L05. Data Transformation

Sim, Min Kyu, Ph.D., mksim@seoultech.ac.kr

SEOULTECH 서울과학기술대학교 데이터사이언스학과

Section 1

I. Packages (Libraries)

Package

정의

- 응용 프로그램, 확장 프로그램, library
- 데이터나 함수의 모음

R의 Package

- R을 처음에 설치하면 `base`라는 package가 인스톨 되어있음
- `base`외의 확장 기능을 제공하는 패키지를 설치하여 사용
- `rmarkdown`, `dplyr`, `tidyr`, `ggplot2`등의 패키지 등을 다룸
- `tidyverse`는 여러 package를 모아둔 패키지이며, 하나같이 훌륭함

Package의 중요성

- R과 Python을 오픈 소스 기반 언어
- 누구나 패키지를 만들고 공유할 수 있음
- 많은 사람들이 많은 패키지를 만들면서 언어의 발전이 일어남

Package의 위상

Environment (환경)	Application (응용프로그램)
윈도우	Excel
안드로이드	카카오톡
R	`dplyr`, `ggplot2`, `rmarkdown`
Python	`pandas`, `numpy`

1. 설치

- `base`외의 패키지를 사용하려면 설치를 먼저 해주어야 함
- 마치 playstore에서 앱을 다운 받아 설치하는 것과 같음
- 따옴표를 넣어서 패키지의 이름을 입력

```
install.packages("package_name")
install.packages("dplyr") # this lecture is about
```

2. 사용 선언

- 코드에서 패키지를 사용하기 전에 패키지 사용을 선언해주어야 함
- 이를 ‘declare’, ‘import’, ‘load’라고 표현
- 따옴표 없이 아래처럼 입력

```
library(package_name)
library(dplyr) # this lecture is about
```

Section 2

II. Dataset 준비 `ISLR::Carseats`

Introduction

ISLR package

- An Introduction to Statistical Learning (ISLR)의 데이터셋과 함수를 모아둔 패키지
- 기계학습에 관한 훌륭한 입문서 (번역본도 있음)
- 웹페이지
  - http://faculty.marshall.usc.edu/gareth-james/ISL/
- MOOC
  - https://www.r-bloggers.com/in-depth-introduction-to-machine-learning-in-15-hours-of-expert-videos/
- 강의노트
  - https://www.alsharif.info/iom530

ISLR::Carsears

- ISLR 패키지의 Carseats라는 데이터셋
- 수백개의 오프라인 매장에서 카시트를 판매하는 업체의 매장별 판매 데이터

```
install.packages("ISLR") # install when using for the first time
library(ISLR)
```

Exploration

1. Load

```r
library(ISLR) # load `ISLR` package
```

2. Data Structure

```r
class(Carseats) # data structure
```

```
## [1] "data.frame"
```

```r
dim(Carseats) # dimensions
```

```
## [1] 400  11
```

```r
str(Carseats) # structural view
```

```
## 'data.frame':    400 obs. of  11 variables:
##  $ Sales      : num  9.5 11.22 10.06 7.4 4.15 ...
##  $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
##  $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
##  $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
##  $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 3 3 ...
##  $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
##  $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
##  $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
```

3. Partial view

```
colnames(Carseats) # column names
```

```
##  [1] "Sales"       "CompPrice"   "Income"      "Advertising" "Population"
##  [6] "Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
## [11] "US"
```

```
head(Carseats) # the first 6 observations
```

```
##   Sales CompPrice Income Advertising Population Price ShelveLoc Age Education
## 1  9.50       138     73          11        276   120       Bad  42        17
## 2 11.22       111     48          16        260    83      Good  65        10
## 3 10.06       113     35          10        269    80    Medium  59        12
## 4  7.40       117    100           4        466    97    Medium  55        14
## 5  4.15       141     64           3        340   128       Bad  38        13
## 6 10.81       124    113          13        501    72       Bad  78        16
##   Urban  US
## 1   Yes Yes
## 2   Yes Yes
## 3   Yes Yes
## 4   Yes Yes
## 5   Yes  No
## 6    No Yes
```

```
tail(Carseats, 2) # the last 2 observations
```

```
##     Sales CompPrice Income Advertising Population Price ShelveLoc Age Education
## 399  5.94       100     79           7        284    95       Bad  50        12
## 400  9.71       134     37           0         27   120      Good  49        16
##     Urban  US
## 399   Yes Yes
## 400   Yes Yes
```

