HW03p

Adina Bechhofer

April 13, 2018

```
knitr::opts_chunk$set(error = TRUE) #this allows errors to be printed into the PDF
```

1. Load pacakge `ggplot2` below using `pacman`.

```
library(pacman)
pacman::p_load('ggplot2')
```

The dataset `diamonds` is in the namespace now as it was loaded with the `ggplot2` package. Run the following code and write about the dataset below.

```
data(diamonds)
?diamonds
str(diamonds)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    53940 obs. of  10 variables:
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

```
diamonds$cut = factor(as.character(diamonds$cut))
diamonds$color = factor(as.character(diamonds$color))
diamonds$clarity = factor(as.character(diamonds$clarity))
```

What is n, p, what do the features mean, what is the most likely response metric and why?

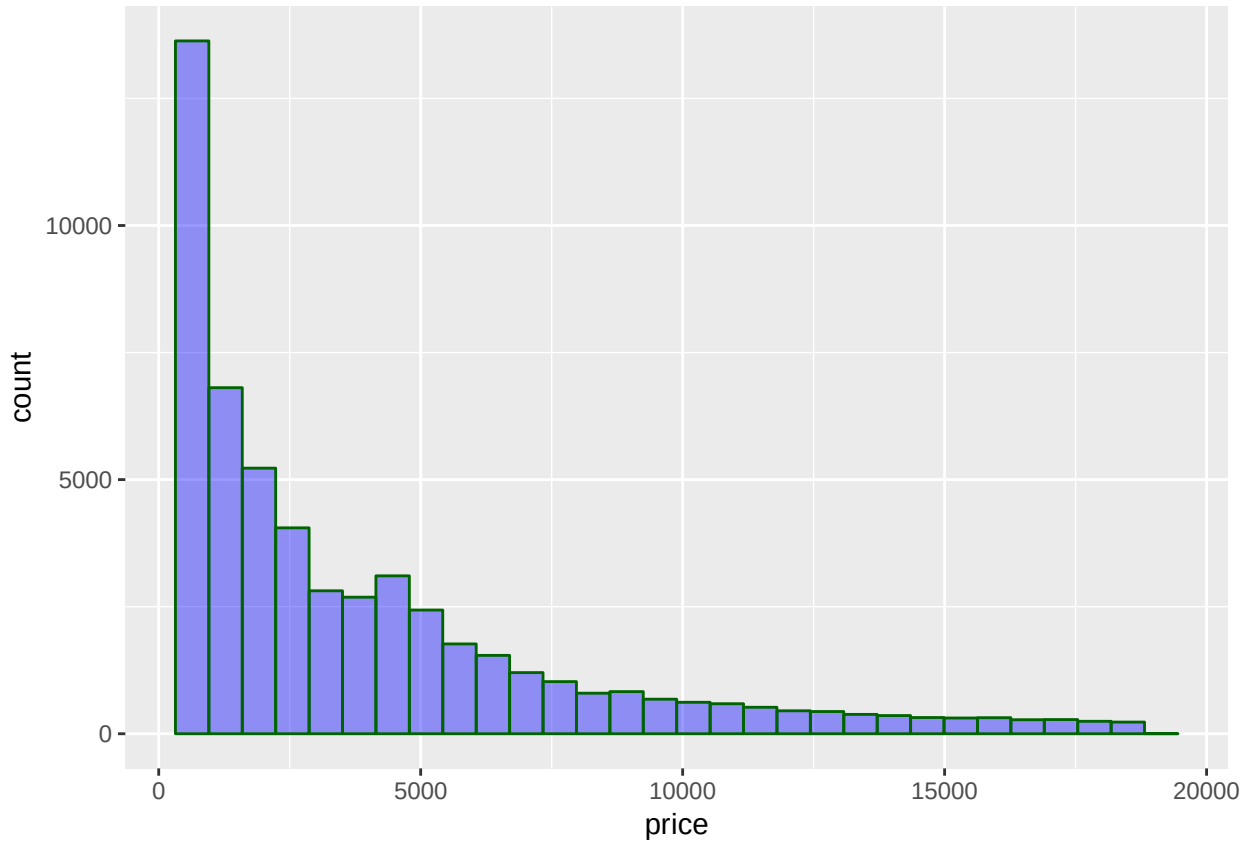
n = 53940, p= 9 Carat (size), cut, color, and clarity are thought to be the 4 factors that determine the price of a diamond. x, y, z, are the length width, and depth respectively. Depth and table are some other metrics of size. The responce, price, is in US dollars. It makes sense to predict price based on features.

Regardless of what you wrote above, the variable `price` will be the response variable going forward.

Use `ggplot` to look at the univariate distributions of *all* predictors. Make sure you handle categorical predictors differently from continuous predictors.

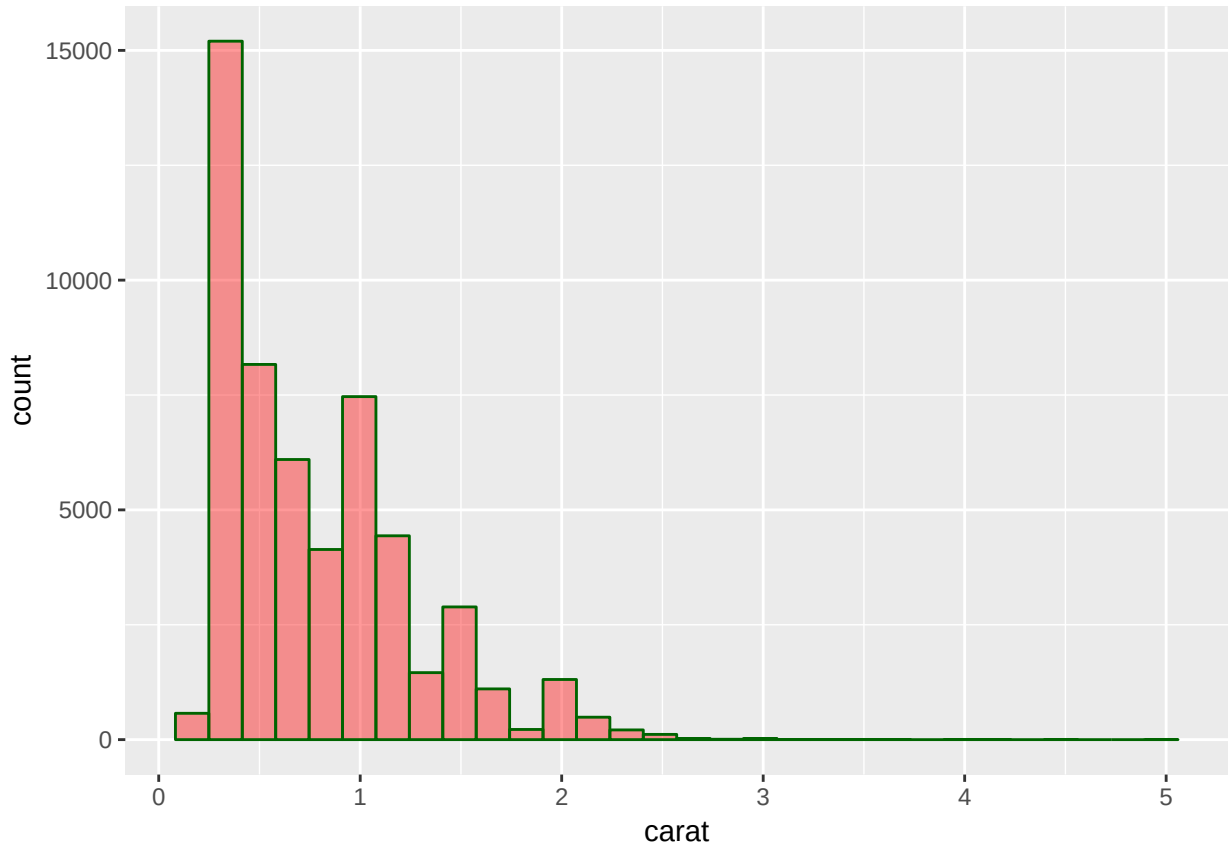
```
#base = ggplot(diamonds, aes(x = carat, y = price))
#base + geom_point(aes(col = color, shape = cut, alpha = clarity)) + scale_color_brewer(type = "div")

