분석프로그래밍 I.02

Operations, Variables, Data types, Vectors

국민대학교 경영학부 빅데이터경영통계학과

2018. 3. 12(월)

1 magrittr 패키지의 파이프

하나의 대상에 여러 함수를 연속적으로 적용해야 할 경우에는 괄호에 여러 번 중첩되어 가독성이 떨어진다. 예를 들어 diamond에 head와 dim 함수를 적용한다면, 다음과 같다.

```
> data(diamonds, package='ggplot2')
> dim(head(diamonds, n=4))
```

magrittr 패키지의 '%>%'과 '.'를 활용하면 복잡하게 중첩된 괄호를 쓸 필요가 없다.

```
> library(magrittr)
> diamonds %>% head(., n=4) %>% dim(.)
```

'.'은 '%>%' 이전의 결과를 나타낸다. 따라서 첫 번째 '.'은 diamonds를 두 번째 '.'는 head(diamonds, n=4)의 결과를 나타낸다. 만약 '.'이 함수의 첫 번째 인자로 쓰일 경우에는 다음과 같이 생략할 수 있다.

```
> diamonds %>% head(n=4) %>% dim()
```

그리고 dim()처럼 인자가 '.'뿐인 경우에는 괄호까지 생략할 수 있다.

```
> diamonds %>% head(n=4) %>% dim
```

다른 예를 들어보자. (이런 것도 가능하다!)

```
> diamonds %>% .$price
> diamonds %>% .[["price"]]
```

위에서 다룬 diamonds 데이터는 데이터 프레임으로 보이지만 사실 tibble(티블)이라는 데이터 형식이다. 이를 확인하기 위해 class 함수를 사용할 수도 있고, tibble이나 dplyr 패키지를 불어들인 후 diamonds를 다시 확인해 볼 수도 있다.

```
> class(diamonds)
[1] "tbl_df"     "tbl"        "data.frame"
> library(dplyr)
...

> diamonds
# A tibble: 53,940 x 10
   carat cut       color clarity depth table price     x     y     z
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
 5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
 6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
 7 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
 8 0.26  Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
 9 0.22  Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
10 0.23  Very Good H     VS1      59.4    61   338  4     4.05  2.39
# ... with 53,930 more rows
```

class 함수의 결과는 diamonds가 데이터프레임(data.frame)과 호환 되는 티블(tibble, tbl_df) 형식임을 알려준다. 티블은 데이터를 좀 더 깔끔하게 표시한다. 'carat, cut, ...'은 열이름, '<dbl>, <ord>, ...'은 각 열의 데이터 형식을 나타내며, 화면의 가로, 세로 너비를 넘어가는 행과 열은 생략된다.[1]

[1]만약 생략되는 부분을 확인하고 싶다면 print(diamonds, n= , width=)를 활용하자. 여기서

왼쪽 가장자리의 '1, 2, ...'는 행의 순서를 보여준다. 마지막으로 '... with 53,930 more rows'는 화면에 표시되지 않은 열의 수를 나타낸다.

티블은 데이터 프레임과 달리 열 이름의 일부를 사용하여 열을 참조할 수 없다 (다른 말로 partial matching이 불가능하다). 다음의 코드를 보자. diaTB, diaDF는 동일한 자료를 티블과 데이터 프레임 형식으로 저장하고 있다. 'price' 열을 부분 이름 'pri'을 써서 참조하려고 하고 있다. diaDF$pri는 가능하지만 diaTB$pri는 불가능하다.

```
> diaTB <- as_tibble(diamonds[1:10, ])
> diaDF <- as.data.frame(diamonds[1:10, ])
>
> diaDF$pri
 [1] 326 326 327 334 335 336 336 337 337 338
> diaDF[, 'pri']
Error in `[.data.frame`(diaDF, , "pri") : undefined columns selected
> diaTB$pri
NULL
Warning message:
Unknown or uninitialised column: 'pri'.
> diaTB[, 'pri']
Error: Column `pri` not found
```

2 dplyr 패키지

dplyr 패키지의 slice, filter, select, mutate, arrange, summarize, group_by, do 등의 함수는 데이터 가공을 도와준다. 특히 이름에서 쉽게 연상되는 기능으로 초보자도 쉽게 코드를 읽을 수 있다. 특히 '%>%'과 함께 사용하면 코드를 직관적으로 이해는데 도움이 된다. 여기서는 mtcars 데이터를 활용하여 dplyr 패키지와 '%>%'를 활용하여 데이터를 가공하는 법을 살펴본다.

```
> data(mtcars)
> tb = as_tibble(mtcars)
```

'n='은 행의 수, 'width= '는 화면의 너비를 나타낸다.

dplyr 패키지의 함수는 입력을 티블로 변환하여 처리한다. 여기서는 as_tibble 함수를 사용하여 미리 티블 형식으로 바꾸었다.