5. Summary stats

```r
summary(Carseats) # summary statistics
```

```
##      Sales          CompPrice       Income        Advertising    
##  Min.   : 0.000   Min.   : 77   Min.   : 21.00   Min.   : 0.000  
##  1st Qu.: 5.390   1st Qu.:115   1st Qu.: 42.75   1st Qu.: 0.000  
##  Median : 7.490   Median :125   Median : 69.00   Median : 5.000  
##  Mean   : 7.496   Mean   :125   Mean   : 68.66   Mean   : 6.635  
##  3rd Qu.: 9.320   3rd Qu.:135   3rd Qu.: 91.00   3rd Qu.:12.000  
##  Max.   :16.270   Max.   :175   Max.   :120.00   Max.   :29.000  
##    Population        Price        ShelveLoc        Age          Education   
##  Min.   : 10.0   Min.   : 24.0   Bad   : 96   Min.   :25.00   Min.   :10.0  
##  1st Qu.:139.0   1st Qu.:100.0   Good  : 85   1st Qu.:39.75   1st Qu.:12.0  
##  Median :272.0   Median :117.0   Medium:219   Median :54.50   Median :14.0  
##  Mean   :264.8   Mean   :115.8                Mean   :53.32   Mean   :13.9  
##  3rd Qu.:398.5   3rd Qu.:131.0                3rd Qu.:66.00   3rd Qu.:16.0  
##  Max.   :509.0   Max.   :191.0                Max.   :80.00   Max.   :18.0  
##  Urban       US     
##  No :118   No :142  
##  Yes:282   Yes:258  
## 
## 
## 
## 
```

6. 각 변수의 data type

```r
sapply(Carseats, class)
```

```
##       Sales   CompPrice      Income Advertising  Population       Price
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##   ShelveLoc         Age   Education       Urban          US
##    "factor"   "numeric"   "numeric"    "factor"    "factor"
```

7. 각 변수에 대해 중복값을 제거한 관찰값 갯수

```r
sapply(Carseats, function(x) length(unique(x)))
```

```
##       Sales   CompPrice      Income Advertising  Population       Price
##         336          73          98          28         275         101
##   ShelveLoc         Age   Education       Urban          US
##           3          56           9           2           2
```

Exploration – Summary

1. Load

```
library(ISLR) # load `ISLR` package
```

2. Data Structure

```
class(Carseats) # data structure
dim(Carseats) # dimensions
str(Carseats) # structural view
```

3. Partial view

```
colnames(Carseats) # column names
head(Carseats) # the first 6 observations
tail(Carseats, 2) # the last 2 observations
```

4. Pop–up windows

```
View(Carseats) # a pop-up windows
```

5. Summary stats

```
summary(Carseats) # summary statistics
```

6. 각 변수의 data type

```
sapply(Carseats, class)
```

7. 각 변수에 대해 중복값을 제거한 관찰값 갯수

```
sapply(Carseats, function(x) length(unique(x)))
```

Section 3

III. `dplyr` package

Background

1. 빠르고 쉽게 `data.frame`을 다루는 함수 제공
   - C로 만들어서 빠름
   - SQL와 유사한 직관적인 문법이라 쉬움

```
library(dplyr)
```

2. Authored by **Hadley Wickham**
   - Head Scientist, Rstudio
   - `R for Data Science` 저자, 통계학 박사
   - 본인이 작성한 패키지들을 포함한 `tidyverse`라는 운영
   - Youtube.com에 key note등 좋은 동영상 많음. 누나도 통계학과 교수

Tidy data set

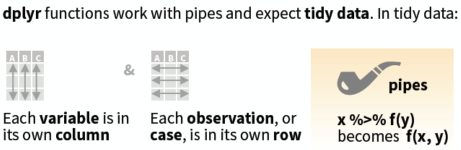

Figure 1: from `dplyr` Cheatsheet

`dplyr`은 tidy `data.frame` 자료 구조를 다루는 함수를 제공한다.

