Exercise DV

BELFADIL Anas

10/14/2019

Exercises (data visualization)

Setting up the environment and loading the necessary libraries

```
knitr::opts_chunk$set(echo = TRUE, message=FALSE, warning=FALSE)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

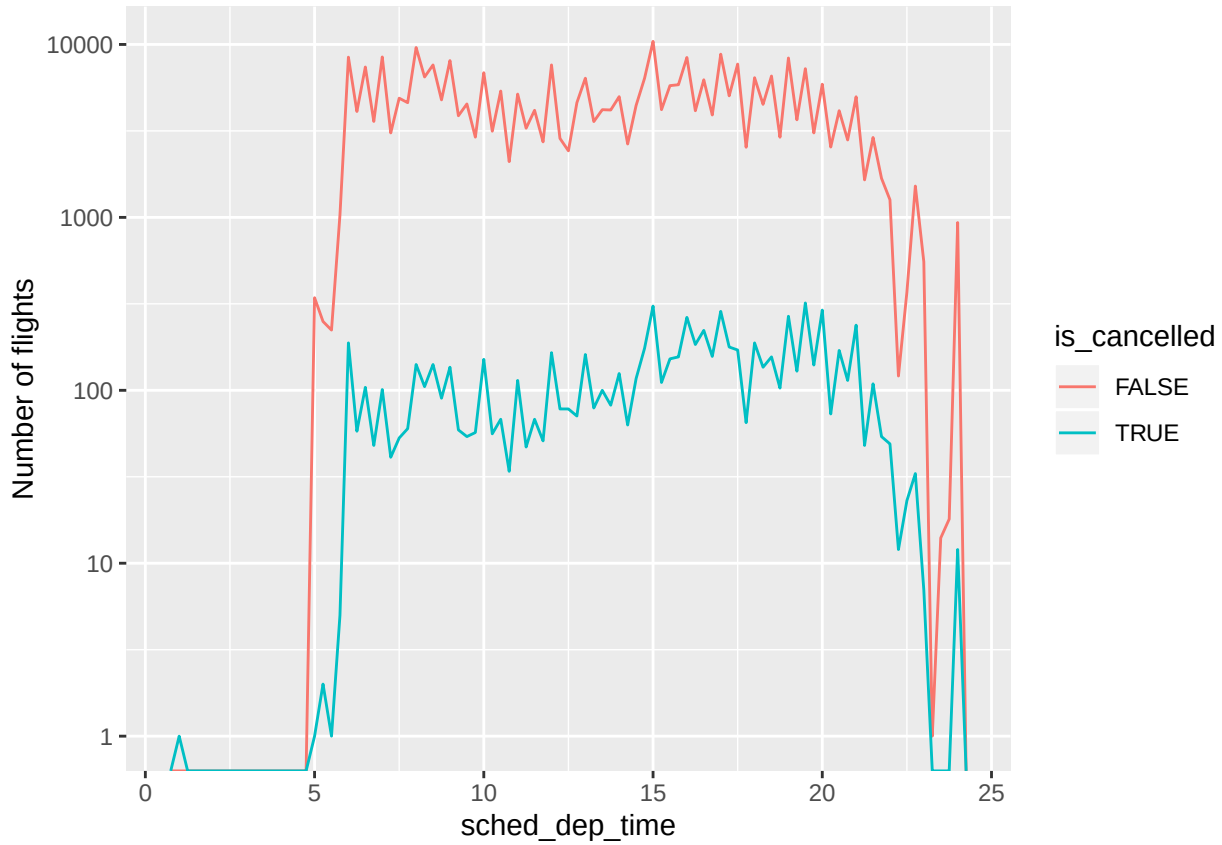
```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(nycflights13)
```

1. Use what you've learned to improve the visualization of the departure times of cancelled vs. non-cancelled flights. (NOTE: missing values in the dep_time variable indicate that the flight was cancelled)

To answer this question I will try to represent the number of cancelled and non-cancelled flights per 30 mins. Because the non-cancelled flights are orders of magnitude higher than the cancelled ones, I'll do a logarithmic scaling for the y-axis tto be able to represent the two on the same graph.

