

M21 - dplyr

LearningSpoonsR

2019-02-17

I. Packages & Files

II. ISLR 패키지의 `Carseats` 불러오기

III. dplyr

I. Packages & Files

Package

- ▶ 정의
 - ▶ 응용 프로그램, 확장 프로그램, library
 - ▶ 패키지는 데이터나 함수를 제공하는 프로그램
- ▶ R의 Package
 - ▶ R을 처음에 설치하면 **base**라는 package가 인스톨 되어있음
 - ▶ **base**외의 확장 기능을 제공하는 패키지를 설치하여 사용
 - ▶ **dplyr**, **ggplot2**, **rmarkdown**등의 강력한 패키지 위주로 수업합니다.
- ▶ Package의 위상
 - ▶ R과 Python을 오픈 소스 기반 언어
 - ▶ 누구나 패키지를 만들고 공유할 수 있음
 - ▶ 많은 사람들이 많은 패키지를 만들면서 발전이 일어남
- ▶ Package의 간단한 이해

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

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")
```

사용 선언

- ▶ 코드에서 패키지를 사용하기 전에 패키지 사용을 선언해주어야 함
- ▶ “declare”, “import”, “load”라고 표현
- ▶ 따옴표 없이 아래처럼 입력

```
library(package_name)  
library(dplyr)
```

File types

Source 파일

- ▶ 서식(폰트의 종류와 크기)이 없기에 메모장에서도 편집/저장 가능
- ▶ **.R**
 - ▶ R 문법을 따르는 명령어로 구성
- ▶ **.Rmd**
 - ▶ 3가지 부분으로 구성되어 문서 생성
 - ▶ **Yaml**: 문서의 포맷 결정하는 부분
 - ▶ **Markdown**: 워드프로세서처럼 자유롭게 작성
 - ▶ **R chunks**: R 명령어

Data 파일

- ▶ 일반 데이터 파일
 - ▶ **.csv**
 - ▶ Comma Separated Values
 - ▶ 분리자(separator)가 쉼마(comma)로 정형화된 파일
 - ▶ **.txt**
 - ▶ 다양한 분리자로 구분된 파일
 - ▶ **.xls, .xlsx**
- ▶ R 데이터 파일
 - ▶ R에서 작업중인 데이터를 저장하고 불러오는 파일
 - ▶ **.Rdata**
 - ▶ 특정 객체(변수 혹은 함수)를 저장하거나
 - ▶ 현재 메모리 상태 전체를 저장할 수 있음
 - ▶ 나중에 복원하고 작업을 이어서 하는데에 유용

데이터 파일 입출력

파일을 불러오면 `data.frame` 객체로 저장됨!

1. `.csv`

- ▶ 첫 번째 행이 제목인 경우, 첫 번째 행이 `dataset`의 column name이 됨
- ▶ `header=TRUE`가 default input (입력하지 않아도 자동으로 `header=TRUE`가 됨)
- ▶ `stringsAsFactors = FALSE`를 넣지 않으면 `data.frame`의 모든 string변수가 factor형으로 저장됨

```
dataset <- read.csv("file_name.csv", header = TRUE, stringsAsFactors = FALSE)
dataset <- read.csv("file_name.csv", stringsAsFactors = FALSE)
```

- ▶ 첫 번째 행이 제목이 아니라면 `header=FALSE`를 넣어줘야겠죠?

```
dataset <- read.csv("file_name.csv", header = FALSE, stringsAsFactors = FALSE)
```

- ▶ `dataset`을 `file_name.csv`로 저장하려면?