1. Data Structure는 `data.frame`
2. 각각의 row는 관찰값
3. 각각의 column은 변수

The pipe Operator (%>%)

What?

- %>%로 입력
- magrittr 패키지가 origin이며, dplyr에도 포함되어 있는 연산자.
- 앞에서 부터 읽게 해주어 가독성을 높임
- 연속적인 operation을 가능하게 해줌
- f(x)와 x %>% f()가 같음
- f(x,y)와 x %>% f(y)가 같음

Advantage

1. Mathematical expression
   - $y = \sqrt{\sqrt{(x+1)^2 + y} + y}$
2. Way of base (and other languages)
   - `y <- sqrt(sqrt((x+1)^2+y)+y)`
3. Way of pipe
   - `y <- (x+1)^2 %>% add(y) %>% sqrt() %>% add(y) %>% sqrt()`
   - Just the way we would understand!
   - Less number of multiple parenthesis!

Section 4

III. Basic Manipulations of `dplyr` package

Overview

	What it does	Column/Row
1. `rename`	변수 이름 바꿈	Column의 이름을 바꿈
2. `filter`	관찰값 추출	Row를 추출
3. `select`	변수 선택	Column을 선택
4. `arrange`	관찰값 정렬	Row를 정렬
5. `mutate`	변수 생성	Column을 생성
6. `group_by`+`summarise`	Categorical 변수를 이용해 집계	

1. rename (변수 이름 바꿈, Column의 이름을 바꿈)

```
# RENAME `Sales` to `Revenue`
temp <- rename(Carseats, Revenue = Sales) # dplyr
colnames(temp)
```

```
##  [1] "Revenue"     "CompPrice"   "Income"      "Advertising" "Population"
##  [6] "Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
## [11] "US"
```

```
# RENAME `Revenue` back to `Sales`
names(temp)[names(temp)=="Revenue"] <- "Sales" # base
colnames(temp)
```

```
##  [1] "Sales"       "CompPrice"   "Income"      "Advertising" "Population"
##  [6] "Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
## [11] "US"
```

2. filter (관찰값 추출, Row 추출)

```
# FILTER obs only if Income > 100
temp <- filter(Carseats, Income > 90) # dplyr
temp <- Carseats %>% filter(Income > 90) # dplyr + pipe
temp <- Carseats[Carseats$Income > 90,] # base
dim(temp)
```

```
## [1] 101  11
```

```
# FILTER obs only if Age between 30 and 40
temp <- filter(Carseats, Age >= 30 & Age < 40) # dplyr
temp <- Carseats %>% filter(Age >= 30 & Age < 40) # dplyr + pipe
temp <- Carseats[((Carseats$Age >= 30) & (Carseats$Age < 40)),] # base
dim(temp)
```

```
## [1] 63 11
```

3. select (변수 선택, Column을 선택)

```
# SELECT the variables Income and Population
temp <- select(Carseats, Income, Population) # dplyr
temp <- Carseats %>% select(Income, Population) # dplyr + pipe
temp <- Carseats[,c("Income", "Population")] # base
colnames(temp)
```

```
## [1] "Income"     "Population"
```

4. arrange (관찰값 정렬, Row를 정렬)

```
# Ascending (1-2-3)
Carseats <- arrange(Carseats, Price) # dplyr
Carseats <- Carseats %>% arrange(Price) # dplyr + pipe
Carseats <- Carseats[order(Carseats$Price),] # base
head(Carseats %>% select(Income, Price))
```

```
##   Income Price
## 1     69    24
## 2     78    49
## 3    106    53
## 4     81    54
## 5    106    55
## 6     33    63
```

```
# Descending (3-2-1)
Carseats <- arrange(Carseats, desc(Price)) # dplyr
Carseats <- Carseats %>% arrange(desc(Price)) # dplyr + pipe
Carseats <- Carseats %>% arrange(-Price) # dplyr + pipe
Carseats <- Carseats[order(Carseats$Price, decreasing = TRUE),] # base
head(Carseats %>% select(Income, Price))
```

```
##   Income Price
## 1     58   191
## 2     24   185
## 3     42   173
## 4     21   171
## 5     69   166
## 6     65   166
```

