Data Transformation using dplyr and forcats

Kamarul Imran Musa

2017-10-27

Contents

1 Data transformation (data munging or data wrangling)

1.1 Definition of data wrangling

Data munging or data wrangling is loosely the process of manually converting or mapping data from one "raw" form into another format that allows for more convenient consumption of the data with the help of semi-automated tools.

Refer https://community.modeanalytics.com/sql/tutorial/data-wrangling-with-sql/

1.2 Package: dplyr

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges

Refer https://github.com/tidyverse/dplyr

1.3 Data wrangling using dplyr

When we communicate with data, common procedures include

1. reduce the size of dataset by selecting certain variables
2. create new variables from existing variables
3. sort observation of a variable
4. group observations that fulfil certain criteria
5. reduce variable to groups to in order to estimate summary statistic

2 Using dplyr package

For the procedures listed above, the corresponding **dplyr** functions are

1. reduce the size of dataset by selecting certain variables : `select`
2. create new variables from existing variables : `mutate`
3. sort observation of a variable : `arrange`
4. group observations that fulfil certain criteria : `filter`
5. reduce variable to groups to in order to estimate summary statistic : `summarize` + `group_by`

2.1 Preparation and data

2.1.1 Working directory and data format

Make sure, if you deal with your own data:

1. that you have set your working directory
2. you have read the data using the correct package::function

2.1.2 Training data

To replicate the examples demonstrated on *tidyverse* website https://github.com/tidyverse/dplyr we will use similar dataset or datasets Grammar of variables

The data name is `starwars` coming with **dplyr** package. This data comes from SWAPI, the Star Wars API, http://swapi.co/.

A tibble with 87 rows and 13 variables:

```
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
```

```
names(starwars)
```

```
##  [1] "name"       "height"     "mass"       "hair_color" "skin_color"
##  [6] "eye_color"  "birth_year" "gender"     "homeworld"  "species"
## [11] "films"      "vehicles"   "starships"
```

Let us examine the first 10 observations of the data

```
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>
```

2.2 Function: select and mutate

2.2.1 Select

To select some columns of all columns

The are 13 variables. Now, we want to select only a few of them:

1. name
2. height (cm)
3. mass (kg)
4. gender

```r
mysw <- select(starwars, name, gender, height, mass)
head(mysw, 10)
```

```
## # A tibble: 10 x 4
##                  name gender height  mass
##                 <chr>  <chr>  <int> <dbl>
##  1     Luke Skywalker   male    172    77
##  2              C-3PO   <NA>    167    75
##  3              R2-D2   <NA>     96    32
##  4        Darth Vader   male    202   136
##  5        Leia Organa female    150    49
##  6          Owen Lars   male    178   120
##  7 Beru Whitesun lars female    165    75
##  8              R5-D4   <NA>     97    32
##  9  Biggs Darklighter   male    183    84
## 10     Obi-Wan Kenobi   male    182    77
```

2.2.2 Mutate

Create a new variable - mutate

We want to create a new variable `bmi` which equals mass in kg divided by squared height (in meter)

$$\frac{kg}{m^2}$$

```r
mysw <- mutate(mysw, bmi = mass/(height/100)^2)
mysw
```

```
## # A tibble: 87 x 5
##                  name gender height  mass      bmi
##                 <chr>  <chr>  <int> <dbl>    <dbl>
##  1     Luke Skywalker   male    172    77 26.02758
##  2              C-3PO   <NA>    167    75 26.89232
##  3              R2-D2   <NA>     96    32 34.72222
##  4        Darth Vader   male    202   136 33.33007
##  5        Leia Organa female    150    49 21.77778
##  6          Owen Lars   male    178   120 37.87401
##  7 Beru Whitesun lars female    165    75 27.54821
##  8              R5-D4   <NA>     97    32 34.00999
##  9  Biggs Darklighter   male    183    84 25.08286
## 10     Obi-Wan Kenobi   male    182    77 23.24598
## # ... with 77 more rows
```

