Data Management II
using dplyr and forcats

Kamarul Imran Musa
Assoc Prof (Epidemiology and Statistics)

2018-06-12

Contents

1 Data transformation (data munging or data wrangling) 2
1.1 Definition of data wrangling . . . 2
1.2 Package: dplyr . . . 2
1.2.1 About dplyr package . . . 2
1.3 Data wrangling (or transformation) using dplyr . . . 2

2 Using dplyr package 2
2.1 Hands-on 1: Preparation and data . . . 2
2.1.1 Working directory and data format . . . 2
2.1.2 Training data . . . 3
2.2 Hands-on 2: `select()` and `mutate()` . . . 4
2.2.1 `select()` . . . 4
2.2.2 `mutate()` . . . 4
2.3 Hands-on 3: `arrange()` and `filter()` . . . 5
2.3.1 arrange . . . 5
2.3.2 filter . . . 6
2.4 Hands-on 4: group_by . . . 7
2.4.1 Summarize data - summarize . . . 7
2.5 Summary . . . 7

3 Using forcats package 7
3.1 Hands-on 5: forcats . . . 7
3.1.1 New dataset . . . 7
3.1.2 Convert numeric to factor variables . . . 8
3.1.3 Recode old to new levels . . . 8

4 Session 9

1 Data transformation (data munging or data wrangling)

1.1 Definition of data wrangling

1. Data munging or data wrangling is loosely the process of manually converting or mapping data from one "raw" form into another format.
2. The process allows for more convenient consumption of the data.
3. In doing so, we will be using semi-automated tools.

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

1.2 Package: dplyr

1.2.1 About dplyr package

dplyr is a package grouped inside **tidyverse** collection of packages.

dplyr package is a very useful package to munge or wrangle or to tranform your data.

It is really a grammar of data manipulation. It provides 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 (or transformation) using dplyr

When we communicate with data, common procedures include

1. reducing the size of dataset by selecting certain variables (or columns)
2. generating new variables from existing variables
3. sorting observation of a variable
4. grouping observations that fulfil certain criteria
5. reducing 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. `select` - to reduce the size of dataset by selecting certain variables
2. `mutate` - to generate new variables from existing variables
3. `arrange` - to sort observation of a variable
4. `filter` - to group observations that fulfil certain criteria
5. `group_by` and `summarize` - to reduce variable to groups to in order to estimate summary statistic

2.1 Hands-on 1: Preparation and data

2.1.1 Working directory and data format

It is very important to ensure you know where your working directory is.

Every time you want to start processing your data, please make sure:

1. you have set your working directory. Type `setwd()` will display the active **working directory**.

2. then, read (import) the data you want to analyze.

Remember, there are a number of ways and packages useful to read the data. They depend on the format of your data. For example SPSS format, Stata format, SAS format, MS Excel format and `.csv` format.

Two useful packages - **haven** and **foreign** packages - are very useful to read or import your data into R memory. Take note that, different packages require you to specify different parameters inside the arguments.

2.1.2 Training data

To make life easier and to facilitate reproducibility, we will use examples available from the public domains.

To reproduce the outputs demonstrated on **tidyverse** website https://github.com/tidyverse/dplyr, we will use similar dataset or datasets with them.

One of the useful datasets is 'starwars'. The `starwars` data comes with **dplyr** package. This data comes from SWAPI, the Star Wars API accessible at http://swapi.co/.

The 'starwars' data is class of `tibble`. The data have:

- 87 rows (observations)
- 13 columns (variables)

Now, let us:

1. load the **dplyr** package
2. examine the column names (variable names)

```
library(dplyr)
```

```
##
## 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"
```