2.1 행의 순서로 데이터의 부분 참조

데이터 테이블 tb의 두 번째에서 열다섯 번째 행을 참조하려면 tb[2:15,]로 쓰면 된다. slice 함수를 쓴다면 slice(tb, 2:15)이 된다.

```
tb[2:15, ]
slice(tb, 2:15)
```

이를 '%>%'와 함께 쓴다면 다음과 같다.

```
tb %>% .[2:15, ]
tb %>% slice(., 2:15)
```

위의 slice(., 2:15)의 경우 '.'이 첫 번째 인자이므로 생략할 수 있다.

```
tb %>% slice(2:15)
tb %>% slice(c(2:10, 11, 12, 13, 14, 15))
```

2.2 논리 벡터를 사용하여 행 부분 참조

mtcar(또는 티블 형식 tb)에서 mpg가 20 초과인 행만을 뽑아 보고 싶다. 데이터 프레임에서 자주 사용하는 방법은 tb[tb$mpg>20,]이다. filter 함수를 사용하면 filter(tb, mpg>20) 또는 tb %>% filter(., mpg>20)이 된다. '.'를 생략한다면 tb %>% filter(mpg>20)이 된다.

```
tb[tb$mpg>20, ]
filter(tb, mpg>20)

tb %>% filter(., mpg>20)

tb %>% filter(mpg>20)
```

2.3 열 이름이나 번호로 부분 참조

티블 데이터 tb에서 첫 번째 와 세 번째 열을 보고 싶다면 데이터 프레임처럼 tb[, c(1,3)]을 사용할 수 있다. dplyr의 select 함수를 사용하면 select(tb, c(1,3))이 된다. %>%를 사용하면, tb %>% select(c(1,3))이 된다.

```
tb[, c(1,3)]
select(tb, c(1,3))
tb %>% select(c(1,3))
```

열 이름을 사용하고 싶다면 다음과 같다. select 함수를 사용할 때에는 열이름에 따옴표(" 또는 ')를 생략할 수 있다. 그리고 열이름을 하나의 벡터로 만들 필요도 없다.

```
tb[, c("cyl", "hp")]

select(tb, c("cyl", "hp"))
select(tb, c(cyl, hp))

tb %>% select(c("cyl", "hp"))
tb %>% select(c(cyl, hp))

tb %>% select("cyl", "hp")
tb %>% select(cyl, hp)
```

select의 좋은 점의 하나는 열이름에 ':'을 쓸 수 있다는 점이다. 예를 들어 데이터 프레임 tb에서 열이름 hp에서 열이름 qsec까지를 선택하고 싶다고 해보자. 열의 순번을 안다면 tb %>% select(4:7)을 할 수 있다(hp는 tb의 4번째 열이고, qsec는 tb의 7번째 열이다). 하지만 열의 순번을 모른다면? 열의 수가 굉장히 많은 데이터에서 열의 순번을 파악하는 것이 생각만큼 쉽지 않다.

```
> which(colnames(tb)=='hp')
[1] 4
> which(colnames(tb)=='qsec')
[1] 7
> tb[, which(colnames(tb)=='hp'):which(colnames(tb)=='qsec')]
# A tibble: 32 x 4
      hp  drat    wt  qsec
   <dbl> <dbl> <dbl> <dbl>
```

```
 1   110  3.9   2.62  16.5
 2   110  3.9   2.88  17.0
 3    93  3.85  2.32  18.6
 ...
```

select를 사용한다면 간단하게 select(hp:qsec)으로 쓸 수 있다. (하지만 select('hp':'qsec')은 쓸 수 없음을 주의하자.)

```
> tb %>% select(hp:qsec)
# A tibble: 32 x 4
      hp  drat    wt  qsec
   <dbl> <dbl> <dbl> <dbl>
 1   110  3.9   2.62  16.5
 2   110  3.9   2.88  17.0
 3    93  3.85  2.32  18.6
 ...
```

