Graphics Assignment#2

182STG15 이시영

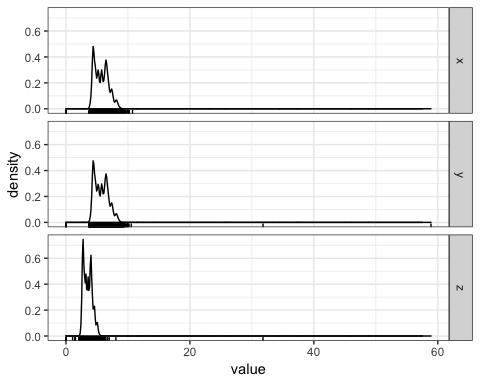
2019/3/13

## 1.

head(diamonds)

## # A tibble: 6 x 10  
## carat cut color clarity depth table price x y z  
## <dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 0.230 Ideal E SI2 61.5 55. 326 3.95 3.98 2.43  
## 2 0.210 Premium E SI1 59.8 61. 326 3.89 3.84 2.31  
## 3 0.230 Good E VS1 56.9 65. 327 4.05 4.07 2.31  
## 4 0.290 Premium I VS2 62.4 58. 334 4.20 4.23 2.63  
## 5 0.310 Good J SI2 63.3 58. 335 4.34 4.35 2.75  
## 6 0.240 Very Good J VVS2 62.8 57. 336 3.94 3.96 2.48

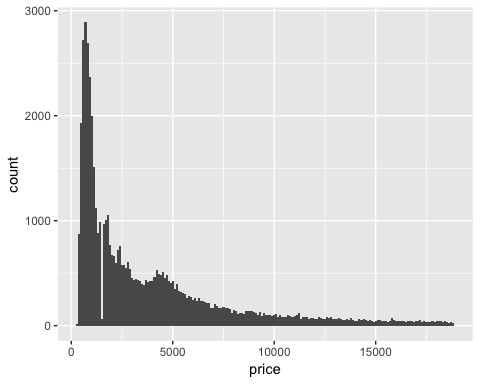
diamonds %>%  
 mutate(id = row\_number()) %>%  
 select(x, y, z, id) %>%  
 gather(variable, value, -id) %>%  
 ggplot(aes(x = value)) +  
 geom\_density() +  
 geom\_rug() +  
 facet\_grid(variable ~ .) +  
 theme\_bw()



x, y, z의 분포가 모드 right skewed임을 알 수 있다. 대부분의 다이아몬드의 크기는 작으나 몇 개의 outlier들이 존재한다(y & z). z는 x 와 y보다는 작은 값들을 가진다.

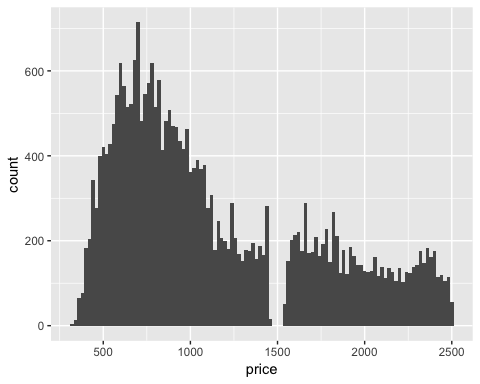
## 2.

ggplot(diamonds, aes(price)) + geom\_histogram(binwidth = 100)



5000미만의 가격에서 데이터가 몰려있는 것을 관측할 수 있다. 또한 다이아몬드가 존재하지 않는 가격대가 있다. 가격이 5000미만인 다이아몬드의 정보를 걸러서 다시 히스토그램을 그려보았을 때,

ggplot(filter(diamonds, price < 2500), aes(x=price)) + geom\_histogram(binwidth = 20)



가격이 1500인 부근에서 다이아모든 관측치가 없는 것을 알 수 있다.