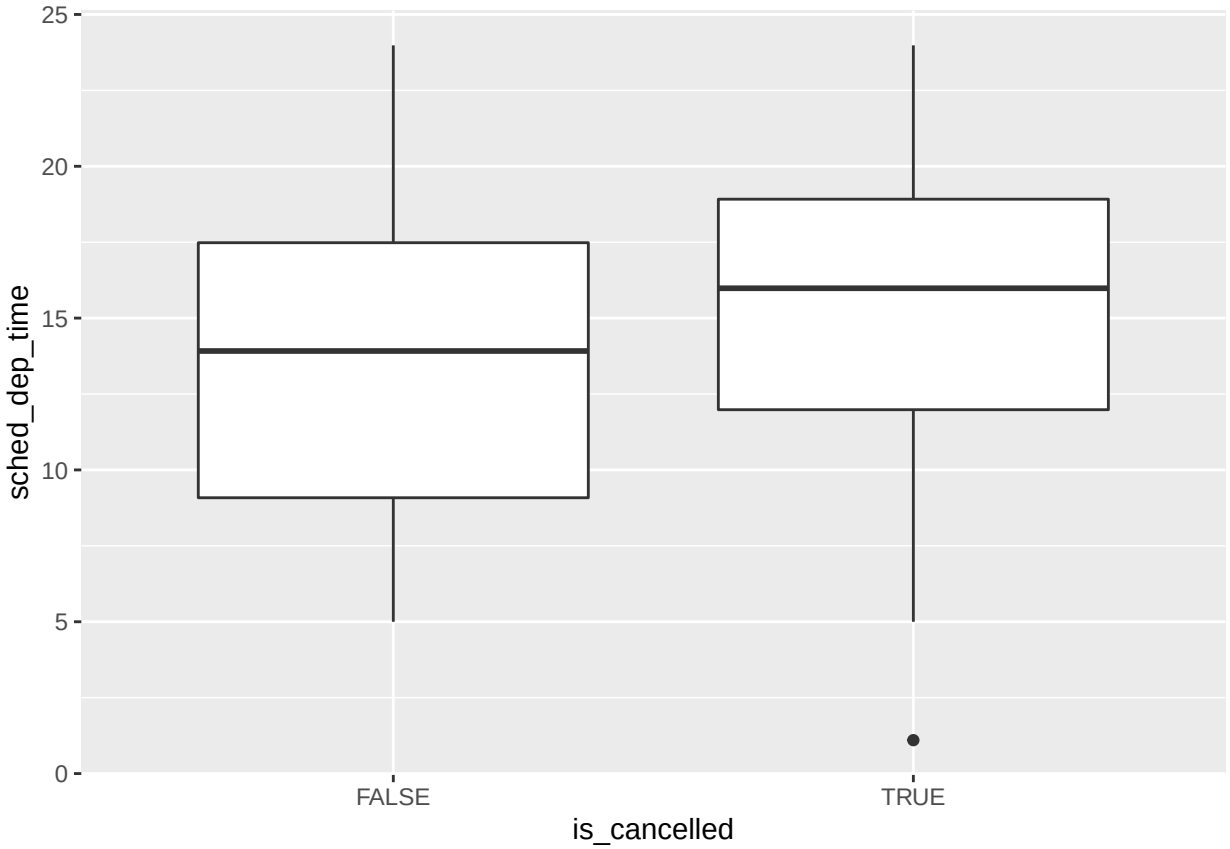
```
f <- flights %>%
  mutate(
    # creating a categorical variable for cancelled and non-cancelled flights
    is_cancelled = is.na(dep_time),
    # converting sched_dep_time to a continuous variable
    sched_hour = sched_dep_time %/% 100,
    sched_min = sched_dep_time %% 100,
    sched_dep_time = sched_hour + sched_min / 60
  )
f %>%
  ggplot(mapping = aes(sched_dep_time)) +
    geom_freqpoly(mapping = aes(colour = is_cancelled), binwidth = 1/4) +
    # logarithmic scale for y gives us a better visualization
    scale_y_log10("Number of flights")
```



We can clearly see the correlation between the number of cancelled and non-cancelled flights.

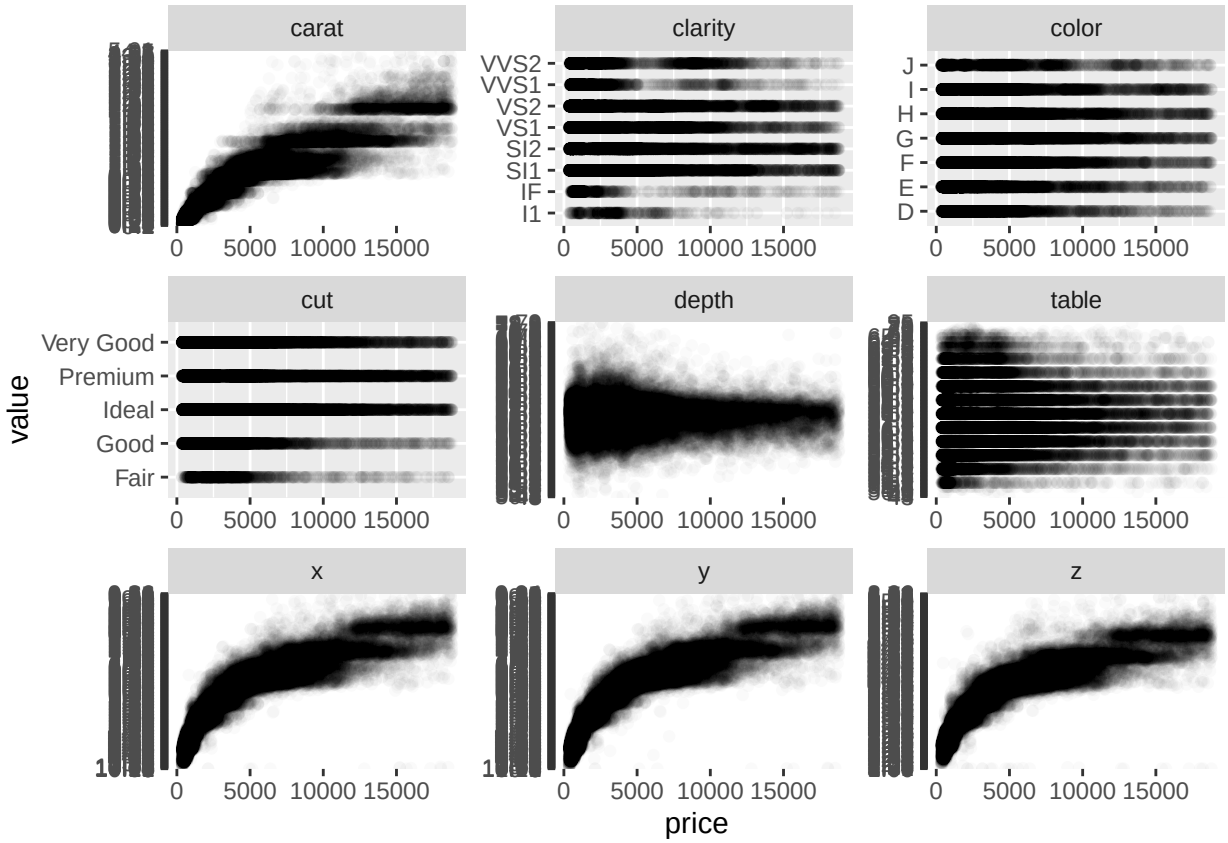
We can also visualize this as box plots :