2.3 arrange and filter

2.3.1 arrange

This will sort the observation based on the values of a variable. Let see from the biggest bmi to the lowest bmi, first.

```r
mysw <- arrange(mysw, desc(bmi))
head(mysw)
```

```
## # A tibble: 6 x 5
##                    name        gender height  mass       bmi
##                   <chr>         <chr>  <int> <dbl>     <dbl>
## 1 Jabba Desilijic Tiure hermaphrodite    175  1358 443.42857
## 2              Dud Bolt          male     94    45  50.92802
## 3                  Yoda          male     66    17  39.02663
## 4             Owen Lars          male    178   120  37.87401
## 5                 IG-88          none    200   140  35.00000
## 6                 R2-D2          <NA>     96    32  34.72222
```

Then, from the lowest to the biggest bmi

```
mysw <- arrange(mysw, bmi)
head(mysw)
```

```
## # A tibble: 6 x 5
##            name gender height  mass      bmi
##           <chr>  <chr>  <int> <dbl>    <dbl>
## 1    Wat Tambor   male    193    48 12.88625
## 2    Adi Gallia female    184    50 14.76843
## 3     Sly Moore female    178    48 15.14960
## 4  Roos Tarpals   male    224    82 16.34247
## 5 Padmé Amidala female    165    45 16.52893
## 6       Lama Su   male    229    88 16.78076
```

2.3.2 filter

Group observations based on certain criteria - filter

We would like to create a new dataset containing only male gender and BMI at or above 30

```
mysw_m_40 <- filter(mysw, gender == 'male', bmi >= 30)
head(mysw_m_40)
```

```
## # A tibble: 6 x 5
##               name gender height  mass      bmi
##              <chr>  <chr>  <int> <dbl>    <dbl>
## 1            Bossk   male    190   113 31.30194
## 2          Sebulba   male    112    40 31.88776
## 3      Darth Vader   male    202   136 33.33007
## 4 Jek Tono Porkins   male    180   110 33.95062
## 5         Grievous   male    216   159 34.07922
## 6        Owen Lars   male    178   120 37.87401
```

How about, create a new dataset containing height above 200 or BMI above 45, but does not include NA in bmi

```
mysw_ht_45 <- filter(mysw, height >200 | bmi >45, bmi != 'NA')
mysw_ht_45
```

```
## # A tibble: 9 x 5
##                    name        gender height  mass       bmi
##                   <chr>         <chr>  <int> <dbl>     <dbl>
## 1          Roos Tarpals          male    224    82  16.34247
## 2               Lama Su          male    229    88  16.78076
## 3            Tion Medon          male    206    80  18.85192
## 4             Chewbacca          male    228   112  21.54509
```

```
## 5               Tarfful          male    234   136  24.83746
## 6           Darth Vader          male    202   136  33.33007
## 7              Grievous          male    216   159  34.07922
## 8              Dud Bolt          male     94    45  50.92802
## 9 Jabba Desilijic Tiure hermaphrodite    175  1358 443.42857
```

2.4 group_by

2.4.1 Summarize data - summarize

A useful function that sometimes needed is `group_by`

```
mysw_g <- group_by(mysw, gender)
summarise(mysw_g, meanbmi = mean(bmi, na.rm = TRUE),
                  meanht  = mean(height, na.rm = TRUE),
                  meanmass = mean(mass, na.rm = TRUE))
```

```
## # A tibble: 5 x 4
##          gender   meanbmi   meanht   meanmass
##           <chr>     <dbl>    <dbl>      <dbl>
## 1        female  18.82002 165.4706   54.02000
## 2 hermaphrodite 443.42857 175.0000 1358.00000
## 3          male  25.65037 179.2373   81.00455
## 4          none  35.00000 200.0000  140.00000
## 5          <NA>  31.87485 120.0000   46.33333
```

2.5 Summary

'dplyr' package is a very useful package that encourage users to use proper verb when manipulating variables (columns) and observations (rows).

We have learned to use 5 functions but there are more functions available. Other useful functions are:

1. distinct()
2. mutate() and transmute()
3. sample_n() and sample_frac()

Package 'dplyr' is very useful when it is combined with another function that is 'group_by'

3 Using forcats package

3.1 Categorical variables

One of the most important uses of factors is in statistical modeling; since categorical variables enter into statistical models differently than continuous variables, storing data as factors insures that the modeling functions will treat such data correctly.

