Part2 - Wrangle

IMSOHYUN

2019 년 3 월 14 일

```
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
```

1. Explore the distribution of each of the `x`, `y`, and `z` variables in `diamonds`. What do you learn? Think about a diamond and how you might decide which dimension is the length, width, and depth.

x,y,z 모두 왼쪽으로 치우친 형태를 보이고 있다. max 값과 Q3 값을 비교해 보았을 때, 차이가 많이 나는 y 와 z 에서 아웃라이어가 발견되었다. Z 의 분포 값이 제일 작은 것으로 보아. 다이아몬드의 'depth'라고 생각이 든다. X 와 y 의 값은 거의 비슷하지만 '길이'에서 이상점이 존재할 확률이 크다고 생각하여, x 가 'width', y 가 'length'라고 생각이 든다.

```
quantile(diamonds$x)
```

```
##    0%   25%   50%   75%  100%
##  0.00  4.71  5.70  6.54 10.74
```

```
quantile(diamonds$y)
```

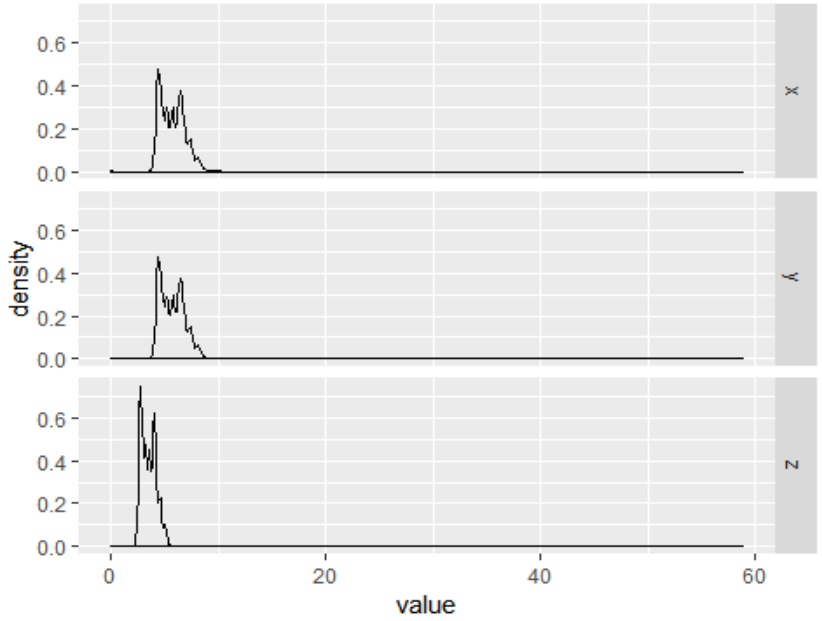
```
##    0%   25%   50%   75%  100%
##  0.00  4.72  5.71  6.54 58.90
```

```
quantile(diamonds$z)
```

```
##    0%   25%   50%   75%  100%
##  0.00  2.91  3.53  4.04 31.80
```

```
diamonds %>%
  mutate(id = row_number()) %>%
  select(x, y, z, id) %>%
```

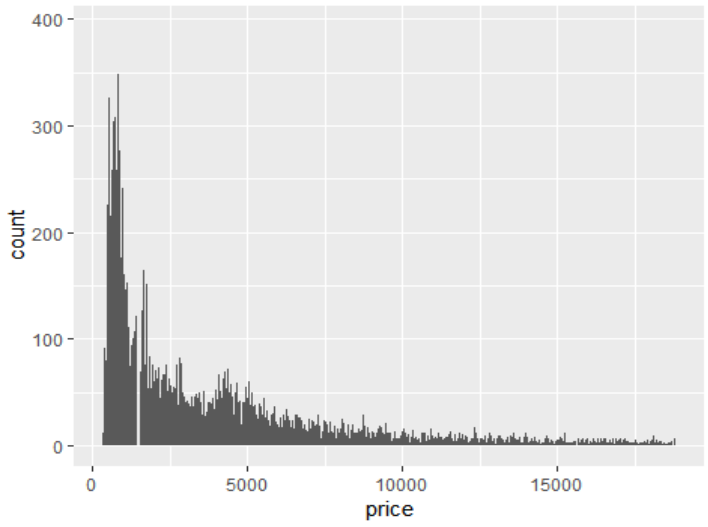
```
  gather(variable, value, -id) %>%
  ggplot(aes(x = value)) +
  geom_density() +
  facet_grid(variable ~ .)
```