```
f %>%
  ggplot(mapping = aes(x = is_cancelled, y = sched_dep_time)) +
  geom_boxplot()
```



2. What variable in the diamonds data set is most important for predicting the price of a diamond?

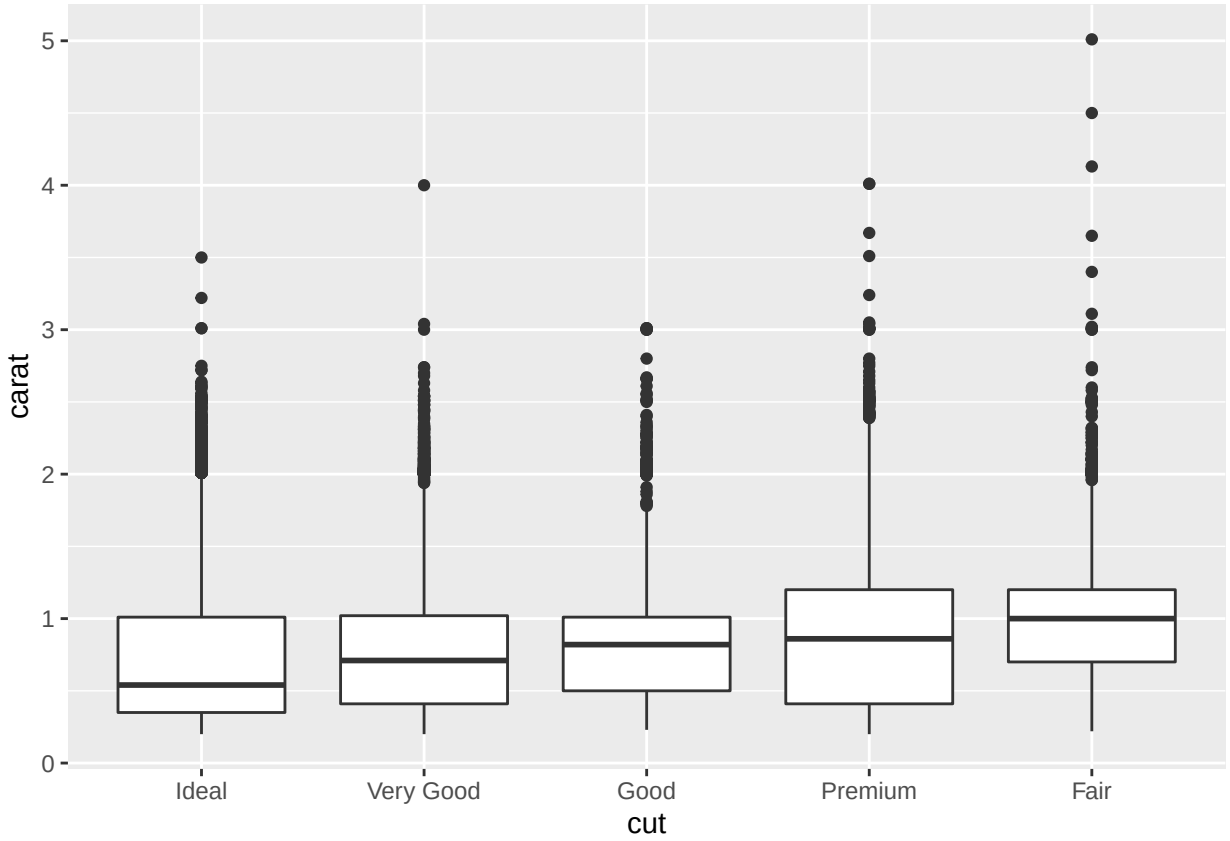
```
diamonds %>%
  gather(-price, key = "var", value = "value") %>%
  ggplot(aes(x = value, y = price)) +
    geom_point(alpha = 1/50) +
    facet_wrap(~ var, scales = "free") +
    coord_flip()
```



From the plots above we can conclude that carat and the diamond size are the two most important variables for predicting the price, and it seems that the co-variance is more important in the case of the variable carat, because the plots seems to become parallel to the x axis more quickly for the size variables than for the carat one.