마지막으로 select는 참조하고자 하는 열이름을 하나의 벡터로 만들지 않아도 되지만, slice의 경우는 그렇지 않다는 점에 유의하자.

```
> slice(tb, c(5,7))
# A tibble: 2 x 11
    mpg   cyl  disp    hp  drat    wt  qsec    vs    am
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  18.7     8   360   175  3.15  3.44  17.0     0     0
2  14.3     8   360   245  3.21  3.57  15.8     0     0
# ... with 2 more variables: gear <dbl>, carb <dbl>
> slice(tb, 5, 7)
Error in slice_impl(.data, dots) : slice only accepts one expression
```

2.4 특정한 조건을 만족하는 열 이름 참조

select 함수 안에 다음의 함수를 써서 열이름이 특정한 조건을 만족하는 열만 선별할 수 있다.

먼저 이해하기 쉬운 starts_with( ), ends_with( ), contains_with( )를 보자. 다음의 예로 충분히 이해할 수 있을 것이다.

`starts_with('ab')`	ab로 시작하는
`ends_with('yz')`	yz로 끝나는
`contains_with('ef')`	ef를 포함하는
`one_of(coln)`	문자 벡터 coln의 각 원소와 일치하는
`matches('..[cd]')`	정규표현식 '..[cd]'에 대응하는

```
> tb3 <- tb %>% slice(1:3)
> tb3
# A tibble: 3 x 11
    mpg   cyl  disp    hp  drat    wt  qsec    vs    am
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  21       6   160   110  3.9   2.62  16.5     0     1
2  21       6   160   110  3.9   2.88  17.0     0     1
3  22.8     4   108    93  3.85  2.32  18.6     1     1
# ... with 2 more variables: gear <dbl>, carb <dbl>
> tb3 %>% select(starts_with('c'))
# A tibble: 3 x 2
    cyl  carb
  <dbl> <dbl>
1     6     4
2     6     4
3     4     1
> tb3 %>% select(starts_with('ca'))
# A tibble: 3 x 1
   carb
  <dbl>
1     4
2     4
3     1
> tb3 %>% select(ends_with('p'))
# A tibble: 3 x 2
   disp    hp
  <dbl> <dbl>
1   160   110
2   160   110
3   108    93
> tb3 %>% select(contains('c'))
# A tibble: 3 x 3
    cyl  qsec  carb
  <dbl> <dbl> <dbl>
1     6  16.5     4
2     6  17.0     4
3     4  18.6     1
```

`select` 함수를 쓰면 열이름을 따옴표 없이 쓸 수 있다는 장점이 있다. 하지만 열이름을 저장하는 문자 벡터를 사용하려면 어떻게 해야 하는가? 보통은 `'mpg'`

로 쓰면 열이름이 mpg라는 의미이고, mpg는 mpg라는 변수를 의미한다. 하지만 select 함수 안에서는 mpg는 열이름 mpg를 나타낸다. 만약 mpg 벡터를 의미하고 싶다면 one_of() 함수를 사용한다. (여기서는 열이름을 나타내는 벡터로 coln을 사용하였다. **coln**ame을 의미하는 이름이다.)

```
> coln <- c('drat', 'qsec')
> tb3 %>% select(one_of(coln))
# A tibble: 3 x 2
   drat  qsec
  <dbl> <dbl>
1  3.9   16.5
2  3.9   17.0
3  3.85  18.6
```

matches() 함수는 정규표현식을 사용하여 열이름을 선택하기 위해 사용한다. 예를 들어 정규표현식 '^(.s|.{4})'는 두번째 문자가 s이거나 네문자로 이루어진 경우를 나타낸다. 이를 사용해서 열을 선택하면 다음과 같다.

```
> tb3 %>% select(matches('^(.s|.{4})'))
# A tibble: 3 x 6
   disp  drat  qsec    vs  gear  carb
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   160  3.9   16.5     0     4     4
2   160  3.9   17.0     0     4     4
3   108  3.85  18.6     1     4     1
```

이런 함수(starts_with, ends_with 등)의 도움 없이 동일한 열을 선택하고자 한다면 보통 정규표현식을 사용하게 된다. 동일한 역할을 앞에서 소개한 함수를 사용한 경우와 정규표현식을 사용한 경우를 비교해보면 다음과 같다.

tb %>% select(starts_with('c'))	tb[, grep('^c', colnames(tb))]
tb %>% select(ends_with('p'))	tb[, grep('p$', colnames(tb))]
tb %>% select(contains('c'))	tb[, grep('c', colnames(tb))]