```
write.csv(dataset, "file_name.csv")
```

2. .txt

- ▶ csv파일과 매우 유사
- ▶ csv파일은 구분자가 comma이기에 별도의 입력이 필요하지 않았지만...
- ▶ txt파일에서는 구분자(seperator, **sep**)를 입력해주어야 함.
- ▶ Ex) **sep**="\t" (Tab), **sep**="." (Period), **sep**="\n" (Linebreak)

```
dataset <- read.table("file_name.txt", header = FALSE, sep = "\t", stringsAsFactors = FALSE)
write.table(dataset, "fileName.txt") # save
```

3. .xls, .xlsx

- ▶ 불러오기는 가능
- ▶ 저장은 불가능
- ▶ **col_names=FALSE**는 **header=FALSE**에 대응
- ▶ **stringsAsFactors = FALSE**를 하지 않아도 string으로 저장

```
library(readxl)
dataset <- read_excel("file_name.xlsx")
dataset <- read_excel("file_name.xlsx", col_names = FALSE)
```


4. R 데이터 파일

- ▶ 특정 객체(변수 혹은 함수)들을 저장하고 나중에 사용

```
save(dataset, "file_name.Rdata") # save one data set  
save(dataset1, dataset2, "file_name.Rdata") # save two data sets  
load("file_name.Rdata") # load
```

- ▶ 현재 메모리의 모든 객체(변수와 함수)를 저장하고 나중에 복원 가능

```
save.image("file_name.Rdata") # save entire workspace  
load("file_name.Rdata") # load
```

경로 (path & directory)

Working directory

- ▶ R엔진이 인식하고 있는 현재 폴더
 - ▶ `getwd()`를 실행하면 확인 가능
 - ▶ `setwd("directory_you_want")`를 사용해서 변경가능
- ▶ R Studio 실행 방식에 따라서 working directory가 달라짐
 - ▶ Rstudio 아이콘을 더블클릭해서 실행 → 디폴트 설정값
 - ▶ `.R`이나 `.Rmd`파일을 더블클릭해서 실행 → 해당 소스파일이 위치한 폴더

경로 (path)

▶ 절대경로

- ▶ WD와 상관없이 전체 경로로 입력
- ▶ Ex1) `read.csv("C:/LS-DS/M21/data/file_name.csv")`
- ▶ 컴퓨터나 폴더가 변경되면 다시 지정해줘야 함

▶ 상대경로

- ▶ WD 기준으로 입력
- ▶ Ex2) `read.csv("file_name.csv")`가 제대로 작동하려면 WD에 `file_name.csv`가 존재해야 함.
- ▶ Ex3) WD가 "C:/LS-DS/M21"인 경우에 `read.csv("/data/file_name.csv")`을 입력하면 Ex1)과 같음
- ▶ `..`을 이용하면 상위 폴더 (parent directory)로 이동
- ▶ Ex4) WD가 "C:/LS-DS/M22"인 경우에 `read.csv("../M21/data/file_name.csv")`을 입력하면 Ex1)과 같음

▶ Tip

- ▶ Project 단위로 폴더를 만들어서 관리
- ▶ 소스파일은 해당 폴더에 보관
- ▶ 같은 폴더, 혹은 데이터 파일이 여러개인 경우에 `/data/` 폴더에 데이터파일 보관하며 상대경로를 이용해서 불러오기

II. ISLR 패키지의 Carseats 불러오기

▶ 수백개의 오프라인 매장에서 카시트를 판매하는 업체의 매장별 판매 데이터

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

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

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

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

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

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

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

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

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

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

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