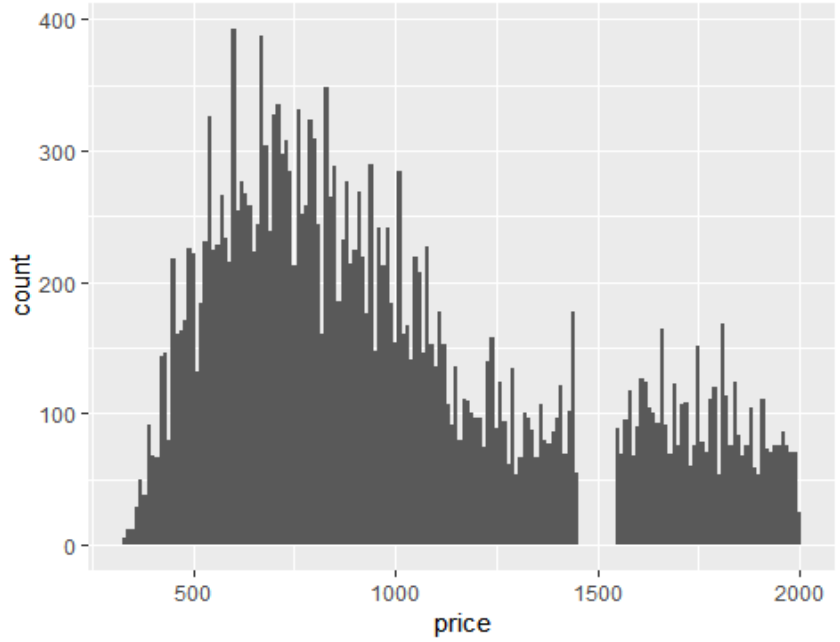
2. Explore the distribution of `price`. Do you discover anything unusual or surprising?

비싼 다이아몬드일수록 희귀하다고 생각이 들었고 분포 역시 왼쪽으로 치우친 형태를 보였다. 하지만 중간에 비어 있는 부분이 있어, 그 부분만 확인해보았더니 이상하게도 가격이 1500 인 다이아몬드는 존재하지 않았다. 또한 많은 수의 다이아몬드가 가격이 500-700 선에서 존재하였다.

