Exploratory Data Analysis with R

Data Cleaning - Part II

Xuemao Zhang
East Stroudsburg University

October 10, 2022

Outline

- `group_by()` and `summarize()`
- Reshaping data using `dplyr` or `tidyr` package
- Merging data sets

We need the packages

```
library(tidyverse)
library(gapminder)
```

Recall

- There are 8 fundamental data manipulation verbs in `dplyr`.
  - `mutate()` and `transmutate()`: Add/create new variables.
  - `select()`: Select columns (variables) by their names.
  - `filter()`: Pick rows (observations/samples) based on their values.
  - `distinct()`: Remove duplicate rows.
  - `arrange()`: Reorder the rows.
  - `rename()`: Rename columns.
  - `summarise()`: Compute statistical summaries (e.g., computing the mean or the sum) and thus reduce multiple values down to a single summary. It is similar to `R::base::aggregate`
    - `group_by()` to group variables for summarise
- `slice_sample` (it was `sample_n()` and `sample_frac()`) is used to sample from a data set.

Sampling from a data set

- When we deal with Big Data, we can using sampling to get a rough idea about the data. But be careful!
- Consider the data gapminder in the package gapminder

```
str(gapminder)
```

```
## tibble [1,704 x 6] (S3: tbl_df/tbl/data.frame)
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3
##  $ year     : int [1:1704] 1952 1957 1962 1967 1972 1977 1982 1987 1992
##  $ lifeExp  : num [1:1704] 28.8 30.3 32 34 36.1 ...
##  $ pop      : int [1:1704] 8425333 9240934 10267083 11537966 13079460 14
##  $ gdpPercap: num [1:1704] 779 821 853 836 740 ...
```

```
slice_sample(gapminder, n=3) # sample_n(gapminder,3)
```

```
## # A tibble: 3 x 6
##   country   continent  year lifeExp      pop gdpPercap
##   <fct>     <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Serbia    Europe     2002    73.2 10111559     7236.
## 2 Swaziland Africa     2002    43.9  1130269     4128.
## 3 Sweden    Europe     1987    77.2  8421403    23587.
```

```
slice_sample(gapminder, prop=0.5) #sample_frac(gapminder,0.5)
```

Sampling from a data set

```
slice_sample(gapminder, prop=0.5) #sample_frac(gapminder,0.5)
```

```
## # A tibble: 852 x 6
##    country        continent  year lifeExp      pop gdpPercap
##    <fct>          <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Ghana          Africa     1957    44.8  6391288     1044.
##  2 Egypt          Africa     1997    67.2 66134291     4173.
##  3 Angola         Africa     1992    40.6  8735988     2628.
##  4 Myanmar        Asia       1982    58.1 34680442      424
##  5 Malaysia       Asia       2002    73.0 22662365    10207.
##  6 Iceland        Europe     1992    78.8   259012    25144.
##  7 Angola         Africa     1972    37.9  5894858     5473.
##  8 Israel         Asia       1997    78.3  5531387    20897.
##  9 Oman           Asia       2002    74.2  2713462    19775.
## 10 United Kingdom Europe     1992    76.4 57866349    22705.
## # ... with 842 more rows
```

group_by() and summarize()

- summarize() Summarize allows you to compute summary statistics.
- summarize() becomes extremely useful when combined with group_by().

```
by_continent = group_by(gapminder, continent)
sum_continent = summarize(by_continent, aveLife = mean(lifeExp))
sum_continent;
```

```
## # A tibble: 5 x 2
##   continent aveLife
##   <fct>       <dbl>
## 1 Africa       48.9
## 2 Americas     64.7
## 3 Asia         60.1
## 4 Europe       71.9
## 5 Oceania      74.3
```

group_by() and summarize()

- What is the average life expectancy by continent and what was the sample size for each continent?
- Let's use the pipe %>% operator
- The special summary function n() returns counts.

```
gapminder %>% group_by(continent) %>%
  summarize(aveLife = mean(lifeExp),n=n())
```

```
## # A tibble: 5 x 3
##   continent aveLife     n
##   <fct>       <dbl> <int>
## 1 Africa       48.9   624
## 2 Americas     64.7   300
## 3 Asia         60.1   396
## 4 Europe       71.9   360
## 5 Oceania      74.3    24
```