```
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 ...
```

▶ 기초 통계량

```
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
##  Min.   : 10.0   Min.   : 24.0   Bad   : 96   Min.   :25.00
##  1st Qu.:139.0   1st Qu.:100.0   Good  : 85   1st Qu.:39.75
##  Median :272.0   Median :117.0   Medium:219   Median :54.50
##  Mean   :264.8   Mean   :115.8           Mean   :53.32
##  3rd Qu.:398.5   3rd Qu.:131.0           3rd Qu.:66.00
##  Max.   :509.0   Max.   :191.0           Max.   :80.00
##  Education      Urban      US
##  Min.   :10.0   No :118   No :142
##  1st Qu.:12.0   Yes:282   Yes:258
##  Median :14.0
##  Mean   :13.9
##  3rd Qu.:16.0
##  Max.   :18.0
```

▶ 각 변수의 data type

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

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

▶ 각 변수에 대해 중복값을 제거한 관찰값 갯수

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

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


III. dplyr

Background

1. 직관적인 문법으로 빠르게 **data.frame**을 다루는 패키지
 - ▶ 가장 빠른 언어인 C를 기반으로 만듦
 - ▶ 가장 직관적인 SQL (Standard Query Language)과 유사하게 만들어져있음
 - ▶ 코드 가독성이 높음
 - ▶ 그러나 base 명령어도 같이 알아두면 장점이 있음
 - ▶ 타인의 코드 참조
 - ▶ 파이썬, SQL등 다른 언어를 배울 때 도움이 됨
2. 제작자: Hadley Wickham, Ph.D.
 - ▶ Head Scientist, Rstudio
 - ▶ **ggplot2**, **dplyr**, **reshape2**, **stringr** 등의 본인이 작성한 패키지를 묶어서 **tidyverse**라는 패키지로 묶음
 - ▶ 통계학 박사 후 R에서 다수의 사용하기 좋은 패키지 개발
 - ▶ **R for Data Science** 저자
 - ▶ youtube.com에 keynote등 좋은 동영상 많아요!
 - ▶ 누나도 통계학 전공 교수

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



Figure 1: from **dplyr** Cheatsheet

▶ **dplyr**은 tidy `data.frame` 자료 구조를 가짐

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

▶ **pipe**는 앞에서 부터 읽게 해주어 가독성을 높임!

- ▶ `f(x,y)`와 `x %>% f(y)`가 같음
- ▶ Ex) 아래가 모두 같음
 - ▶ $y = (x_1 + x_2)^2 + x_3$
 - ▶ `y = add(square(x_1+x_2), x_3)`
 - ▶ `y = (x_1+x_2) %>% square() %>% add(x_3)`
 - ▶ ...

Basic Manipulations

1. **rename**

- ▶ 변수 이름 바꾸기
- ▶ column의 이름을 바꿈

2. **filter**

- ▶ 관찰값 추출
- ▶ row를 선택함

3. **select**

- ▶ 변수 추출
- ▶ column을 선택함

4. **arrange**

- ▶ 관찰값 정렬
- ▶ row를 재정렬함

5. **mutate**

- ▶ 변수 생성
- ▶ column을 만듦

6. **group_by + summarise**

- ▶ Categorical 변수를 이용해 집계

1. rename (이름 바꾸기)

```
library(dplyr)
# dplyr: rename `Sales` to `Revenue`
temp <- rename(Carseats, Revenue = Sales)
# base: rename `Revenue` to `Sales`
names(temp)[names(temp)=="Revenue"] <- "Sales"
```

2. filter (관찰값 추출, Row 추출)

```
# takes obs only if Income > 100
temp <- filter(Carseats, Income > 100) # dplyr
temp <- Carseats %>% filter(Income > 100) # dplyr, pipe
temp <- Carseats[Carseats$Income > 100,] # base
# takes obs only if Age between 30 and 40
temp <- filter(Carseats, Age >= 30 & Age < 40) # dplyr
temp <- Carseats %>% filter(Age >= 30 & Age < 40) # dplyr + pipe
temp <- Carseats[((Carseats$Age >= 30) & (Carseats$Age < 40)),] # base
```

3. select (변수 추출, Column 선택)

```
# Choose the variable Income and Population  
temp <- select(Carseats, Income, Population) # dplyr  
temp <- Carseats %>% select(Income, Population) # dplyr  
temp <- Carseats[,c("Income", "Population")] # base
```

4. arrange (정렬)

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

5. mutate (새로운 변수 만들기)

```
# 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
```

mutate with 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")
```

데이터셋에 대해 궁금한 점

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

```
# Successive treatments by pipe
focusCity <- Carseats %>%
  filter(Income > 100) %>% # high income
  filter(Age >= 30 & Age < 40) %>% # Take only cities with avg age = 30s
  mutate(AdvPerCapita = Advertising/Population) %>% # average per capita
  select(Sales, Income, Age, Population, Education, AdvPerCapita) %>% # select variables
  arrange(Sales) # ascending (1-2-3)
```