## 3.

## [1] "0.2" "0.21" "0.22" "0.23" "0.24" "0.25" "0.26" "0.27" "0.28" "0.29"  
## [11] "0.3" "0.31" "0.32" "0.33" "0.34" "0.35" "0.36" "0.37" "0.38" "0.39"  
## [21] "0.4" "0.41" "0.42" "0.43" "0.44" "0.45" "0.46" "0.47" "0.48" "0.49"  
## [31] "0.5" "0.51" "0.52" "0.53" "0.54" "0.55" "0.56" "0.57" "0.58" "0.59"  
## [41] "0.6" "0.61" "0.62" "0.63" "0.64" "0.65" "0.66" "0.67" "0.68" "0.69"  
## [51] "0.7" "0.71" "0.72" "0.73" "0.74" "0.75" "0.76" "0.77" "0.78" "0.79"  
## [61] "0.8" "0.81" "0.82" "0.83" "0.84" "0.85" "0.86" "0.87" "0.88" "0.89"  
## [71] "0.9" "0.91" "0.92" "0.93" "0.94" "0.95" "0.96" "0.97" "0.98" "0.99"  
## [81] "1" "1.01" "1.02" "1.03" "1.04" "1.05" "1.06" "1.07" "1.08" "1.09"  
## [91] "1.1" "1.11" "1.12" "1.13" "1.14" "1.15" "1.16" "1.17" "1.18" "1.19"  
## [101] "1.2" "1.21" "1.22" "1.23" "1.24" "1.25" "1.26" "1.27" "1.28" "1.29"  
## [111] "1.3" "1.31" "1.32" "1.33" "1.34" "1.35" "1.36" "1.37" "1.38" "1.39"  
## [121] "1.4" "1.41" "1.42" "1.43" "1.44" "1.45" "1.46" "1.47" "1.48" "1.49"  
## [131] "1.5" "1.51" "1.52" "1.53" "1.54" "1.55" "1.56" "1.57" "1.58" "1.59"  
## [141] "1.6" "1.61" "1.62" "1.63" "1.64" "1.65" "1.66" "1.67" "1.68" "1.69"  
## [151] "1.7" "1.71" "1.72" "1.73" "1.74" "1.75" "1.76" "1.77" "1.78" "1.79"  
## [161] "1.8" "1.81" "1.82" "1.83" "1.84" "1.85" "1.86" "1.87" "1.88" "1.89"  
## [171] "1.9" "1.91" "1.92" "1.93" "1.94" "1.95" "1.96" "1.97" "1.98" "1.99"  
## [181] "2" "2.01" "2.02" "2.03" "2.04" "2.05" "2.06" "2.07" "2.08" "2.09"  
## [191] "2.1" "2.11" "2.12" "2.13" "2.14" "2.15" "2.16" "2.17" "2.18" "2.19"  
## [201] "2.2" "2.21" "2.22" "2.23" "2.24" "2.25" "2.26" "2.27" "2.28" "2.29"  
## [211] "2.3" "2.31" "2.32" "2.33" "2.34" "2.35" "2.36" "2.37" "2.38" "2.39"  
## [221] "2.4" "2.41" "2.42" "2.43" "2.44" "2.45" "2.46" "2.47" "2.48" "2.49"  
## [231] "2.5" "2.51" "2.52" "2.53" "2.54" "2.55" "2.56" "2.57" "2.58" "2.59"  
## [241] "2.6" "2.61" "2.63" "2.64" "2.65" "2.66" "2.67" "2.68" "2.7" "2.71"  
## [251] "2.72" "2.74" "2.75" "2.77" "2.8" "3" "3.01" "3.02" "3.04" "3.05"  
## [261] "3.11" "3.22" "3.24" "3.4" "3.5" "3.51" "3.65" "3.67" "4" "4.01"  
## [271] "4.13" "4.5" "5.01"

carat은 소수 둘째짜리까지 Count.

diamonds %>% filter(carat == 0.99 | carat == 1) %>%   
 count(carat)

## # A tibble: 2 x 2  
## carat n  
## <dbl> <int>  
## 1 0.990 23  
## 2 1.00 1558

1캐럿 다이아가 0.99캐럿 다이아에 비해 약 68배 더 많다.

diamonds %>% filter(0.9 <= carat & carat <= 1.1) %>%   
 count(carat)

## # A tibble: 21 x 2  
## carat n  
## <dbl> <int>  
## 1 0.900 1485  
## 2 0.910 570  
## 3 0.920 226  
## 4 0.930 142  
## 5 0.940 59  
## 6 0.950 65  
## 7 0.960 103  
## 8 0.970 59  
## 9 0.980 31  
## 10 0.990 23  
## # ... with 11 more rows

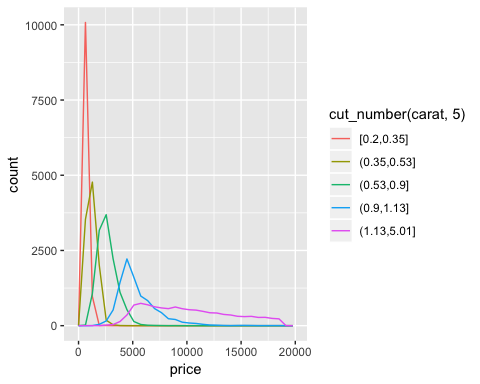
0.9캐럿, 1.0이 많이 관측되는 것으로 보아 0.99와 같이 근사한 값들은 반올림하는 경향이 있는 것 같다.

