Data Manipulation with dplyr in R

Bourbon0212

2019/9/18

Transforming Data

Before start, it's essential to learn about your data with `str()` or `dplyr::glimpse()`

- `str()` : Give the rough look of the structure of the data.
- `glimpse()` : An enhanced version of `str()` of `dplyr` package.

Review some data manipulation verbs from dplyr (link)

- `select()` : **Variables (Col)**, select columns from a dataset, or remove one with `-` .
  - eg. `counties %>% select(state, county, population, unemployment)`
- `mutate()` : **Variables (Col)**, use present data to build new columns of values.
  - eg. `mutate(unemployed_population = population * unemployment / 100)`
- `filter()` : **Observation (Row)**, single out rows from a dataset that meets the condition.
  - eg. `counties %>% filter(state == "New York")`
- `arrange()` : **Observation (Row)**, reorders rows in a dataset.
  - eg. `counties %>% arrange(population)`

```r
library(dplyr)

counties = readRDS("counties.rds")

counties_transform <- counties %>%
  # Select the five columns
```

```
  select(state, county, population, men, women) %>%
  # Add the proportion_men variable
  mutate(proportion_men = men / population) %>%
  # Filter for population of at least 10,000
  filter(population > 10000) %>%
  # Arrange proportion of men in descending order
  arrange(desc(proportion_men))
```

state	county	population	men	women	proportion_men
Virginia	Sussex	11864	8130	3734	0.6852664
California	Lassen	32645	21818	10827	0.6683412
Georgia	Chattahoochee	11914	7940	3974	0.6664428
Louisiana	West Feliciana	15415	10228	5187	0.6635096
Florida	Union	15191	9830	5361	0.6470937
Texas	Jones	19978	12652	7326	0.6332966

Aggregating Data

- `count()` : Count the rows of dataset, count the variable or count the `wt` (weight) of the variable.
- `summarize()` : Calculating values derived from original data, and generate a new dataset.
  - `summarize()` = `summarise()`
  - Common summarize functions: `sum()` , `mean()` , `median()` , `min()` , `max()` , `n()`
- `group_by()` : Use before `summarize()` or `mutate()` to summarize/mutate by group.
  - `group_by()` function itself **DONOT** make any changes to the dataset.
  - `ungroup()` : To ungroup if no need grouping anymore.
- `top_n(n, var)` : Filter out the top `n` for that `variable` .

```r
# Count the rows of dataset, similar to n()
counties %>%
  count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1  3138
```

```r
# Count the variable, eg. total numbers of counties of each state
counties %>%
 count(state, sort = TRUE)
```

```
## # A tibble: 50 x 2
##    state              n
##    <chr>          <int>
##  1 Texas            253
##  2 Georgia          159
##  3 Virginia         133
##  4 Kentucky         120
##  5 Missouri         115
##  6 Kansas           105
##  7 Illinois         102
##  8 North Carolina   100
##  9 Iowa              99
## 10 Tennessee         95
## # ... with 40 more rows
```

```r
# Count the weight of the variable, eg. total population of each state
counties %>%
  count(state, wt = population, sort = TRUE)
```

```
## # A tibble: 50 x 2
##    state                 n
##    <chr>             <dbl>
##  1 California     38421464
##  2 Texas          26538497
##  3 New York       19673174
##  4 Florida        19645772
##  5 Illinois       12873761
##  6 Pennsylvania   12779559
##  7 Ohio           11575977
##  8 Georgia        10006693
##  9 Michigan        9900571
## 10 North Carolina  9845333
## # ... with 40 more rows
```

Here’s a question,

> In how many states do more people live in metro areas than non-metro areas?
> “Metro” (for high-density city areas) or “Nonmetro” (for suburban and country areas).

```
# Select 3 columns
counties_selected <- counties %>%
  select(state, metro, population)

# Step by step to answer the question
counties_selected %>%
  group_by(state, metro) %>%
  summarize(total_pop = sum(population)) %>%
  top_n(1, total_pop)  %>%
  ungroup() %>%
  count(metro)
```

```
## # A tibble: 2 x 2
##   metro        n
```

```
##    <chr>    <int>
## 1 Metro       44
## 2 Nonmetro     6
```