2.5 특정한 열 이름 제외

만약 cyl, qsec을 제외한 나머지 열을 선택하고 싶다면 다음의 두 방법을 사용할 수 있다.

```
tb %>% select(-cyl, -qsec)
tb %>% select(-c(cyl, qsec))
```

특정한 조건을 만족하는 열 이름을 제외하고 싶다면 위에서 소개한 함수 starts_with, ends_with 등의 앞에 '-'를 붙인다.

```
tb %>% select(-starts_with('c'))
tb %>% select(-ends_with('p'))
tb %>% select(-contains('c'))
```

2.6 새로운 열 추가

새로운 열을 추가하고자 한다면 dplyr 패키지의 mutate 함수를 사용할 수 있다. 열이름을 정할 수도 있고, 생략할 수도 있다. 여러 열을 추가할 수도 있다. 다음의 예를 보자.

```
tb2 <- tb %>% select(hp, cyl, qsec) %>% slice(1:3)

tb2 %>% mutate(hp/cyl)
tb2 %>% mutate(hpPerCyl = hp/cyl)
tb2 %>% mutate(hpPerCyl = hp/cyl, V2 = hp*qsec)
```

그 결과는 다음과 같다.

```
> tb2 <- tb %>% select(hp, cyl, qsec) %>% slice(1:3)
> tb2 %>% mutate(hp/cyl)
# A tibble: 3 x 4
     hp   cyl  qsec `hp/cyl`
  <dbl> <dbl> <dbl>    <dbl>
1   110     6  16.5     18.3
2   110     6  17.0     18.3
3    93     4  18.6     23.2
> tb2 %>% mutate(hpPerCyl = hp/cyl)
# A tibble: 3 x 4
     hp   cyl  qsec hpPerCyl
  <dbl> <dbl> <dbl>    <dbl>
```

```
1   110     6  16.5     18.3                                                  13
2   110     6  17.0     18.3                                                  14
3    93     4  18.6     23.2                                                  15
> tb2 %>% mutate(hpPerCyl = hp/cyl, V2 = hp*qsec)                             16
# A tibble: 3 x 5                                                             17
     hp   cyl  qsec hpPerCyl    V2                                            18
  <dbl> <dbl> <dbl>    <dbl> <dbl>                                           19
1   110     6  16.5     18.3 1811.                                           20
2   110     6  17.0     18.3 1872.                                           21
3    93     4  18.6     23.2 1731.                                           22
```

tb2 %>% mutate(hp/cyl)를 결과를 보면 열이름이 `hp/cyl`로 되어 있다. 'hp/cyl'과 같은 열이름은 'hp 나누기 cyl'과 구분할 수 없기 때문에 잘 쓰이지 않는다. 하지만 'hp/cyl'을 열이름으로 정해줄 수 있으며, 이를 사용하려면 다음과 같이 `를 사용하여 열이름을 감싸준다. (다음 예에서 mutate의 결과는 따로 변수에 담아야 함을 주의하자.)

```
> tb2$`hp/cyl`                                                              1
NULL                                                                        2
Warning message:                                                            3
Unknown or uninitialised column: 'hp/cyl'.                                  4
> tb3 <- tb2 %>% mutate(hp/cyl)                                             5
> tb3$`hp/cyl`                                                              6
[1] 18.33333 18.33333 23.25000                                              7
```

다음과 같은 기존의 방법과 비교해보자.

```
1 tb$V2 = with(tb, hp*qsec)
2 tb[c('V1', 'V2')] = data.frame(tb$hp/tb$cyl, tb$hp*tb$qsec)
```

2.7 정렬하기

패키지 dply은 데이터 프레임을 정렬하는 직관적인 방법을 제공한다. 만약 tb의 cyl 열을 기준으로 정렬을 하고 싶다면, tb %>% arrange(cyl)라고 쓴다. 내림차순 정렬은 tb %>% arrange(desc(cyl))라고 쓴다.