Common summarization options

- mean: mean within groups
- sum: sum within groups
- sd: standard deviation within groups
- max: max within groups
- n(): number of observations in each group
- first(): first in group
- last(): last in group
- nth(n=3): nth in group (3rd here)

group_by() and summarize()

- group_by() function allows us to create groups based on more than one variable: group_by(data, variables).

```
gapminder %>% group_by(continent, year)%>%
summarize(freq=n())
```

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

```
## # A tibble: 60 x 3
## # Groups:   continent [5]
##    continent  year  freq
##    <fct>     <int> <int>
##  1 Africa     1952    52
##  2 Africa     1957    52
##  3 Africa     1962    52
##  4 Africa     1967    52
##  5 Africa     1972    52
##  6 Africa     1977    52
##  7 Africa     1982    52
##  8 Africa     1987    52
##  9 Africa     1992    52
## 10 Africa     1997    52
```

Reshaping data

- Reshaping data from wide (fat) to long (tall)
- Reshaping data from long (tall) to wide (fat)

See the "tidyr cheat Sheet":

https://raw.githubusercontent.com/rstudio/cheatsheets/main/tidyr.pdf

- This lecture is a modification from SISBID - University of Washington

What is wide/long data?

- Let's consider an example from an experimental design
- **Experiment**: people apply some treatment and then observe its effects on the subjects (subjects in an experiments generally are called experimental units)
  - An experiment requires random assignment of subjects to treatments.
  - If done correctly, experiments provide most compelling evidence that a treatment causes an observed outcome
  - For example, in a randomized clinical study patients in the experimental groups receive the drug while patients in the control groups receive a placebo or sugar pill. The patients do not know if they are receiving the experimental treatment or placebo.

What is wide/long data?

- Some terminologies in Design of Experiments:
  - An experimental unit is the object on which a measurement (or measurements) is taken.
  - The response is the variable being measured by the experimenter.
  - A factor is an independent variable whose values are controlled and varied by the experimenter.
  - A level is the intensity setting of a factor.
  - A treatment is a specific combination of factor levels.

What is wide/long data?

- Example: Is the attention span of children affected by whether or not they had a good breakfast? Twelve children were randomly divided into three groups and assigned to a different meal plan. The response was attention span in minutes during the morning reading time.

No Breakfast	Light Breakfast	Full Breakfast
8	14	10
7	16	12
9	12	16
13	17	15

- The response variable is attention span.
- The experimenter chooses 3 levels of a single factor - breakfast
- Each level of the factor is a treatment
- The experiment is replicated 4 times

What is wide/long data?

- Wide data

No Breakfast	Light Breakfast	Full Breakfast
8	14	10
7	16	12
9	12	16
13	17	15

What is wide/long data?

- Long data

Attention span	treatment
8	No Breakfast
7	No Breakfast
9	No Breakfast
13	No Breakfast
14	Light Breakfast
16	Light Breakfast
12	Light Breakfast
17	Light Breakfast
10	Full Breakfast
12	Full Breakfast
16	Full Breakfast
15	Full Breakfast

What is wide/long data?

More accurately, data is wide or long is **with respect** to certain variables.

Reshaping data using tidyr package

`tidyr` allows you to "tidy" your data:

- `pivot_longer()` - make multiple columns into variables, (wide to long)
- `pivot_wider()` - make a variable into multiple columns, (long to wide)
- `separate` and `extract()` - pull a single string(character) column into multiple columns
- `unite` - combine multiple columns into a single string(character) column

Reshaping data using tidyr package

```r
breakfast=read.csv("../data/breakfast.csv", header = T, sep = ",")
str(breakfast)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ No.Breakfast   : int  8 7 9 13
##  $ Light.Breakfast: int  14 16 12 17
##  $ Full.Breakfast : int  10 12 16 15
```

Reshaping data from wide (fat) to long (tall): tidyr

`tidyr::pivot_longer()` - puts column data into rows.