Advanced Skills of Transforming Data

- Using `select()` to select columns with `select_helpers`
    - The colon (`:`) is useful for getting many columns at a time.
    - `starts_with(“X”)` : every name that starts with `X` .
    - `ends_with(“X”)` : every name that ends with `X` .
    - `contains(“X”)` : every name that contains `X` .
    - `matches(“X”)` : every name that matches `X` , where `X` can be a regular expression.
    - `num_range(“x”, 1:5)` : the variables named x01, x02, x03, x04 and x05.
    - `one_of(x)` : every name that appears in `x` , which should be a character vector.
- Comparison of `select()` , `transmute()` , `rename()` , `mutate()`

.	Keep only specified variables	Keep other variables
Can’t change values	`select()`	`rename()`
Can change values	`transmute()`	`mutate()`

```
# Change the name of the unemployment column, new_name = old_name
counties %>%
  rename(unemployment_rate = unemployment)

# Keep the state and county columns, and the columns containing poverty
counties %>%
  select(state, county, contains("poverty"))

# Calculate the fraction_women column without dropping the other columns
counties %>%
  mutate(fraction_women = women / population)

# Keep only the state, county, and employment_rate columns
```

```
counties %>%
  transmute(state, county, employment_rate = employed / population)
```

Case Study: Babynames

> Q1: Finding the year each name is most common.

```
babynames = readRDS("babynames.RDS")

babynames_fraction = babynames %>%
  group_by(year) %>%
  mutate(year_total = sum(number)) %>%
  ungroup() %>%
  mutate(fraction = number / year_total)

babynames_fraction %>%
  group_by(name) %>%
  top_n(1, fraction)
```

```
## # A tibble: 48,040 x 5
## # Groups:   name [48,040]
##     year name      number year_total  fraction
##    <dbl> <chr>      <int>      <int>     <dbl>
##  1  1880 Abbott         5     201478 0.0000248
##  2  1880 Abe           50     201478 0.000248
##  3  1880 Abner         27     201478 0.000134
##  4  1880 Adelbert      28     201478 0.000139
##  5  1880 Adella        26     201478 0.000129
##  6  1880 Adolf          6     201478 0.0000298
##  7  1880 Adolph        93     201478 0.000462
##  8  1880 Agustus        5     201478 0.0000248
##  9  1880 Albert      1493     201478 0.00741
```

```
## 10  1880 Albertina      7     201478 0.0000347
## # ... with 48,030 more rows
```

From the result, we can see that the first few entries are names that were most popular in the 1880's that start with the letter A.

> Q2: Using ratios to describe the frequency of a name.

- Window Function: `lag()`, use to compare consecutive steps.

```
v <- c(1, 3, 6, 14)
lag(v)
```

```
## [1] NA  1  3  6
```

```
v - lag(v)
```

```
## [1] NA  2  3  8
```

```
babynames_ratios = babynames_fraction %>%
  # Arrange the data in order of name, then year
  arrange(name, year) %>%
  # Group the data by name
  group_by(name) %>%
  # Add a ratio column that contains the ratio between each year
  mutate(ratio = fraction/lag(fraction))

babynames_ratios_biggest = babynames_ratios %>%
  # Extract the largest ratio from each name
  top_n(1, ratio) %>%
  # Sort the ratio column in descending order
  arrange(desc(ratio)) %>%
  # Filter for fractions greater than or equal to 0.001
  filter(fraction >= 0.001)
```

year	name	number	year_total	fraction	ratio
1960	Tammy	14365	4152075	0.0034597	70.11500
2005	Nevaeh	4610	3828460	0.0012041	45.82168
1940	Brenda	5460	2301630	0.0023722	37.53315
1885	Grover	774	240822	0.0032140	35.97493
1945	Cheryl	8170	2652029	0.0030807	24.87909
1955	Lori	4980	4012691	0.0012411	23.24564
2010	Khloe	5411	3672066	0.0014736	23.21587
1950	Debra	6189	3502592	0.0017670	22.63804
2010	Bentley	4001	3672066	0.0010896	22.42690
1935	Marlene	4840	2088487	0.0023175	16.82703

We found biggest jumps in a name. Some of these can be interpreted: for example, Grover Cleveland was a president elected in 1884!