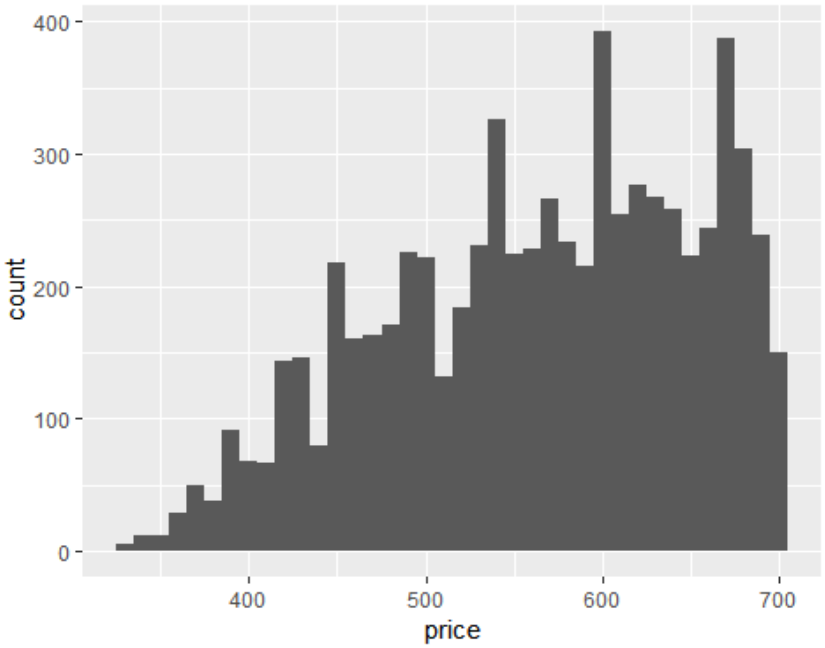
```
ggplot(diamonds, aes(x = price)) + geom_histogram(binwidth = 10)
```



```r
ggplot(filter(diamonds, price < 2000), aes(x = price)) +
  geom_histogram(binwidth = 10)
```




```r
ggplot(filter(diamonds, price < 700), aes(x = price)) +
  geom_histogram(binwidth = 10)
```



3. How many diamonds are 0.99 carat? How many are 1 carat? What do you think is the cause of the difference?

0.99 캐럿은 23 개, 1 캐럿은 1558 개로, 0.001 차이지만 거의 70 배의 차이가 났다. 정확한 이유는 모르겠지만 0.01 단위로, 단위가 작아서 값을 반올림하였을 것 같다.

```
diamonds %>%
  filter(carat >= 0.99, carat <= 1) %>% count(carat)
```

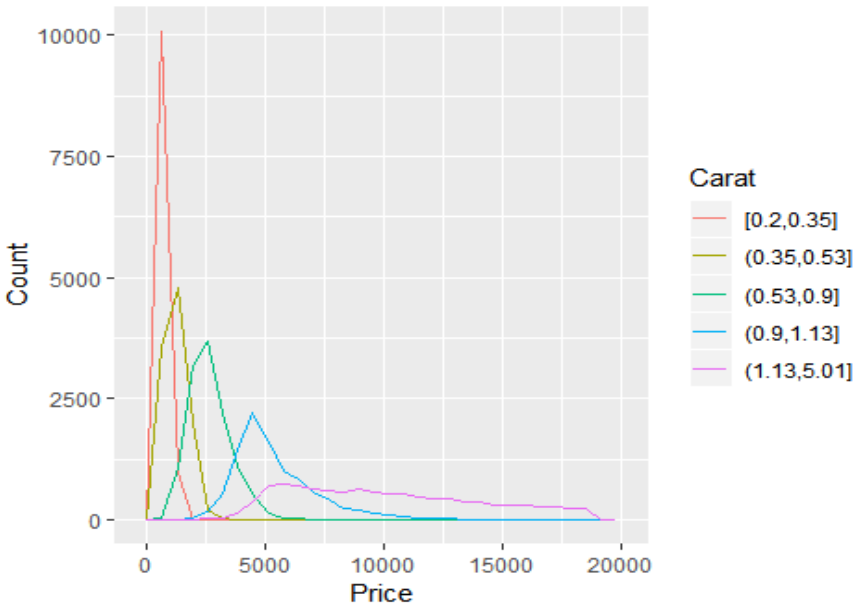
```
## # A tibble: 2 x 2
##   carat     n
##   <dbl> <int>
## 1  0.99    23
## 2  1     1558
```

4. Instead of summarising the conditional distribution with a boxplot, you could use a frequency polygon. What do you need to consider when using `cut_width()` vs `cut_number()`? How does that impact a visualisation of the 2d distribution of `carat` and `price`?