5. mutate (변수 생성, Column을 생성)

```
# dplyr
Carseats <- mutate(Carseats,
               AdvPerCapita = Advertising/Population,
               RevPerCapita = Sales/Population)
# dplyr + pipe
Carseats <- Carseats %>%
  mutate(AdvPerCapita = Advertising/Population,
         RevPerCapita = Sales/Population)
# base
Carseats$AdvPerCapita <- Carseats$Advertising/Carseats$Population
Carseats$RevPerCapita <- Carseats$Sales/Carseats$Population
colnames(Carseats)
```

```
##  [1] "Sales"        "CompPrice"    "Income"       "Advertising"  "Population"
##  [6] "Price"        "ShelveLoc"    "Age"          "Education"    "Urban"
## [11] "US"           "AdvPerCapita" "RevPerCapita"
```

mutate + ifelse()

```
# dplyr
Carseats <- mutate(Carseats, AgeClass = ifelse(Age>=60, "Silver", "non-Silver"))
# dplyr + pipe
Carseats <- Carseats %>% mutate(AgeClass = ifelse(Age>=60, "Silver", "non-Silver"))
# base
Carseats$AgeClass <- ifelse(Carseats$Age >= 60, "Silver", "non-Silver")
head(Carseats, 6)
```

```
##   Sales CompPrice Income Advertising Population Price ShelveLoc Age Education
## 1  0.37       147     58           7        100   191       Bad  27        15
## 2  0.00       139     24           0        358   185    Medium  79        15
## 3  6.67       156     42          13        170   173      Good  74        14
## 4  6.39       131     21           8        220   171      Good  29        14
## 5  5.01       159     69           0        438   166    Medium  46        17
## 6  9.53       175     65          29        419   166    Medium  53        12
##   Urban  US AdvPerCapita RevPerCapita   AgeClass
## 1   Yes Yes   0.07000000   0.00370000 non-Silver
## 2    No  No   0.00000000   0.00000000     Silver
## 3   Yes Yes   0.07647059   0.03923529     Silver
## 4   Yes Yes   0.03636364   0.02904545 non-Silver
## 5   Yes  No   0.00000000   0.01143836 non-Silver
## 6   Yes Yes   0.06921241   0.02274463 non-Silver
```

An Exploration using the above

Motivation

- Carseat 회사의 주요 구매층은 아이를 낳아서 키울 나이인 30대 연령층입니다.
- 소득이 높으면서 도시의 평균 연령이 30대인 도시에 충분한 광고비를 지출하고 있나요?

Successive transformation using pipe (%>%)

```
focusCity <- Carseats %>%
  filter(Income > 100) %>% # FILTER high income
  filter(Age >= 30 & Age < 40) %>% # FILTER only cities whose avg_age = 30s
  mutate(AdvPerCapita = Advertising/Population) %>% # MUTATE averaging per capita
  select(Sales, Income, Age, Population, Education, AdvPerCapita) %>% # SELECT main variables
  arrange(Sales) # ARRANGE for better display
focusCity
```

```
##    Sales Income Age Population Education AdvPerCapita
## 1   5.04    114  34        298        16   0.00000000
## 2   5.32    116  39        170        16   0.00000000
## 3   6.80    117  38        337        10   0.01483680
## 4   7.49    119  35        178        13   0.03370787
## 5   7.67    117  36        400        10   0.02000000
## 6   8.55    111  36        480        16   0.04791667
## 7   8.97    107  33        144        13   0.00000000
## 8   9.03    102  35        123        16   0.10569106
## 9   9.39    118  32        445        15   0.03146067
## 10  9.58    104  37        353        17   0.06515581
## 11 10.36    105  34        428        12   0.04205607
## 12 10.59    120  30        262        10   0.05725191
## 13 12.57    108  33        203        14   0.08374384
```