We want the three column names into "`treatment`" variable in the output data set and the value in "`resp`" variable.

```
library(tidyr)
long = pivot_longer(data=breakfast,
    cols=c(No.Breakfast,Light.Breakfast,Full.Breakfast),
    names_to= "treatment", values_to = "resp")
head(long, 8)
```

```
## # A tibble: 8 x 2
##   treatment        resp
##   <chr>           <int>
## 1 No.Breakfast        8
## 2 Light.Breakfast    14
## 3 Full.Breakfast     10
## 4 No.Breakfast        7
## 5 Light.Breakfast    16
## 6 Full.Breakfast     12
## 7 No.Breakfast        9
## 8 Light.Breakfast    12
```

Reshaping data from wide (fat) to long (tall): tidyr

- We can manually do this easily

```
span=c(breakfast[,1],breakfast[,2],breakfast[,3])
trt=rep(c("No.Breakfast","Light.Breakfast","Full.Breakfast"),
        each=4)
data.frame(resp=span,treatment=trt)
```

```
##    resp       treatment
## 1     8    No.Breakfast
## 2     7    No.Breakfast
## 3     9    No.Breakfast
## 4    13    No.Breakfast
## 5    14 Light.Breakfast
## 6    16 Light.Breakfast
## 7    12 Light.Breakfast
## 8    17 Light.Breakfast
## 9    10  Full.Breakfast
## 10   12  Full.Breakfast
## 11   16  Full.Breakfast
## 12   15  Full.Breakfast
```

Reshaping data from wide (fat) to long (tall): tidyr

```
long %>% count(treatment)
```

```
## # A tibble: 3 x 2
##   treatment            n
##   <chr>            <int>
## 1 Full.Breakfast       4
## 2 Light.Breakfast      4
## 3 No.Breakfast         4
```

Reshaping data from long (tall) to wide (fat): tidyr

- Now we have a long data set.
- Suppose we want to separate the No.Breakfast, Light.Breakfast and Full.Breakfast into different columns
- `tidyr::pivot_wider()` - puts row data into columns.

We want to split the "`treatment`" variable into 3 columns with corresponding values in "`resp`" variable.

Reshaping data from long (tall) to wide (fat): tidyr

```
wide=pivot_wider(data=long,
    names_from = treatment, values_from=resp)
```

```
## Warning: Values from `resp` are not uniquely identified; output w
## * Use `values_fn = list` to suppress this warning.
## * Use `values_fn = {summary_fun}` to summarise duplicates.
## * Use the following dplyr code to identify duplicates.
##   {data} %>%
##     dplyr::group_by(treatment) %>%
##     dplyr::summarise(n = dplyr::n(), .groups = "drop") %>%
##     dplyr::filter(n > 1L)
```

```
wide
```

```
## # A tibble: 1 x 3
##   No.Breakfast Light.Breakfast Full.Breakfast
##   <list>       <list>          <list>
## 1 <int [4]>    <int [4]>       <int [4]>
```

Reshaping data from long (tall) to wide (fat): tidyr

- Add row number to the long data can remove the warning

```
long$row=rep(c(1:4),each=3)
wide=pivot_wider(data=long,
  names_from = treatment, values_from=resp)
wide
```

```
## # A tibble: 4 x 4
##     row No.Breakfast Light.Breakfast Full.Breakfast
##   <int>        <int>           <int>          <int>
## 1     1            8              14             10
## 2     2            7              16             12
## 3     3            9              12             16
## 4     4           13              17             15
```

Reshaping data from long (tall) to wide (fat): tidyr

- The above code is equivalent to the following
  - id_cols is used to uniquely identifies each observation/row. Defaults to all columns in data except for the columns specified in names_from and values_from.
    - ⋆ The order of rows in each group does not matter in this example.

```
long$row=rep(c(1:4),each=3)
wide=pivot_wider(data=long,
  names_from = treatment, values_from=resp,
  id_cols = row)
wide
```

- A lot of missing values will be introduced in the data wide if `long$row=rep(c(1:4),each=3)` is replaced by `long$row=1:nrow(long)` because each subject appears in one and only one treatment.