기존의 방법과 비교해 보자.

dplyr 패키지 함수 활용	기존의 방법
`tb %>% arrange(cyl)`	`tb[order(tb$cyl), ]`
`tb %>% arrange(desc(cyl))`	`tb[order(tb$cyl, decreasing = T), ]`

만약 cyl의 올림차순, hp의 내림차순으로 정렬하고 싶다면 기존의 방법으로는 복잡해진다. 하지만 arrange()와 desc()를 사용하면 다음과 같이 간단하게 쓸 수 있다.

```
> tb3 %>% arrange(cyl, desc(qsec))
# A tibble: 3 x 4
     hp   cyl  qsec `hp/cyl`
  <dbl> <dbl> <dbl>    <dbl>
1    93     4  18.6     23.2
2   110     6  17.0     18.3
3   110     6  16.5     18.3
```

2.8 요약하기

summarise 함수[2]는 데이터 프레임의 내용을 몇 개의 요약값으로 정리할 수 있게 도와준다. 만약 tb의 hp 열의 평균을 구하고 싶다면 다음의 방법을 사용한다.

```
> tb %>% summarise(mean(hp))
# A tibble: 1 x 1
  `mean(hp)`
       <dbl>
1       147.
> tb %>% summarize(V1 = mean(hp))
# A tibble: 1 x 1
     V1
  <dbl>
1  147.
> tb %>% summarise(hpMean = mean(hp), qsecMedian =median(qsec))
# A tibble: 1 x 2
  hpMean qsecMedian
   <dbl>      <dbl>
1   147.       17.7
```

[2] summarize라고 쓸 수도 있다

결과는 티블 형식이다.(‘# A Tibble: 1 x 1’를 주목하자.) 새로이 생성된 티블의 열이름은 새롭게 설정하지 않는다면 summarsize( ) 괄호 안의 수식이 그대로 지정된다(예. `mean(hp)`). 여러 열을 사용할 수도 있고, 여러 열을 생성할 수도 있다.

```
> tb %>% summarise(newVar1 = mean(hp) + median(qsec))
# A tibble: 1 x 1
  newVar1
    <dbl>
1    164.
> tb %>% summarise(newVar1 = mean(hp), newVar2 = median(qsec))
# A tibble: 1 x 2
  newVar1 newVar2
    <dbl>   <dbl>
1    147.    17.7
```

그리고 새롭게 생성된 열은 바로 다음 열에서 활용할 수도 있다(newVar3 = newVar1 + newVar2)).

```
> tb %>% summarise(newVar1 = mean(hp), newVar2 = median(qsec), newVar3
    = newVar1 + newVar2)
# A tibble: 1 x 3
  newVar1 newVar2 newVar3
    <dbl>   <dbl>   <dbl>
1    147.    17.7    164.
>
> data.frame(mean(tb$hp), median(tb$qsec), newVar1+newVar2)
Error in data.frame(mean(tb$hp), median(tb$qsec), newVar1 + newVar2) :
  object 'newVar1' not found
```

2.9 집단별로 나누기

요약하는 함수 summarise는 그 자체로도 의미가 있지만 집단을 나누는 함수 group_by와 함께 자주 쓰인다.

집단별로 나누는 함수 group_by는 특정한 열의 값에 따라 티블(또는 데이터 프레임)을 나누는 역할을 한다. 다음의 예를 보자.

```
> tb3 %>% group_by(cyl)
```

```
# A tibble: 3 x 4
# Groups:   cyl [2]
     hp   cyl  qsec `hp/cyl`
  <dbl> <dbl> <dbl>    <dbl>
1   110     6  16.5     18.3
2   110     6  17.0     18.3
3    93     4  18.6     23.2
> tb3_grp <- tb3 %>% group_by(cyl)
> class(tb3_grp)
[1] "grouped_df" "tbl_df"     "tbl"        "data.frame"
```

group_by의 결과는 그룹이 나눠진 티블로 데이터 프레임과도 호환이 된다 (group_by의 결과에서 '# Groups: cyl [2]'를 주목하자). 집단이 나눠진 티블은 특히 summarise 함수와 do 함수와 함께 사용된다.

2.10 집단별로 요약하기

group_by와 summarise 함수를 %>%로 연결하여 주어진 티블을 집단별로 요약할 수 있다. 예를 들어 tb(mtcars)에서 자동 기어(automatic)와 수동 기어(manual)로 집단을 나눈 뒤, 0.25 마일 가속 시간(qsec)의 평균을 구하고자 한다면 다음의 명령을 활용할 수 있다.

```
> tb %>% group_by(am) %>% summarise(mean(qsec))
# A tibble: 2 x 2
     am `mean(qsec)`
  <dbl>        <dbl>
1     0         18.2
2     1         17.4
```

그 결과는 위에서 확인할 수 있다. 수동(am = 0)일 때, 0.25 마일 가속 시간 평균은 18.2초이고, 자동일 때, 0.25 마일 가속 시간 평균은 17.4초임을 확인할 수 있다.