How is that variable correlated with cut?

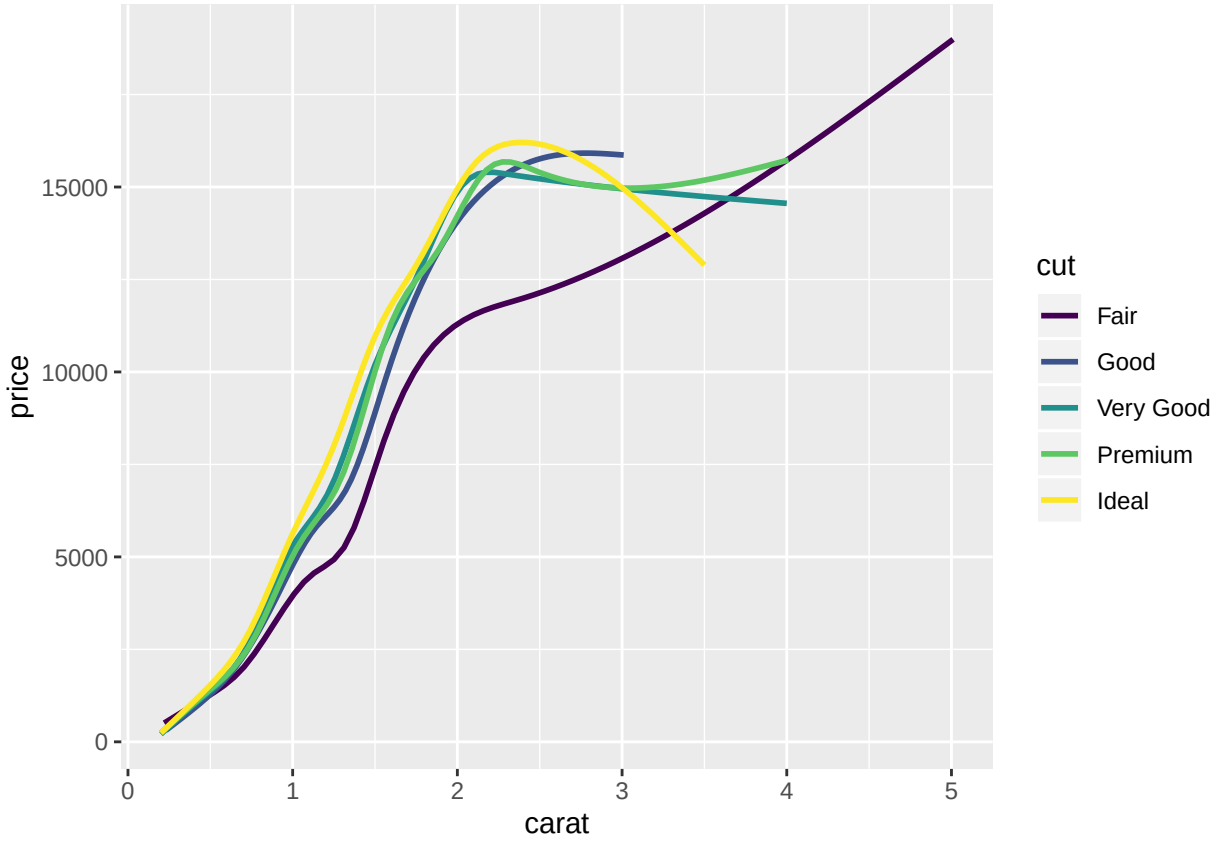
```
diamonds %>%
  ggplot(mapping = aes(x = reorder(cut, carat, FUN = median), y = carat))+
  geom_boxplot()+
  xlab('cut')
```



Why does the combination of those two relationships lead to lower quality diamonds being more expensive?