## 4.

cut\_width()와 cut\_number()는 변수를 기준에 따라 그룹으로 나눈다. cut\_width()는 나눌시 너비를 이용하고, bin의 수는 자동으로 계산된다. cut\_number()는 bin 수를 지정해야하고 너비는 자동을 계산된다. ggplot에서 범주는 최대 8개까지 표현이 가능한다. 따라서 cut\_width() 사용 시 조심해야한다.

ggplot(diamonds, aes(x=price, color = cut\_number(carat, 5))) + geom\_freqpoly()

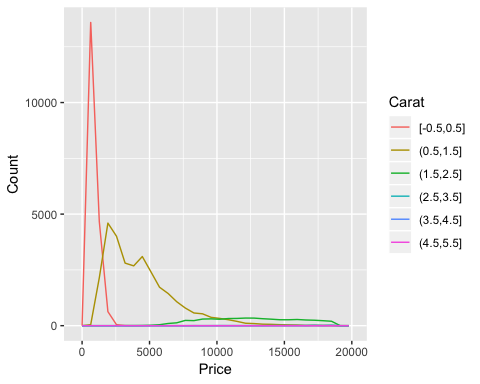
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



carat을 5개의 분위로 나눠서 frequency plot을 그려보았다. 캐럿수가 클수록 frequency는 감소하나 가격은 증가한다.

ggplot(diamonds, aes(x=price, color = cut\_width(carat, 1))) + geom\_freqpoly() +  
 labs(x = 'Price', y = 'Count', color = 'Carat')

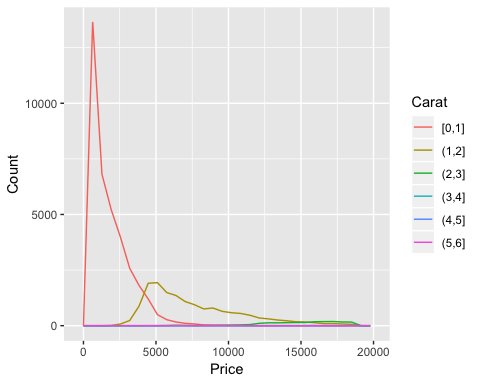
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



0보다 작은 캐럿은 없기에 이 범위를 수정해주기 위하여 cut\_width()에서 boundary값을 0으로 지정해준다.

ggplot(diamonds, aes(x=price, color = cut\_width(carat, 1, boundary = 0))) + geom\_freqpoly() +  
 labs(x = 'Price', y = 'Count', color = 'Carat')

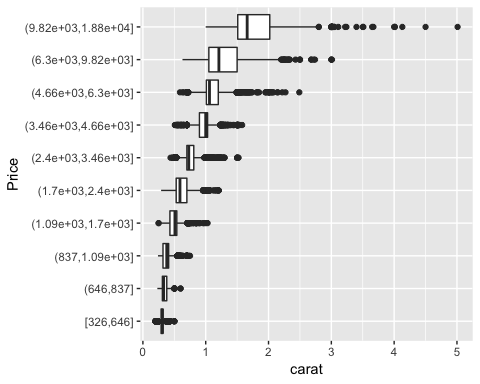
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



캐럿의 크기가 증가할수록 다이아몬드의 수가 적어진다. 또한 가격도 증가한다.

## 5.

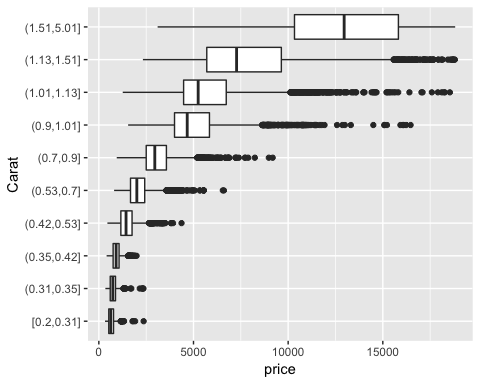
ggplot(diamonds, aes(x=cut\_number(price, 10), y = carat)) +  
 geom\_boxplot() + coord\_flip() + xlab('Price')



price를 10분위로 나눠서 carat의 boxplot을 그려보았을때, price가 증가할수록 carat도 증가하는 경향을 보인다. 또한 높은 가격대에서 carat의 outlier들이 존재한다.