6. group_by + summarise

```
Shelve_effect <- Carseats %>%
  group_by(ShelveLoc) %>%
  summarize(count = n(), # Counting function
            avg_Sales = mean(Sales)) # Average of Sales
Shelve_effect
```

```
## # A tibble: 3 x 3
##   ShelveLoc count avg_Sales
##   <fct>     <int>     <dbl>
## 1 Bad          96      5.52
## 2 Good         85     10.2
## 3 Medium      219      7.31
```

Another example

Motivation

- 도시 인구의 평균 나이가 20대, 30대, 40대 이상인 경우에 Sales에 차이가 있을까요?
  - Step 1. `AgeClass`라는 categorical 변수 생성하고 (`mutate` + `ifelse`)
  - Step 2. `AgeClass`로 묶어서 (`group_by`)
  - Step 3. `mean(Sales)`를 집계하여 (`summarise`)
  - Step 4. 새로운 데이터셋 `ageDiff`를 만듦!

Execution

```
ageDiff <- Carseats %>%
  mutate(AgeClass = # Step 1
           ifelse(Age < 30, "Twenties",
                  ifelse(Age < 40, "Thirties", "FourtyAbove"))) %>%
  group_by(AgeClass) %>% # Step 2
  summarise(count = n(),
            avg_Sales = mean(Sales)) # Step 3
ageDiff
```

```
## # A tibble: 3 x 3
##   AgeClass    count avg_Sales
##   <chr>       <int>     <dbl>
## 1 FourtyAbove   300      7.30
## 2 Thirties       63      8.26
## 3 Twenties       37      7.76
```

Results

- 평균 연령이 30대인 도시에서 가장 매출이 높다.
- Are you sure of this analysis?

Discussion

- 평균 나이에 따른 평균 Revenue의 차이는 무엇을 말해주나요?
- 이 분석이 포함하지 못하고 있는 정보는 어떤 것이 있나요?
- 어떤 데이터가 있으면 더 좋을까요?
- 그 데이터가 있으면 어떻게 하실 건가요?

Takeaway from this small example

- 많은 데이터 분석은 '연속 변수'를 '이산 변수'로 생성하고,
- '이산 변수'의 group별 차이를 파악하는 접근을 사용합니다.
- ex) 연령, 소득, 기온, 성별,···

- 가능한 분석과 원리
  1. **Age**를 factor로 잡아 **Sales**의 Boxplot.
  2. **Age**를 factor로 잡아 scatterplot for **Population** & **Sales**.
  3. **3개 이상의 변수의 관계**를 생각하는 습관
  4. **공간**과 **시간**에 대한 동질성과 이질성을 생각하는 습관

Section 5

IV. Wrap-up

Summary

	What it does	Column/Row
1. `rename`	변수 이름 바꿈	Column의 이름을 바꿈
2. `filter`	관찰값 추출	Row를 추출
3. `select`	변수 선택	Column을 선택
4. `arrange`	관찰값 정렬	Row를 정렬
5. `mutate`	변수 생성	Column을 생성
6. `group_by`+`summarise`	Categorical 변수를 이용해 집계	

`dplyr` vs `base`

	`dplyr`	`base`
문법의 특성	일상적 언어	Classic 프로그래밍 언어
장점	읽고 쓰기 쉬움	타 언어와 style 유사
유사 언어	SQL	Python의 `Pandas`

Data Transformation with dplyr : : CHEAT SHEET

dplyr

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes
x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE**)**
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …**)**
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …**)** Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …**)** Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt**)** Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL**)** Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1**)** Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …**)**
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match) **num_range**(prefix, range) **:**, e.g. mpg:cyl
ends_with(match) **one_of**(…) **-**, e.g, -Species
matches(match) **starts_with**(match)

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …**)**
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …**)**
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …**)** Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …**)** Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL**)** Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, …**)** Rename columns.
rename(iris, Length = Sepal.Length)

R Studio

dplyr

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain all values, all rows.

Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(…, .id = NULL**)**
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, …)
Rows that appear in both x and y.

setdiff(x, y, …)
Rows that appear in x but not y.

union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, …**)**
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, …**)**
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

RStudio