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

- ▶ 도시 인구의 평균 나이가 20대, 30대, 40대 이상인 경우에 Sales에 차이가 있을까요?
 - ▶ Step 1. AgeClass라는 categorical 변수 생성 (by mutate + ifelse)
 - ▶ Step 2. AgeClass로 묶어서 (group_by)
 - ▶ Step 3. mean(Sales)를 집계 (summarise)
 - ▶ Step 4. 새로운 데이터셋 ageDiff이 만들어 짐!

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

```
## # A tibble: 3 x 2  
##   AgeClass    avgRevenue  
##   <chr>      <dbl>  
## 1 FourtyAbove    7.30  
## 2 Thirties      8.26  
## 3 Twenties      7.76
```

- ▶ 이번 분석의 문제점은 무엇인가요?

Discussion

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

Discussion

- ▶ 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. Age를 factor로 잡아 Sales의 Boxplot.
 2. Age를 factor로 잡아 scatterplot for Population & Sales.
 3. 3개 이상의 변수의 관계를 생각하는 습관
 4. 공간과 시간에 대한 동질성과 이질성을 생각하는 습관
- ▶ #generatlizationIsLearning
- ▶ #universalTruth #specificTime&Place

Discussion

dplyr vs base

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

SQL (Standard Query Language)

1. 대용량 데이터 베이스와 통신하는 언어
2. R에서도 **sqldf**라는 패키지를 사용해서 SQL 명령어로 작업할 수 있음 (M51)
3. R을 할 줄 아는 사람이 SQL을 배우는데 걸리는 시간 < 1 day
4. 데이터를 보는 눈을 키워 줍니다.

Data Transformation with dplyr : CHEAT SHEET



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



Each **variable** is in its own **column**



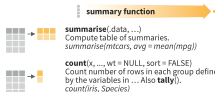
Each **observation**, or **case**, is in its own **row**



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).

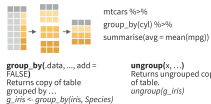


VARIATIONS

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

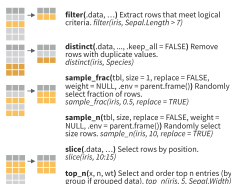
Use **group_by**() to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.



Manipulate Cases

EXTRACT CASES

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

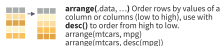


Logical and boolean operators to use with filter()

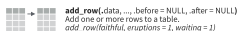
< <= is.na() %in% | xor()
 > >= !is.na() ! &

See ?base:logic and ?Comparison for help.

ARRANGE CASES



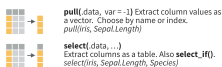
ADD CASES



Manipulate Variables

EXTRACT VARIABLES

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

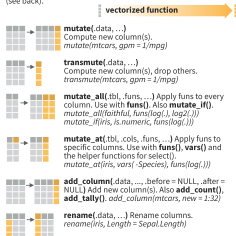


Use these helpers with **select** (), e.g. `select(iris, starts_with("Sepal"))`

ends_with(match) **num_range**(prefix, range) e.g. `mpg:cyl`
matches(match) **one_of**(...) e.g. `Species`
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).



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()
dplyr::cummax() - Cumulative max()
dplyr::cummean() - Cumulative mean()
dplyr::cummin() - Cumulative min()
dplyr::cumprod() - Cumulative prod()
dplyr::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() - $x \geq \text{left} \& x \leq \text{right}$
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
dplyr::pmax() - element-wise max()
dplyr::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.
`o <- rownames_to_column(iris, var = "C")`

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

Also has **rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

x + **y** = **combined**

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("col" = "col2")**, to match on the column(s) to match on.
`left_join(x, y, by = c("D" = "D"))`

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

COMBINE CASES

x + **y** = **combined**

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

x + **y** = **combined**

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.



blank

blank