2.11 집단별로 나눈 티블에 대해 함수 적용하기

앞에서 요약을 하기 위해 사용한 summarise( )의 괄호 안에 쓰이는 함수는 결과로 하나의 값을 반환해야 한다. 만약 range( )처럼 두 개의 값(최소값과 최대값)을 반환하는 경우에는 다음과 같이 에러가 발생한다. 그리고 head처럼 데이터 프레임을 입력으로 받아 데이터 프레임을 반환하는 함수도 쓸 수 없다.

```
> tb %>% summarise(range(hp))
Error in summarise_impl(.data, dots) :
  Column `range(hp)` must be length 1 (a summary value), not 2
```

각 집단으로 나눠진 데이터 테이블에 대해 함수를 적용하여 데이터 테이블이 반환되는 경우에는 do( ) 함수를 쓸 수 있다. 다음의 예를 보자.

```
> tb %>% group_by(am) %>% do(head(., n=2))
# A tibble: 4 x 11
# Groups:   am [2]
    mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
2  18.7     8   360   175  3.15  3.44  17.0     0     0     3     2
3  21       6   160   110  3.9   2.62  16.5     0     1     4     4
4  21       6   160   110  3.9   2.88  17.0     0     1     4     4
```

위의 예에서 tb은 열 am의 값에 따라 두 데이터 프레임을 나눠진 후, 각 데이터 프레임에 head(., n=2)가 적용된다. 이렇게 생성된 두 데이터 프레임이 합쳐져서 결과 데이터 프레임이 생성된다.

집단 별로 적용되는 함수는 결과가 데이터 프레임이어야 함을 유의하자. 예를 들어 summary 함수는 데이터 프레임을 입력으로 받지만 결과는 데이터 프레임이 아니다.

```
> tb %>% group_by(am) %>% do(summary(.))
Error: Results 1, 2 must be data frames, not table
```

하지만 as.data.frame을 활용하여 쉽게 데이터 프레임으로 바꿀 수 있다.

```
> tb %>% group_by(am) %>% do(as.data.frame(summary(.))) %>% slice(1:3)
# A tibble: 6 x 4
```

```
# Groups:   am [2]
     am Var1  Var2       Freq
  <dbl> <fct> <fct>      <chr>
1     0 ""    "     mpg" "Min.   :10.40  "
2     0 ""    "     mpg" "1st Qu.:14.95  "
3     0 ""    "     mpg" "Median :17.30  "
4     1 ""    "     mpg" "Min.   :15.00  "
5     1 ""    "     mpg" "1st Qu.:21.00  "
6     1 ""    "     mpg" "Median :22.80  "
Warning messages:
1: In bind_rows_(x, .id) : Unequal factor levels: coercing to character
2: In bind_rows_(x, .id) :
  binding character and factor vector, coercing into character vector
3: In bind_rows_(x, .id) :
  binding character and factor vector, coercing into character vector
```

2.12 패키지 dplyr을 활용하여 데이터 가공하기 종합

앞에서 소개한 함수들을 사용하여 주어진 데이터 프레임 tb에서 필요한 부분을 선별하고 집단별로 나눈 후 가공하는 절차는 보통 다음의 순서를 따른다.

`tb %>% select() %>% filter() %>% group_by() %>% summarise(), do(), arrange(, .by_group=T)`

2.13 그 밖의 편의 기능: _all, _at, _if와 vars(), funs()

앞서 새로운 열을 만들 때 mutate 함수를 사용했다. 예를 들어 mtcars의 qsec 열에 지수함수 exp를 적용하여 새로운 열을 생성한다면 다음과 같다.

```
mtcars %>% mutate(exp(qsec))
```

만약 모든 열에 대해 지수 함수 exp를 적용해야 한다면 어떻게 해야 하나? 크게 다를 것이 없다. 단지 손이 힘들 뿐.[3]