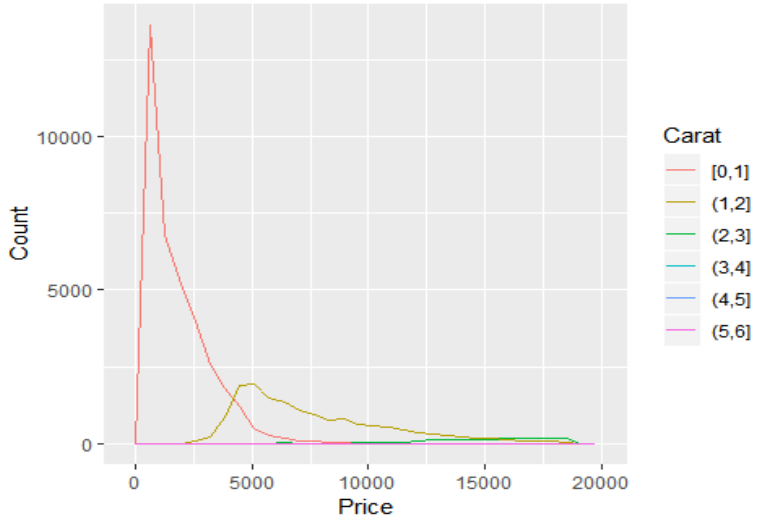
'cut_width()'와 'cut_number' 모두 구간을 나누는 의미이다. 'cut_width()'는 간격을 정해서 나눌수 있고, 'cut_number'는 나눌 구간의 수를 정하면 구간이 자동으로 정해진다. 여기서는 캐럿이 2 이상인 다이아몬드가 거의 존재하지 않으므로 구간의 수를 정하는 'cut_number'가 유용하다고 생각한다.

```
ggplot(diamonds, aes(color = cut_number(carat, 5), x = price)) +
  geom_freqpoly() +
  labs(x = "Price", y = "Count", color = "Carat")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



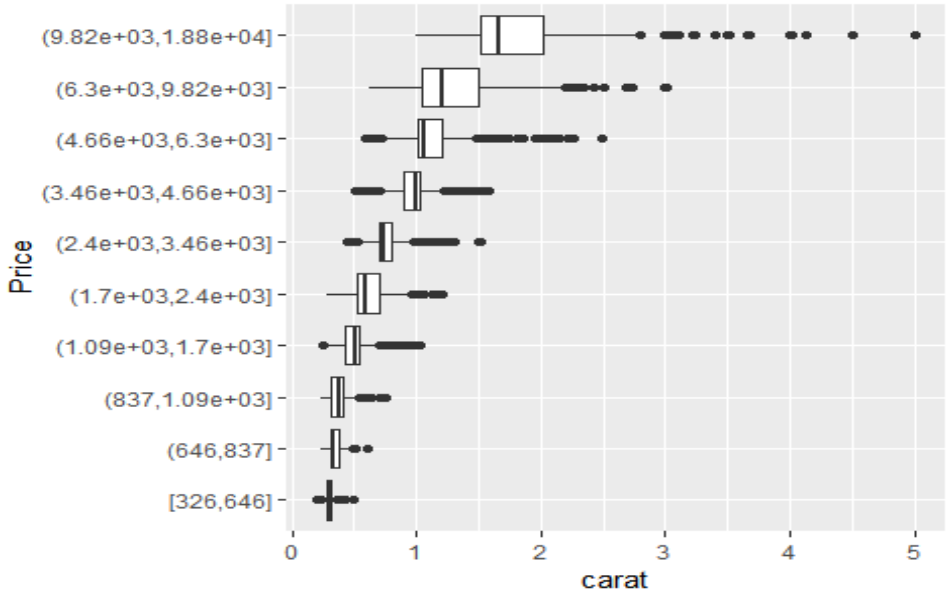
```r
ggplot(diamonds, aes(color = cut_width(carat, 1, boundary = 0), x = price)) +
  geom_freqpoly() +
  labs(x = "Price", y = "Count", color = "Carat")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



5. Visualise the distribution of carat, partitioned by price.

다이아몬드 총 개수를 10 개의 구간으로 그린 box plot 이다. 캐럿이 커질수록 가격이 비싸지는 양의 상관관계로 보인다. 캐럿이 1 이상인 다이아몬드일수록 가격이 증감폭이 커지는 것을 알 수 있다.