Next, examine the first 10 observations of the data. There are 77 more rows and 7 more variables NOT SHOWN. You can also see the types of the variables `chr`, `int`, `dbl`

```
starwars
```

```
## # A tibble: 87 x 13
##    name     height  mass hair_color skin_color eye_color birth_year gender
##    <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
##  1 Luke Sk~    172    77 blond      fair       blue            19   male
##  2 C-3PO       167    75 <NA>       gold       yellow         112   <NA>
##  3 R2-D2        96    32 <NA>       white, bl~ red             33   <NA>
```

```
##  4 Darth V~    202   136 none       white      yellow          41.9 male
##  5 Leia Or~    150    49 brown      light      brown           19   female
##  6 Owen La~    178   120 brown, gr~ light      blue            52   male
##  7 Beru Wh~    165    75 brown      light      blue            47   female
##  8 R5-D4        97    32 <NA>       white, red red             NA   <NA>
##  9 Biggs D~    183    84 black      light      brown           24   male
## 10 Obi-Wan~    182    77 auburn, w~ fair       blue-gray       57   male
## # ... with 77 more rows, and 5 more variables: homeworld <chr>,
## #   species <chr>, films <list>, vehicles <list>, starships <list>
```

2.2 Hands-on 2: `select()` and `mutate()`

2.2.1 `select()`

When you work with large datasets with many columns, sometimes it is easier to select only necessary columns to create a smaller dataset that you can work on.

To create smaller datasets, select only some of the columns. This will greatly help data exploration or analysis.

In `starwars` data, we have 13 variables. Let us select only variables below, then generate a new dataset named as `mysw` :

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

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

```
## # A tibble: 87 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
## # ... with 77 more rows
```

2.2.2 `mutate()`

With `mutate()`, you can generate new variable or variables.

For example, in the dataset `mysw`, we want to create a new variable `bmi` which equals mass in kg divided by squared height (in meter)

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

```
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  26.0
##  2 C-3PO              <NA>      167    75  26.9
##  3 R2-D2              <NA>       96    32  34.7
##  4 Darth Vader        male      202   136  33.3
##  5 Leia Organa        female    150    49  21.8
##  6 Owen Lars          male      178   120  37.9
##  7 Beru Whitesun lars female    165    75  27.5
##  8 R5-D4              <NA>       97    32  34.0
##  9 Biggs Darklighter  male      183    84  25.1
## 10 Obi-Wan Kenobi     male      182    77  23.2
## # ... with 77 more rows
```

2.3 Hands-on 3: `arrange()` and `filter()`

2.3.1 arrange

To easily sort (ascending or descending) data, the `arrange` function is useful.

The `arrange` function will sort the observation based on the values of the specified variable.

Let create a new dataset (`mysw`) by sorting the observation of variable bmi from the biggest bmi to the lowest bmi:

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

```
## # A tibble: 87 x 5
##    name                  gender        height  mass   bmi
##    <chr>                 <chr>          <int> <dbl> <dbl>
##  1 Jabba Desilijic Tiure hermaphrodite    175  1358 443.
##  2 Dud Bolt              male              94    45  50.9
##  3 Yoda                  male              66    17  39.0
##  4 Owen Lars             male             178   120  37.9
##  5 IG-88                 none             200   140  35
##  6 R2-D2                 <NA>              96    32  34.7
##  7 Grievous              male             216   159  34.1
##  8 R5-D4                 <NA>              97    32  34.0
##  9 Jek Tono Porkins      male             180   110  34.0
## 10 Darth Vader           male             202   136  33.3
## # ... with 77 more rows
```

Now, we will replace the dataset `mysw` with data that contain bmi values from the lowest to the biggest bmi

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

```
## # A tibble: 87 x 5
##    name          gender height  mass   bmi
##    <chr>         <chr>   <int> <dbl> <dbl>
##  1 Wat Tambor    male      193    48  12.9
```

```
##  2 Adi Gallia    female    184    50  14.8
##  3 Sly Moore     female    178    48  15.1
##  4 Roos Tarpals  male      224    82  16.3
##  5 Padmé Amidala female    165    45  16.5
##  6 Lama Su       male      229    88  16.8
##  7 Jar Jar Binks male      196    66  17.2
##  8 Ayla Secura   female    178    55  17.4
##  9 Shaak Ti      female    178    57  18.0
## 10 Barriss Offee female    166    50  18.1
## # ... with 77 more rows
```

2.3.2 filter

To group observations based on certain criteria, we use the `filter` function.