Factors in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed. The factor function is used to create a factor. The only required argument to factor is a vector of values which will be returned as a vector of factor values.

This package helps to work with factor variables. To start with let us creat a dummy dataset

3.2 forcats

3.2.1 New dataset

```
sex1 <- rbinom(n = 100, size = 1, prob = 0.5)
str(sex1)
```

```
##  int [1:100] 1 1 1 0 0 0 0 1 1 0 ...
```

```
race1 <- rep(seq(1:4), 25)
str(race1)
```

```
##  int [1:100] 1 2 3 4 1 2 3 4 1 2 ...
```

```
data_f <- data.frame(sex1, race1)
head(data_f)
```

```
##   sex1 race1
## 1    1     1
## 2    1     2
## 3    1     3
## 4    0     4
## 5    0     1
## 6    0     2
```

We can see the data now. Now let us see the structure of all variables. You should see that they are all in the integer (numerical) format

```
str(data_f)
```

```
## 'data.frame':    100 obs. of  2 variables:
##  $ sex1 : int  1 1 1 0 0 0 0 1 1 0 ...
##  $ race1: int  1 2 3 4 1 2 3 4 1 2 ...
```

3.2.2 Convert numeric to factor variables

1. sex1 (int) to male (factor)
2. race1 (int) to race2 (factor)

```
data_f$male <- factor(data_f$sex1, labels = c('No', 'Yes'))
data_f$race2 <- factor(data_f$race1, labels = c('Mal', 'Chi', 'Ind', 'Others'))
str(data_f)
```

```
## 'data.frame':    100 obs. of  4 variables:
##  $ sex1 : int  1 1 1 0 0 0 0 1 1 0 ...
##  $ race1: int  1 2 3 4 1 2 3 4 1 2 ...
##  $ male : Factor w/ 2 levels "No","Yes": 2 2 2 1 1 1 1 2 2 1 ...
##  $ race2: Factor w/ 4 levels "Mal","Chi","Ind",..: 1 2 3 4 1 2 3 4 1 2 ...
```

3.2.3 Recode old to new levels

Steps:

1. Create a new variable `malay`
2. From `No` vs `Yes` TO `Fem` and `Male`
3. From `Non-Malay` TO `Chi`, `Non-Malay` TO `Ind` and `Non-Malay` TO `Others`. We keep `Mal` as it is

```
library(dplyr)
library(forcats)
data_f$male <- data_f$male %>% fct_recode('Fem' = 'No', 'Male' = 'Yes')
data_f <- data_f %>% mutate(malay = fct_recode(race2, 'Non-Malay' = 'Chi', 'Non-Malay' = 'Ind', 'Non-Mal
head(data_f)
```

```
##   sex1 race1 male  race2     malay
## 1    1     1 Male    Mal       Mal
## 2    1     2 Male    Chi Non-Malay
## 3    1     3 Male    Ind Non-Malay
## 4    0     4  Fem Others Non-Malay
## 5    0     1  Fem    Mal       Mal
## 6    0     2  Fem    Chi Non-Malay
```

4 Session

```
sessionInfo()
```

```
## R version 3.4.1 (2017-06-30)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 15063)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] forcats_0.2.0        bindrcpp_0.2         dplyr_0.7.4
## [4] RevoUtilsMath_10.0.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.13     assertthat_0.2.0 digest_0.6.12    rprojroot_1.2
##  [5] R6_2.2.2         backports_1.1.1  magrittr_1.5     evaluate_0.10.1
##  [9] rlang_0.1.2      stringi_1.1.5    rmarkdown_1.6    RevoUtils_10.0.5
## [13] tools_3.4.1      stringr_1.2.0    glue_1.1.1       yaml_2.1.14
## [17] compiler_3.4.1   pkgconfig_2.0.1  htmltools_0.3.6  bindr_0.1
## [21] knitr_1.17       tibble_1.3.4
```

5 References

1. https://community.modeanalytics.com/sql/tutorial/data-wrangling-with-sql/
2. https://github.com/tidyverse/dplyr

3. https://blog.rstudio.com/2016/08/31/forcats-0-1-0/
4. https://www.stat.berkeley.edu/classes/s133/factors.html
5.