[3] 저자는 다음의 코드를 활용했다.

```
coln0 <- colnames(mtcars); coln <- colnames(mtcars); substr(coln, 1,
1) = toupper(substr(coln, 1, 1)); coln <- paste('exp', coln, sep='');
paste('mutate(', paste(coln, "=exp(", coln0,")", sep='', collapse=', '), ')',
sep='')
```

```
> mtcars %>% mutate(expMpg=exp(mpg), expCyl=exp(cyl), expDisp=exp(disp)
    , expHp=exp(hp), expDrat=exp(drat), expWt=exp(wt), expQsec=exp(
    qsec), expVs=exp(vs), expAm=exp(am), expGear=exp(gear), expCarb=
    exp(carb)) %>% head(n=3)
   mpg cyl disp  hp drat    wt  qsec vs am gear carb     expMpg
1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4 1318815734
2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4 1318815734
3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1 7978370264
     expCyl       expDisp        expHp  expDrat    expWt   expQsec
1 403.42879 3.069850e+69 5.920972e+47 49.40245 13.73572  14076257
2 403.42879 3.069850e+69 5.920972e+47 49.40245 17.72542  24642915
3  54.59815 8.013164e+46 2.451246e+40 46.99306 10.17567 120842669
     expVs    expAm  expGear   expCarb
1 1.000000 2.718282 54.59815 54.598150
2 1.000000 2.718282 54.59815 54.598150
3 2.718282 2.718282 54.59815  2.718282
```

물론 이것도 한 방법이지만 처음 이 코드를 본 사람은 꽤나 어리둥절할 것이다. 하지만 이 코드가 수행하는 일은 '모든 열에 대해 지수함수 exp를 적용하라'이다. 개념적으로는 꽤나 단순한 것이다. dplyr에서는 이렇게 모든 열에 동일한 함수를 적용하는 경우을 위해 mutate_all이라는 함수를 마련해 놓았다. mutate_all 함수를 쓴다면 위의 코드는 다음과 같이 단순해 진다.

```
> mtcars %>% mutate_all(exp) %>% head(n=3)
         mpg       cyl          disp           hp      drat
1 1318815734 403.42879 3.069850e+69 5.920972e+47 49.40245
2 1318815734 403.42879 3.069850e+69 5.920972e+47 49.40245
3 7978370264  54.59815 8.013164e+46 2.451246e+40 46.99306
        wt      qsec       vs       am     gear       carb
1 13.73572  14076257 1.000000 2.718282 54.59815 54.598150
2 17.72542  24642915 1.000000 2.718282 54.59815 54.598150
3 10.17567 120842669 2.718282 2.718282 54.59815  2.718282
```

하지만 두 코드는 완전히 동일하지는 않다. mutate의 경우 기존의 열이 보존되지만, mutate_all의 경우 기존의 열에 함수가 적용된 결과가 덮어씌워진다. 어쨋든 mutate_all의 _all은 모든 열에 적용됨을 시사한다.

_all는 dplyr의 거의 모든 함수의 뒤에 붙어서 새로운 함수를 나타낸다. 그리고 열을 선택하는 방법을 나타내는 접미사는 _all 이외에도 _at과 _if가 있다.

다음의 표를 보자.

	_all	_at	_if
select	select_all	select_at	select_if
mutate	mutate_all	mutate_at	mutate_if
transmute	transmute_all	transmute_at	transmute_if
group_by	group_by_all	group_by_at	group_by_if
summarise	summarise_all	summarise_at	summarise_if

먼저 mutate를 활용해서 _at과 _if를 설명해보자. _at의 경우는 함수를 적용할 열의 이름이 변수(문자열 벡터)에 저장되어 있는 경우에 쓸 수 있다. 다음의 예제를 보자.

```
> coln = c('cyl', 'disp', 'drat', 'carb')
> mtcars %>% mutate_at(coln, exp) %>% head(n=3)
   mpg       cyl          disp  hp     drat    wt  qsec vs am gear
1 21.0 403.42879 3.069850e+69 110 49.40245 2.620 16.46  0  1    4
2 21.0 403.42879 3.069850e+69 110 49.40245 2.875 17.02  0  1    4
3 22.8  54.59815 8.013164e+46  93 46.99306 2.320 18.61  1  1    4
       carb
1 54.598150
2 54.598150
3  2.718282
```

다른 열은 모두 보존이 되었고, 문자열 벡터의 원소 'cyl', 'disp', 'drat'에 해당하는 열에 지수함수 exp가 적용되었다. 그런데 생각해보면 열을 선택하는 명령은 따로 존재하지 않는가? select! 다음의 예와 비교를 해보자.

```
> mtcars %>% select(starts_with('c'), starts_with('d')) %>% mutate_all(
    exp) %>% head(n=3)
       cyl      carb         disp     drat
1 403.42879 54.598150 3.069850e+69 49.40245
2 403.42879 54.598150 3.069850e+69 49.40245
3  54.59815  2.718282 8.013164e+46 46.99306
```