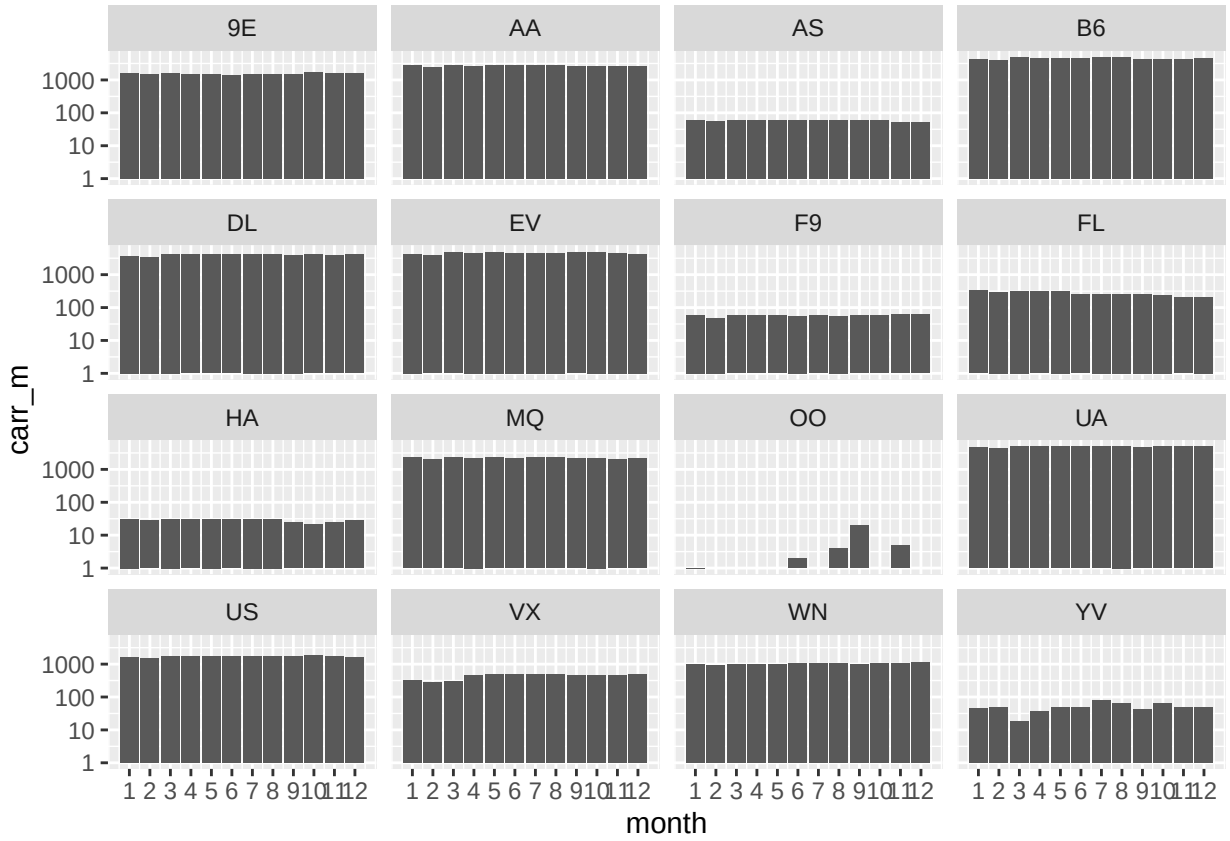
from the boxplot above, it seems that low quality diamonds (cut = Fair) have high carat value in average than other cuts, and that what leads have more expensive diamods of low cut quality. But, for the same carat value lower quality cuts are indeed cheaper.

```
diamonds %>%
  ggplot(mapping = aes(x = carat, y = price, color = cut)) +
  geom_smooth(se = FALSE)
```



3. Visualize the number of flights of each airline by month.

```
flights %>%
  group_by (carrier, month) %>%
  summarise (carr_m = n()) %>%
  ggplot() +
  geom_col(mapping = aes(x = month, y = carr_m)) +
  facet_wrap(~ carrier) +
  scale_x_continuous(breaks = 1:12) +
  scale_y_log10()
```

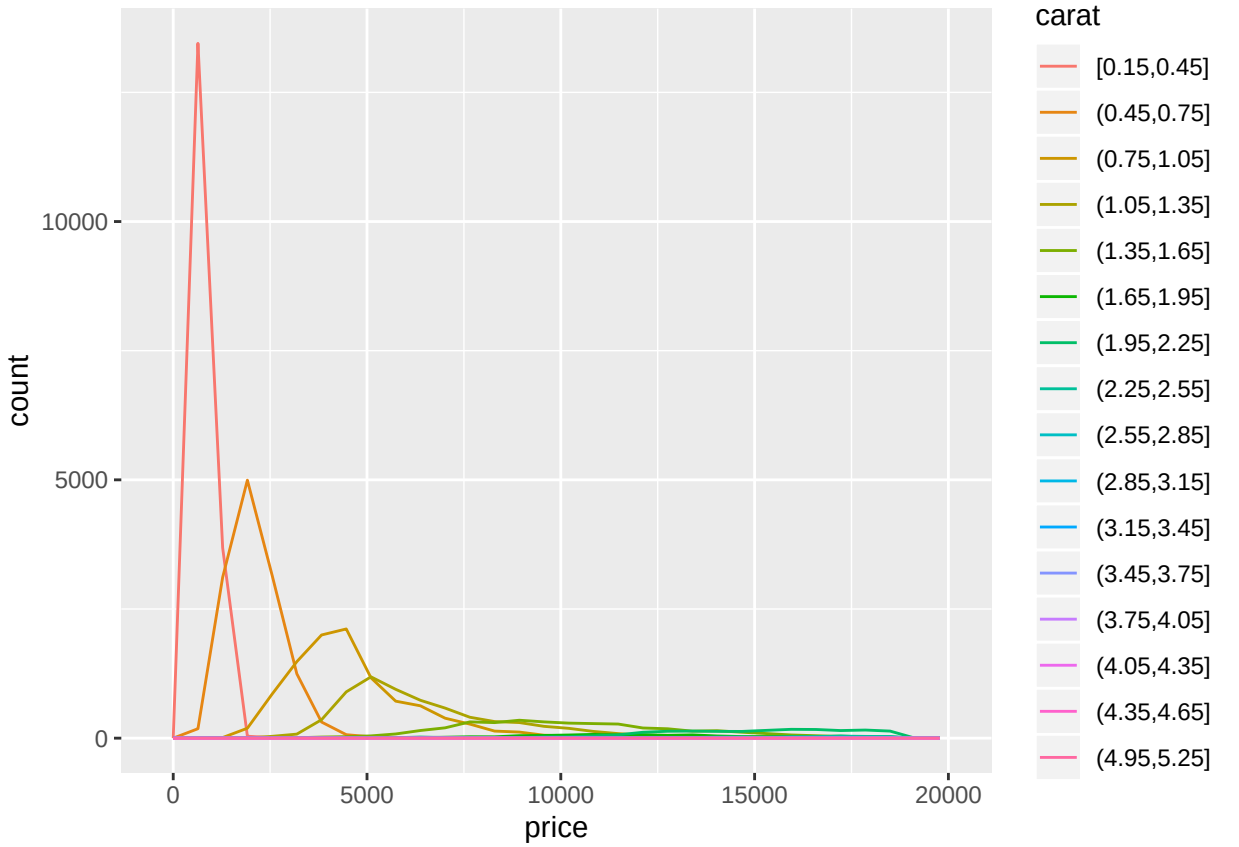


4. Instead of summarizing the conditional distribution with a boxplot, you could use a frequency polygon. What do you need to consider when using cut_width() vs cut_number()? How does that impact a visualization of the 2d distribution of carat