## 6.

ggplot(diamonds, aes(x = cut\_number(carat,10), y=price)) + geom\_boxplot() +  
 coord\_flip() + xlab('Carat')



carat이 클수록 가격대의 distribution이 넓다. carat이 크다고 무조건 가격이 높은 것이 아니다. 다이아몬드의 가격 책정의 또다른 기준(clarity, color, etc.)이 있을 것으로 예상된다. 하지만 carat이 작은 다이아몬드의경우 가격의 변동이 크지 않기에 또 다른 기준(clarity, color, etc.)이 작은 carat의 다이아몬드에는 영향력이 크지 않다.

## 7.

### table2

tb2\_cases <- table2 %>% filter(type == 'cases') %>%   
 rename(cases = count) %>%   
 arrange(country, year)  
tb2\_population <- table2 %>% filter(type == 'population') %>%   
 rename(population = count) %>%   
 arrange(country, year)  
tb2\_per\_cap <- tibble(  
 year = tb2\_cases$year,  
 country = tb2\_cases$country,  
 cases = tb2\_cases$cases,  
 population = tb2\_population$population  
) %>% mutate(cases\_per\_cap = cases/population \* 10000) %>%   
 select(country, year, cases\_per\_cap)  
tb2\_per\_cap <- tb2\_per\_cap %>%   
 mutate(type = 'cases\_per\_cap') %>%   
 rename(count = cases\_per\_cap)  
bind\_rows(table2, tb2\_per\_cap) %>%   
 arrange(country, year, type, count)

## # A tibble: 18 x 4  
## country year type count  
## <chr> <int> <chr> <dbl>  
## 1 Afghanistan 1999 cases 7.45e+2  
## 2 Afghanistan 1999 cases\_per\_cap 3.73e-1  
## 3 Afghanistan 1999 population 2.00e+7  
## 4 Afghanistan 2000 cases 2.67e+3  
## 5 Afghanistan 2000 cases\_per\_cap 1.29e+0  
## 6 Afghanistan 2000 population 2.06e+7  
## 7 Brazil 1999 cases 3.77e+4  
## 8 Brazil 1999 cases\_per\_cap 2.19e+0  
## 9 Brazil 1999 population 1.72e+8  
## 10 Brazil 2000 cases 8.05e+4  
## 11 Brazil 2000 cases\_per\_cap 4.61e+0  
## 12 Brazil 2000 population 1.75e+8  
## 13 China 1999 cases 2.12e+5  
## 14 China 1999 cases\_per\_cap 1.67e+0  
## 15 China 1999 population 1.27e+9  
## 16 China 2000 cases 2.14e+5  
## 17 China 2000 cases\_per\_cap 1.67e+0  
## 18 China 2000 population 1.28e+9

### table4a + table4b

head(table4a)

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

head(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, #cases / population \* 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

## 8.

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)

## # A tibble: 2 x 3  
## half `2015` `2016`  
## <dbl> <dbl> <dbl>  
## 1 1. 1.88 0.920  
## 2 2. 0.590 0.170

stocks %>% spread(year, return) %>%   
 gather('year','return','2015','2016')

## # A tibble: 4 x 3  
## half year return  
## <dbl> <chr> <dbl>  
## 1 1. 2015 1.88   
## 2 2. 2015 0.590  
## 3 1. 2016 0.920  
## 4 2. 2016 0.170

stocks %>% spread(year, return) %>%   
 gather('year','return','2015','2016', convert = T)

## # A tibble: 4 x 3  
## half year return  
## <dbl> <int> <dbl>  
## 1 1. 2015 1.88   
## 2 2. 2015 0.590  
## 3 1. 2016 0.920  
## 4 2. 2016 0.170

gather()을 사용하면 column type(character로 변환)가 사라진다. 반면 spread는 유지된다. gather()에 convert를 사용하면 데이터의 유형을 찾아준다.

## 9.

preg <- tribble(  
 ~pregnant, ~male, ~female,  
 'yes', NA, 10,  
 'no', 20, 12  
)  
  
preg %>% gather(male, female, key = 'gender', value = 'count')

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

NA obs 삭제

preg %>% gather(male, female, key = 'gender', value = 'count', na.rm = T)

## # A tibble: 3 x 3  
## pregnant gender count  
## \* <chr> <chr> <dbl>  
## 1 no male 20.  
## 2 yes female 10.  
## 3 no female 12.

female에 point

preg %>% gather(male, female, key = 'gender', value = 'count', na.rm = T) %>%   
 mutate(female = gender == 'female',  
 pregnant = pregnant == 'yes') %>%   
 select(female, pregnant, count)

## # A tibble: 3 x 3  
## female pregnant count  
## <lgl> <lgl> <dbl>  
## 1 FALSE FALSE 20.  
## 2 TRUE TRUE 10.  
## 3 TRUE FALSE 12.

이와같이 3가지 tibble로 표현할 수 있다.