결과는 거의 똑같다. select의 경우는 열 이름을 따옴표 안에 쓰지 않아도 되고, starts_with, ends_with와 같은 함수도 쓸 수 있다는 장점이 있다. 만약 mutate_at 함수에서 select와 같은 방법으로 열을 선택하려면 vars라는 함수를

쓸 수 있다.

```
> mtcars %>% mutate_at(vars(starts_with('c'), starts_with('d')), exp)
    %>% head(n=3)
   mpg       cyl          disp  hp     drat    wt  qsec vs am gear
1 21.0 403.42879 3.069850e+69 110 49.40245 2.620 16.46  0  1    4
2 21.0 403.42879 3.069850e+69 110 49.40245 2.875 17.02  0  1    4
3 22.8  54.59815 8.013164e+46  93 46.99306 2.320 18.61  1  1    4
       carb
1 54.598150
2 54.598150
3  2.718282
```

마지막 mutate_if는 특정한 조건을 만족하는 열만을 선택해서 함수를 적용한다. 만약 열의 총합이 100 미만이 열에 대해서만 지수 함수 exp를 적용하고 싶다면 다음과 같이 쓸 수 있다.

```
> mtcars %>% mutate_if(function(x) sum(x)<100, exp) %>% head(n=3)
   mpg cyl disp  hp drat    wt  qsec       vs       am gear       carb
1 21.0   6  160 110 3.90 2.620 16.46 1.000000 2.718282    4 54.598150
2 21.0   6  160 110 3.90 2.875 17.02 1.000000 2.718282    4 54.598150
3 22.8   4  108  93 3.85 2.320 18.61 2.718282 2.718282    4  2.718282
```

이때 한 가지 문제는 열의 이름이 보존되어 있기 때문에 지수 함수가 어떤 열에 적용되었는지 쉽게 알기 힘들다는 단점이 있다. 만약 새로운 열을 생성하면서 함수가 적용되지 않는 열은 제거하고 싶다면 transmute 함수를 사용한다. 다음의 예를 보면 transmute의 역할을 쉽게 이해할 수 있을 것이다.

```
> mtcars %>% transmute(expCarb = exp(carb)) %>% head(n=3)
    expCarb
1 54.598150
2 54.598150
3  2.718282
> mtcars %>% transmute_if(function(x) sum(x)<100, exp) %>% head(n=3)
       vs       am      carb
1 1.000000 2.718282 54.598150
2 1.000000 2.718282 54.598150
3 2.718282 2.718282  2.718282
```

그리고 transmute_all의 결과를 예상해보면 mutate_all과 동일할 것이다.

transmute_if에서 두 번째 인자는 열 벡터를 입력하면 참 또는 진리값을 출력하는 함수이고, 이 함수를 통해 어떤 열에 함수를 적용할지가 결정된다. 이때 미리 마련된 함수가 있지 않다면 function(x) ...과 같은 부분이 추가될 것인데, dplyr에서는 이 부분을 보기 좋게 만들 수 있는 방법이 있다. 다음의 예를 보자.

```
> mtcars %>% transmute_if(function(x) sum(x)<100, exp) %>% head(n=3)
        vs       am      carb
1 1.000000 2.718282 54.598150
2 1.000000 2.718282 54.598150
3 2.718282 2.718282  2.718282
> mtcars %>% transmute_if(funs(sum(.)<100), exp) %>% head(n=3)
        vs       am      carb
1 1.000000 2.718282 54.598150
2 1.000000 2.718282 54.598150
3 2.718282 2.718282  2.718282
```

vars는 열을 선택할 때 편의를 제공하고, funs는 함수를 만들 때 편의를 제공하는 함수라고 생각하면 편하다.

다음의 예를 보자 그 의미를 파악해보자.

```

> mtcars %>% mutate_if(funs(sum(.) >= 100), funs(paste(.,"+",sep="")))
    %>% head(n=3)
    mpg cyl disp   hp  drat     wt   qsec vs am gear carb
1   21+  6+ 160+ 110+  3.9+  2.62+ 16.46+  0  1   4+    4
2   21+  6+ 160+ 110+  3.9+ 2.875+ 17.02+  0  1   4+    4
3 22.8+  4+ 108+  93+ 3.85+  2.32+ 18.61+  1  1   4+    1
> mtcars %>% transmute_at(vars(starts_with('d')), exp) %>% head(n=3)
          disp     drat
1 3.069850e+69 49.40245
2 3.069850e+69 49.40245
3 8.013164e+46 46.99306
```