and price (diamonds dataset)?

In plotting a frequency polygon cut_number() and cut_width() could be used as aesthetics arguments to present conditional distributions (~ P(A/B)), the difference between the two is that cut_width() makes a distribution of A on equal intervals of width = 'width' of B, this not always the best option for visualization, especially when A is skewed to some values of B.

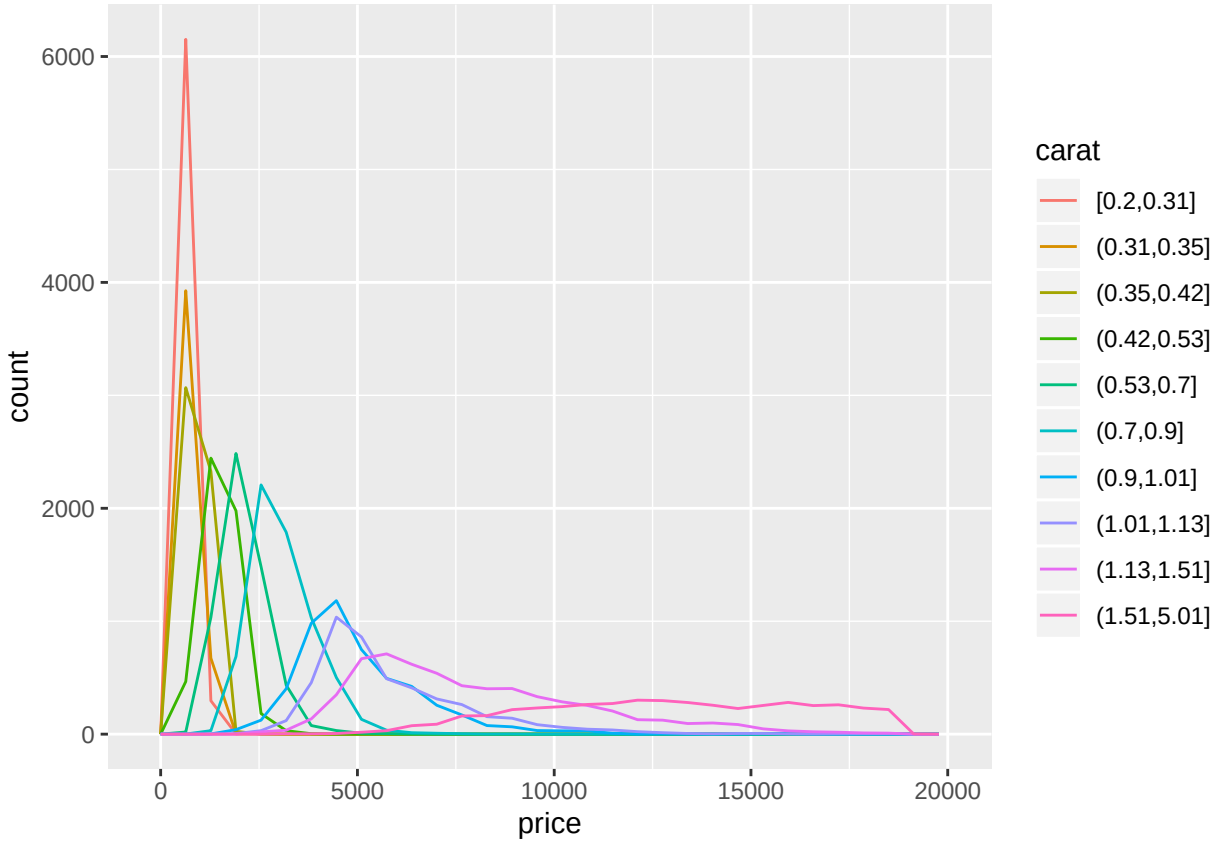
```
diamonds %>%
  ggplot(mapping = aes(x = price, color = cut_width(carat, 0.3))) +
  geom_freqpoly() +
  labs(color='carat')
```



We see here that there's a concentration of diamonds on the low price and low carat side, witch give an uneven distribution of the number of observations and consequently a bad representation.