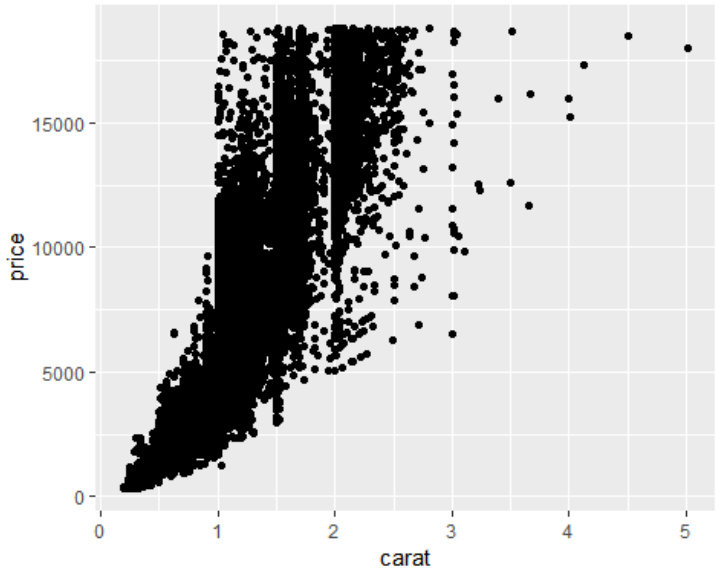
```r
ggplot(diamonds, aes(x = cut_number(price, 10), y = carat)) +
  geom_boxplot() + coord_flip() + xlab("Price")
```



6 . How does the price distribution of very large diamonds compare to small diamonds. Is it as you expect, or does it surprise you?

다이아몬드가 클수록 가격이 비싸지는 건 당연한 사실이라고 생각한다. 하지만 캐럿이 같더라도 가격이 천차만별 한 것을 알았다. 이는 다이아몬드 세공과정의 차이라고 생각이 든다.

```
ggplot(diamonds) + geom_point(aes(carat,price))
```



7. Compute the `rate` for `table2`, and `table4a` + `table4b`. You will need to perform four operations:

1) Extract the number of TB cases per country per year.

```
# table2 : 나라, 년도별 cases 와 population 의 값으로 되어있는 tibble
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

```
filter(table2, type == "cases") %>%
  rename(cases = count) %>% select(country, year, cases) %>%
  arrange(country, year)
```

```
## # A tibble: 6 x 3
##   country      year  cases
##   <chr>       <int>  <int>
## 1 Afghanistan  1999    745
## 2 Afghanistan  2000   2666
## 3 Brazil       1999  37737
## 4 Brazil       2000  80488
## 5 China        1999 212258
## 6 China        2000 213766
```

2) Extract the matching population per country per year.

```
filter(table2, type == "population") %>%
  rename(population = count) %>% select(country, year, population) %>%
  arrange(country, year)
```

```
## # A tibble: 6 x 3
##   country      year population
##   <chr>       <int>      <int>
## 1 Afghanistan  1999   19987071
## 2 Afghanistan  2000   20595360
## 3 Brazil       1999  172006362
## 4 Brazil       2000  174504898
## 5 China        1999 1272915272
## 6 China        2000 1280428583
```

3) Divide cases by population, and multiply by 10000.