Reshaping data from long (tall) to wide (fat): tidyr

- Again, you can manually do this easily

```
attach(long)
data.frame(No.Breakfast=long[treatment=='No.Breakfast',2],
  Light.Breakfast=long[treatment=='Light.Breakfast',2],
  Full.Breakfast=long[treatment=='Full.Breakfast',2])
```

```
##   resp resp.1 resp.2
## 1    8     14     10
## 2    7     16     12
## 3    9     12     16
## 4   13     17     15
```

Reshaping data from long (tall) to wide (fat): tidyr

- The variable names are not correct because data type of `long` is tibble

```
str(long)
```

```
## tibble [12 x 3] (S3: tbl_df/tbl/data.frame)
##  $ treatment: chr [1:12] "No.Breakfast" "Light.Breakfast" "Full.Breakfas
##  $ resp     : int [1:12] 8 14 10 7 16 12 9 12 16 13 ...
##  $ row      : int [1:12] 1 1 1 2 2 2 3 3 3 4 ...
```

```
long=as.data.frame(long)
data.frame(No.Breakfast=long[treatment=='No.Breakfast',2],
  Light.Breakfast=long[treatment=='Light.Breakfast',2],
  Full.Breakfast=long[treatment=='Full.Breakfast',2])
```

```
##   No.Breakfast Light.Breakfast Full.Breakfast
## 1            8              14             10
## 2            7              16             12
## 3            9              12             16
## 4           13              17             15
```

```
detach(long)
```

Merge data sets

- base::`merge()` function.
- dplyr has its own version of this in the form of several **functions**: `left_join`, `right_join`, `inner_join`, `full_join`, `anti_join`.
- The difference between these functions is what happens when there is a row in one data frame without a corresponding row in the other data frame.
  - `inner_join` discards such rows.
  - `full_join` always keeps them, filling in missing data with NA.
  - `left_join` always keeps rows from the first data frame
  - `right_join` always keeps rows from the second data frame
  - `anti_join` is a bit different, it gives you rows from the first data frame that aren't in the second data frame.

Merge data sets: `inner_join`

inner_join animation

```
#suppose country and year determines the ID
data1= data.frame(country = c("A","A","B","B","C","C"),
        year=c(2017,2018,2017,2017,2017,2018),
        x1=c(1,2,3,4,5,6))
data2= data.frame(country = c("A","A","B","B","C"),
        year=c(2017,2018,2017,2018,2018),
        x2=c(7,8,9,10,11))
```

Merge data sets: `inner_join`

```
data1;
```

```
##   country year x1
## 1       A 2017  1
## 2       A 2018  2
## 3       B 2017  3
## 4       B 2017  4
## 5       C 2017  5
## 6       C 2018  6
```

```
data2;
```

```
##   country year x2
## 1       A 2017  7
## 2       A 2018  8
## 3       B 2017  9
## 4       B 2018 10
## 5       C 2018 11
```

Merge data sets: `inner_join`

- `inner_join` returns matching rows only.

```
inner_join(data1,data2, by=c("country","year"))
```

```
##   country year x1 x2
## 1       A 2017  1  7
## 2       A 2018  2  8
## 3       B 2017  3  9
## 4       B 2017  4  9
## 5       C 2018  6 11
```

Merge data sets: `full_join`

full_join animation

- `full_join` returns all rows. So NA could be introduced.

```
full_join(data1,data2, by=c("country","year"))
```

```
##   country year x1 x2
## 1       A 2017  1  7
## 2       A 2018  2  8
## 3       B 2017  3  9
## 4       B 2017  4  9
## 5       C 2017  5 NA
## 6       C 2018  6 11
## 7       B 2018 NA 10
```

Merge data sets: `left_join`

left_join animation

- `left_join` always keeps rows from the first data frame

```
lj=left_join(data1,data2, by=c("country","year"))
```

Merge data sets: `right_join`

right_join animation

- `right_join` always keeps rows from the second data frame

```
rj1=right_join(data1,data2, by=c("country","year"))
dim(rj1)
```

```
## [1] 6 4
```

right_join: Switching arguments

```r
rj2=right_join(data2,data1, by=c("country","year"))
dim(rj2)
```

```
## [1] 6 4
```

```r
rj1 = arrange(rj1, x1, x2)%>% select(country, year, x1, x2)
rj2 = arrange(rj2, x1, x2)%>% select(country, year, x1, x2)
lj = arrange(lj, x1, x2)%>% select(country, year, x1, x2)
```

```r
identical(rj1, rj2)
```

```
## [1] FALSE
```

```r
identical(rj2, lj)  ## after some rearranging
```

```
## [1] TRUE
```

Merge data sets: `anti_join`

anit_join animation

- `anti_join` returns rows from the first data frame that aren't in the second data frame.

```
anti_join(data1,data2, by=c("country","year"))
```

```
##   country year x1
## 1       C 2017  5
```

License