = mean(mass, na.rm = TRUE)  
 )

## Adding missing grouping variables: `species`, `sex`  
## `summarise()` has grouped output by 'species'. You can override using the  
## `.groups` argument.

## # A tibble: 41 x 4  
## # Groups: species [38]  
## species sex height mass  
## <chr> <chr> <dbl> <dbl>  
## 1 Aleena male 79 15   
## 2 Besalisk male 198 102   
## 3 Cerean male 198 82   
## 4 Chagrian male 196 NaN   
## 5 Clawdite female 168 55   
## 6 Droid none 131. 69.8  
## 7 Dug male 112 40   
## 8 Ewok male 88 20   
## 9 Geonosian male 183 80   
## 10 Gungan male 209. 74   
## # ... with 31 more rows

### rename() to rename columns

### distinct() values of a variable

temp <- rename(sw1, male\_female = sex)  
temp <- distinct(sw1, sex)  
temp <- distinct(sw1, sex, species)

### Sampling

temp <- sample\_n(sw1, 10)  
temp <- sample\_n(sw1, 10, replace = TRUE) # bootstrapping  
  
# replace NA in height column with mean for height variable  
mean\_height <- mean(sw1$height, na.rm = TRUE)  
sw1$height <- ifelse(is.na(sw1$height), mean\_height, sw1$height)  
  
# sampling with probability  
temp <- sample\_n(sw1, 10, weight = height, replace = TRUE)   
  
# boot strapping   
temp <- sample\_frac(sw1, 1.0, replace = TRUE, weight = 1 / height)

### - End -