On the other hand cut_number(A, n) makes n groups with (approximately) equal numbers of observation by chosing narrow intervals of B where A is dense, this results in a better 2d reprenstation.

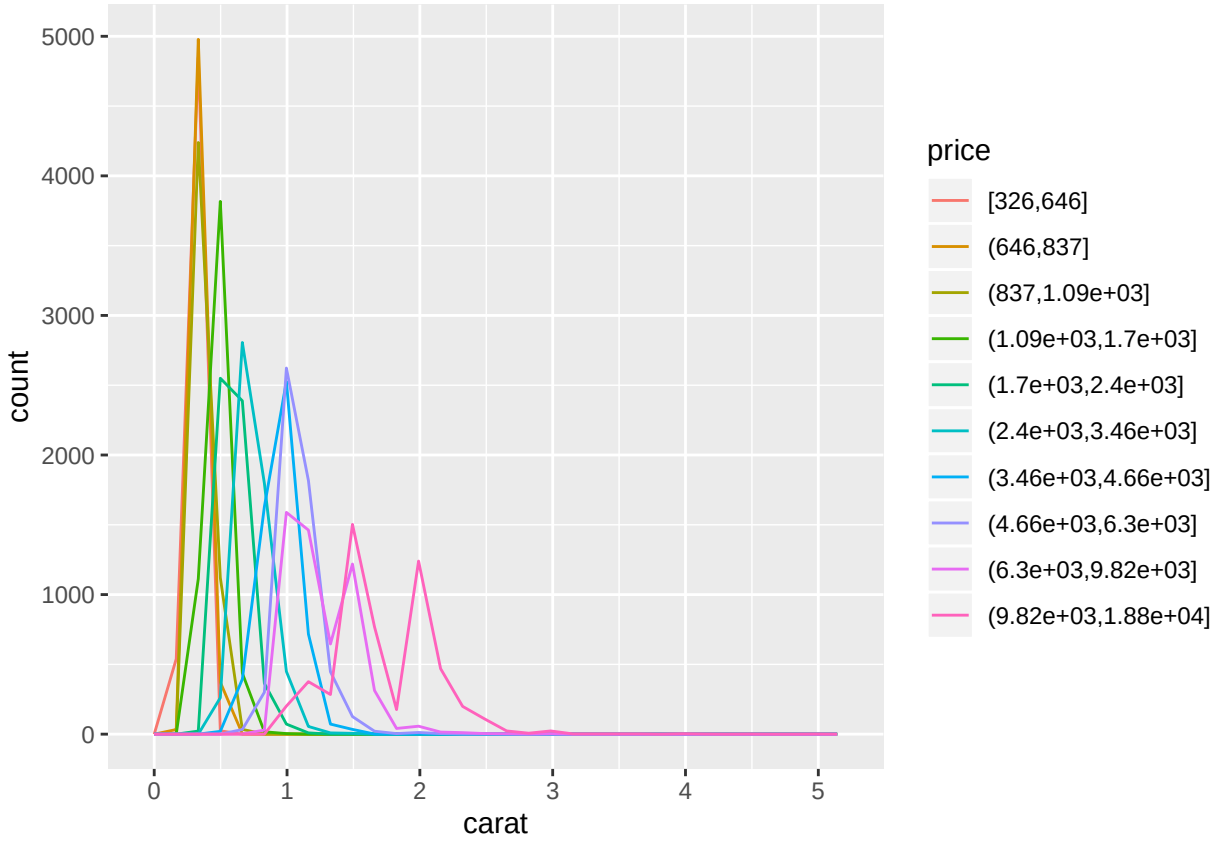
```
diamonds %>%
  ggplot(mapping = aes(x = price, color = cut_number(carat, 10))) +
  geom_freqpoly() +
  labs(color='carat')
```