Here, We will create a new dataset `mysw_m_40` that contains only **male** gender and **bmi at or above 30**

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

```
## # A tibble: 8 x 5
##   name             gender height  mass   bmi
##   <chr>            <chr>   <int> <dbl> <dbl>
## 1 Bossk            male      190   113  31.3
## 2 Sebulba          male      112    40  31.9
## 3 Darth Vader      male      202   136  33.3
## 4 Jek Tono Porkins male      180   110  34.0
## 5 Grievous         male      216   159  34.1
## 6 Owen Lars        male      178   120  37.9
## 7 Yoda             male       66    17  39.0
## 8 Dud Bolt         male       94    45  50.9
```

There are missing (NA) bmi observation in the dataset. How about, we create a new dataset containing height above 200 or BMI above 45, BUT does not include NA observation for 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  16.3
## 2 Lama Su               male             229    88  16.8
## 3 Tion Medon            male             206    80  18.9
## 4 Chewbacca             male             228   112  21.5
## 5 Tarfful               male             234   136  24.8
## 6 Darth Vader           male             202   136  33.3
## 7 Grievous              male             216   159  34.1
## 8 Dud Bolt              male              94    45  50.9
## 9 Jabba Desilijic Tiure hermaphrodite    175  1358 443.
```

2.4 Hands-on 4: group_by

2.4.1 Summarize data - summarize

To use `summarize` function, the `group_by` function is almost always necessary.

The `group_by` function will prepare the data for group analysis. For example,

1. to get summary values for mean **bmi**, mean **ht** and mean **mass**
2. for male, female, hermaphrodite and none (**gender** variable)

```
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           18.8   165.     54.0
## 2 hermaphrodite   443.    175    1358
## 3 male             25.7   179.     81.0
## 4 none             35     200     140
## 5 <NA>             31.9   120      46.3
```

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()

Also note that, package 'dplyr' is very useful when it is combined with another function that is 'group_by'

3 Using forcats package

This package helps to work with factor variables.

3.1 Hands-on 5: forcats

3.1.1 New dataset

To start with let us create a dummy dataset:

1. a vector data **sex1** , values = 0,1
2. a vector data **race1** , values = 1,2,3,4
3. a dataframe (dataset) **data_f**

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

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

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

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

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

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

We can see the data now, above.

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

```r
str(data_f)
```

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

3.1.2 Convert numeric to factor variables

1. Generate male variable from sex: sex1 (int) to male (a factor variable). Label as No or Yes
2. Generate race2 variable from race1: race1 (int) to race2 (a factor variable). Label as Mal, Chi, Ind, Others

```r
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  0 0 1 0 1 0 0 1 0 0 ...
##  $ race1: int  1 2 3 4 1 2 3 4 1 2 ...
##  $ male : Factor w/ 2 levels "No","Yes": 1 1 2 1 2 1 1 2 1 1 ...
##  $ race2: Factor w/ 4 levels "Mal","Chi","Ind",..: 1 2 3 4 1 2 3 4 1 2 ...
```

3.1.3 Recode old to new levels

Steps:

1. For variable **male**, change from `No` vs `Yes` TO `Fem` and `Male`
2. Create a new variable **malay** from variable **race2** AND
3. Label `Chi` to `Non-Malay`, `Ind` to `Non-Malay` and `Others` to `Non-Malay`. But we keep `Mal` as it is

Now, we need **dplyr** and **forcats** perform steps 1 to 3 above.

```
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-Malay' = 'Others'))
head(data_f)
```

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

4 Session

```
sessionInfo()
```

```
## R version 3.5.0 (2018-04-23)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 17134)
##
## 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_0.3.0        bindrcpp_0.2.2       dplyr_0.7.5
## [4] RevoUtils_11.0.0     RevoUtilsMath_11.0.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.17     knitr_1.20       bindr_0.1.1      magrittr_1.5
##  [5] tidyselect_0.2.4 R6_2.2.2         rlang_0.2.1      stringr_1.3.1
##  [9] tools_3.5.0      utf8_1.1.4       cli_1.0.0        htmltools_0.3.6
## [13] yaml_2.1.19      rprojroot_1.3-2  digest_0.6.15    assertthat_0.2.0
## [17] tibble_1.4.2     crayon_1.3.4     purrr_0.2.5      glue_1.2.0
## [21] evaluate_0.10.1  rmarkdown_1.9    stringi_1.2.2    compiler_3.5.0
## [25] pillar_1.2.3     backports_1.1.2  pkgconfig_2.0.1
```