```
# table4a : 나라, 년도별 cases 값
# table4b : 나라, 년도별 population 값
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

```
table4b
```

```
## # A tibble: 3 x 3
##   country         `1999`     `2000`
## * <chr>            <int>      <int>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
```

```
table4c <-
  tibble(
```

```
    country = table4a$country,
    `1999` = table4a[["1999"]] / table4b[["1999"]] * 10000,
    `2000` = table4a[["2000"]] / table4b[["2000"]] * 10000
  )
table4c
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
##   <chr>        <dbl>  <dbl>
## 1 Afghanistan  0.373   1.29
## 2 Brazil       2.19    4.61
## 3 China        1.67    1.67
```

4) Store back in the appropriate place.

각각의 데이터를 cases, polulation, table4c라는 이름으로 저장시켰다.

8. Why are `gather()` and `spread()` not perfectly symmetrical? Carefully consider the following example:

```
stocks <- tibble(
  year = c(2015, 2015, 2016, 2016),
  half = c(1, 2, 1, 2),
  return = c(1.88, 0.59, 0.92, 0.17)
)

stocks %>%
  spread(year, return) %>%
  gather(`2015`:`2016`, key = "year", value = "return")
```

```
## # A tibble: 4 x 3
##    half year  return
##   <dbl> <chr>  <dbl>
## 1     1 2015    1.88
## 2     2 2015    0.59
## 3     1 2016    0.92
## 4     2 2016    0.17
```

Spread 는 데이터를 나누는 함수고, gather 은 데이터를 모으는 함수라서 spread()와 gather()을 같이 쓰면 원래 데이터로 돌아온다고 생각하기 쉽다. 하지만 이 예제를 보면 원데이터와 똑같이 복구 되지 않는다. 원래 데이터에서 year 은 numeric 이라고 표기된다. 하지만 spread()를 하면서 character 이 변수명이 되면서 year 의 값이 numeric 임에도 불구하고 character 이라고 나타난다. 이는 spread() 함수를 사용하면서 원래 데이터의 변수명을 잃기 때문이라고 생각한다.

```
stocks
```

```
## # A tibble: 4 x 3
##    year  half return
```

```
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2016     1   0.92
## 4  2016     2   0.17
```

```
stocks %>%
  spread(year, return)
```

```
## # A tibble: 2 x 3
##    half `2015` `2016`
##   <dbl>  <dbl>  <dbl>
## 1     1   1.88   0.92
## 2     2   0.59   0.17
```

9. Tidy the simple tibble below. Do you need to spread or gather it? What are the variables?

```
preg <- tribble(
  ~pregnant, ~male, ~female,
  "yes", NA, 10,
  "no", 20, 12
)
preg
```

```
## # A tibble: 2 x 3
##   pregnant  male female
##   <chr>    <dbl>  <dbl>
## 1 yes         NA     10
## 2 no          20     12
```

변수들을 보면 pregnant, male, female 로 임신여부, 성별로 나눌수 있다. 먼저 spread()에서는 임신여부로 나타낼 수 있는 pregnant 를 쓸 수 있다. 다음 두 예제를 보았을 때, spread()가 적절한 결과를 보여주지 않는다.

```
preg %>%
  spread(male,pregnant)
```

```
## # A tibble: 2 x 3
##   female `20`  `<NA>`
##    <dbl> <chr> <chr>
## 1     10 <NA>  yes
## 2     12 no    <NA>
```

```
preg %>%
  spread(pregnant,male)
```

```
## # A tibble: 2 x 3
##   female    no   yes
##    <dbl> <dbl> <dbl>
## 1     10    NA    NA
## 2     12    20    NA
```

다음은 gather() 함수이다. male, female 를 성별로 묶을 수 있기 때문에 임신여부, 성별에 따른 count 를 알 수 있는 데이터가 형성될 수 있다.

```
preg %>%
  gather(male, female, key = "sex", value = "count")
```

```
## # A tibble: 4 x 3
##   pregnant sex    count
##   <chr>    <chr>  <dbl>
## 1 yes      male      NA
## 2 no       male      20
## 3 yes      female    10
## 4 no       female    12
```