5. Visualise the distribution of carat, partitioned by price on diamonds dataset .

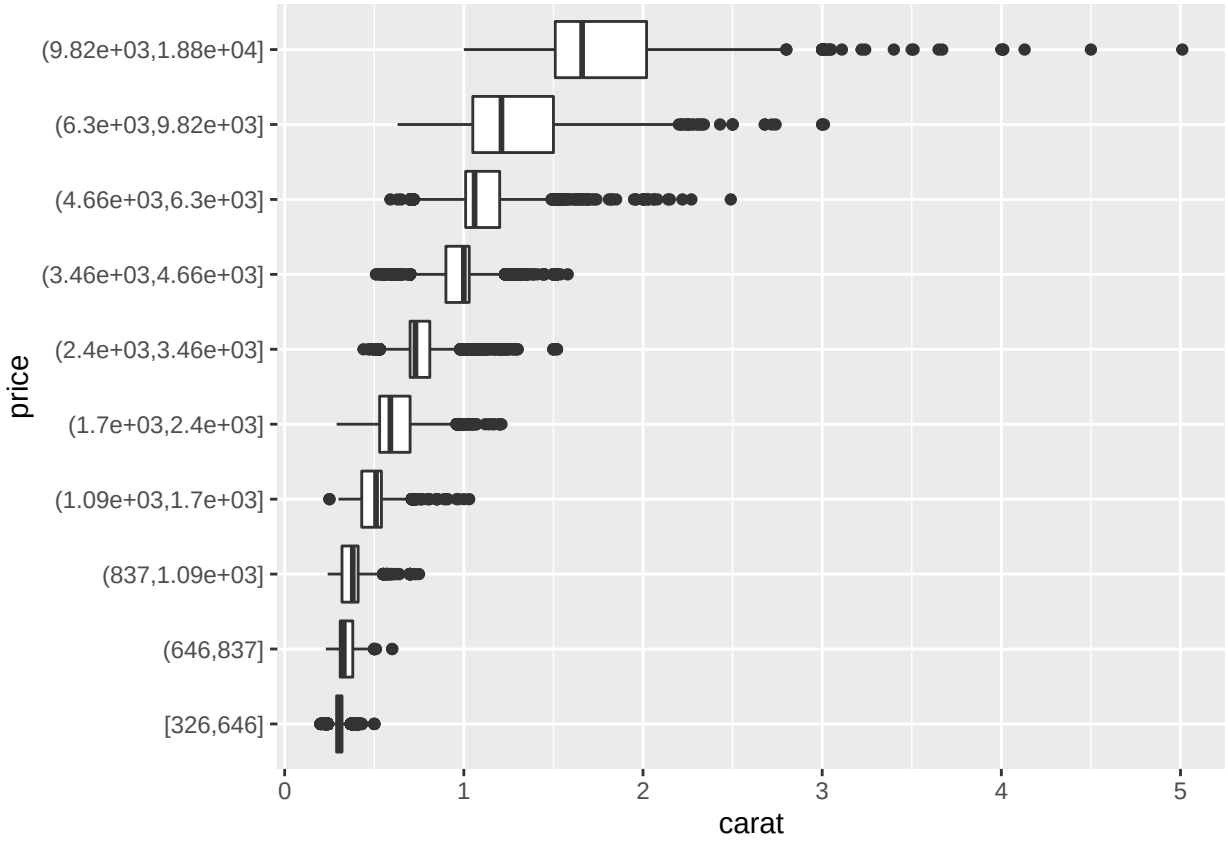
As a freqpoly:

```
diamonds %>%
  ggplot(mapping = aes(x = carat, color = cut_number(price, 10))) +
  geom_freqpoly() +
  labs(color='price')
```



As a boxplot:

```
diamonds %>%
  ggplot(mapping = aes(x = cut_number(price, 10), y = carat)) +
  geom_boxplot() +
  coord_flip() +
  xlab('price')
```



Session info

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_CA.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_CA.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_CA.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_CA.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] nycflights13_1.0.1 forcats_0.4.0      stringr_1.4.0
##  [4] dplyr_0.8.3        purrr_0.3.2        readr_1.3.1
##  [7] tidyr_1.0.0        tibble_2.1.3       ggplot2_3.2.1
```

```
## [10] tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.5  xfun_0.10         splines_3.6.1
##  [4] haven_2.1.1       lattice_0.20-38   colorspace_1.4-1
##  [7] vctrs_0.2.0       generics_0.0.2    viridisLite_0.3.0
## [10] htmltools_0.4.0   yaml_2.2.0        mgcv_1.8-29
## [13] rlang_0.4.0       pillar_1.4.2      glue_1.3.1
## [16] withr_2.1.2       modelr_0.1.5      readxl_1.3.1
## [19] lifecycle_0.1.0   munsell_0.5.0     gtable_0.3.0
## [22] cellranger_1.1.0  rvest_0.3.4       evaluate_0.14
## [25] labeling_0.3      knitr_1.25        broom_0.5.2
## [28] Rcpp_1.0.2        scales_1.0.0      backports_1.1.5
## [31] jsonlite_1.6      hms_0.5.1         digest_0.6.21
## [34] stringi_1.4.3     grid_3.6.1        cli_1.1.0
## [37] tools_3.6.1       magrittr_1.5      lazyeval_0.2.2
## [40] crayon_1.3.4      pkgconfig_2.0.3   zeallot_0.1.0
## [43] Matrix_1.2-17     ellipsis_0.3.0    xml2_1.2.2
## [46] lubridate_1.7.4   assertthat_0.2.1  rmarkdown_1.16
## [49] httr_1.4.1        rstudioapi_0.10   R6_2.4.0
## [52] nlme_3.1-141      compiler_3.6.1
```