ggplot(diamonds, aes(price)) + geom_histogram( col = "darkgreen", fill = "blue", alpha = 0.4)
```

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



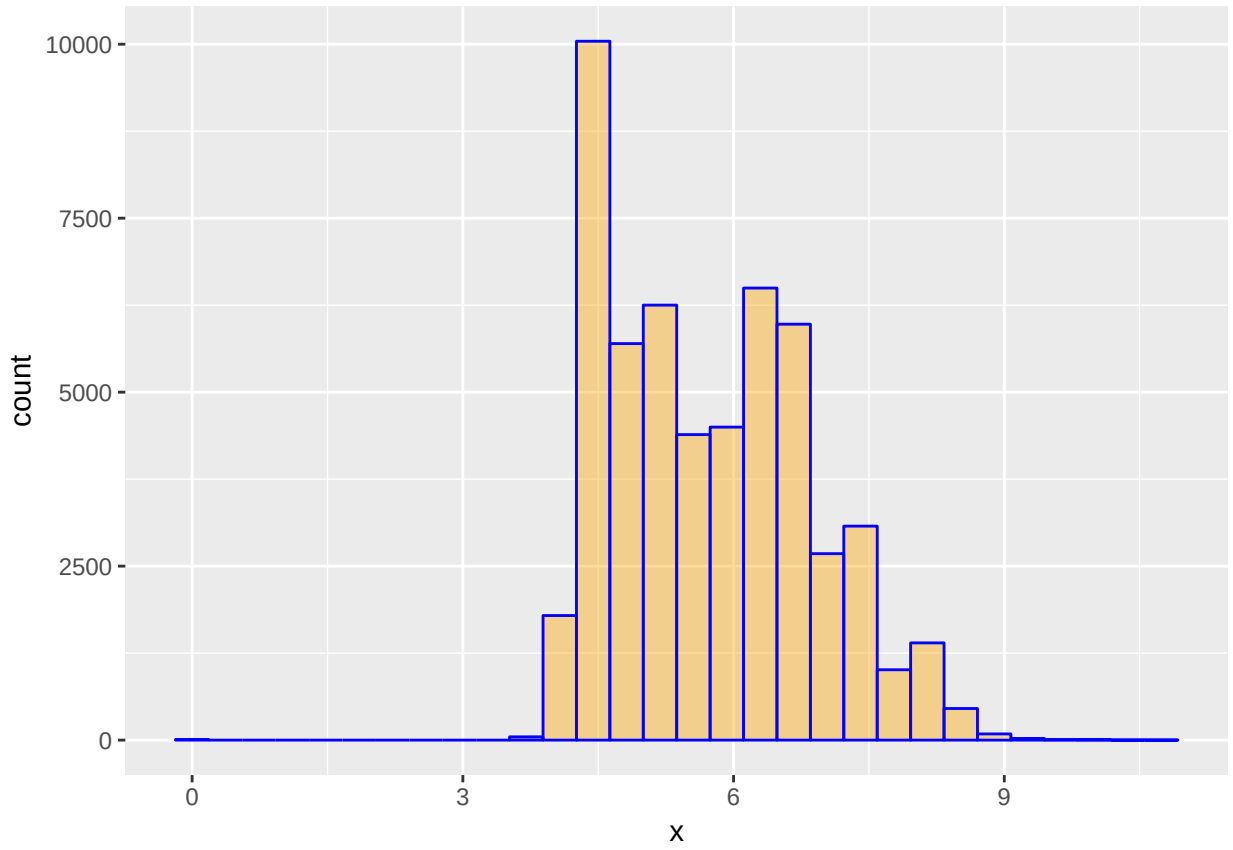

```
ggplot(diamonds, aes(carat)) + geom_histogram(col = "darkgreen", fill = "red", alpha = 0.4)
```

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



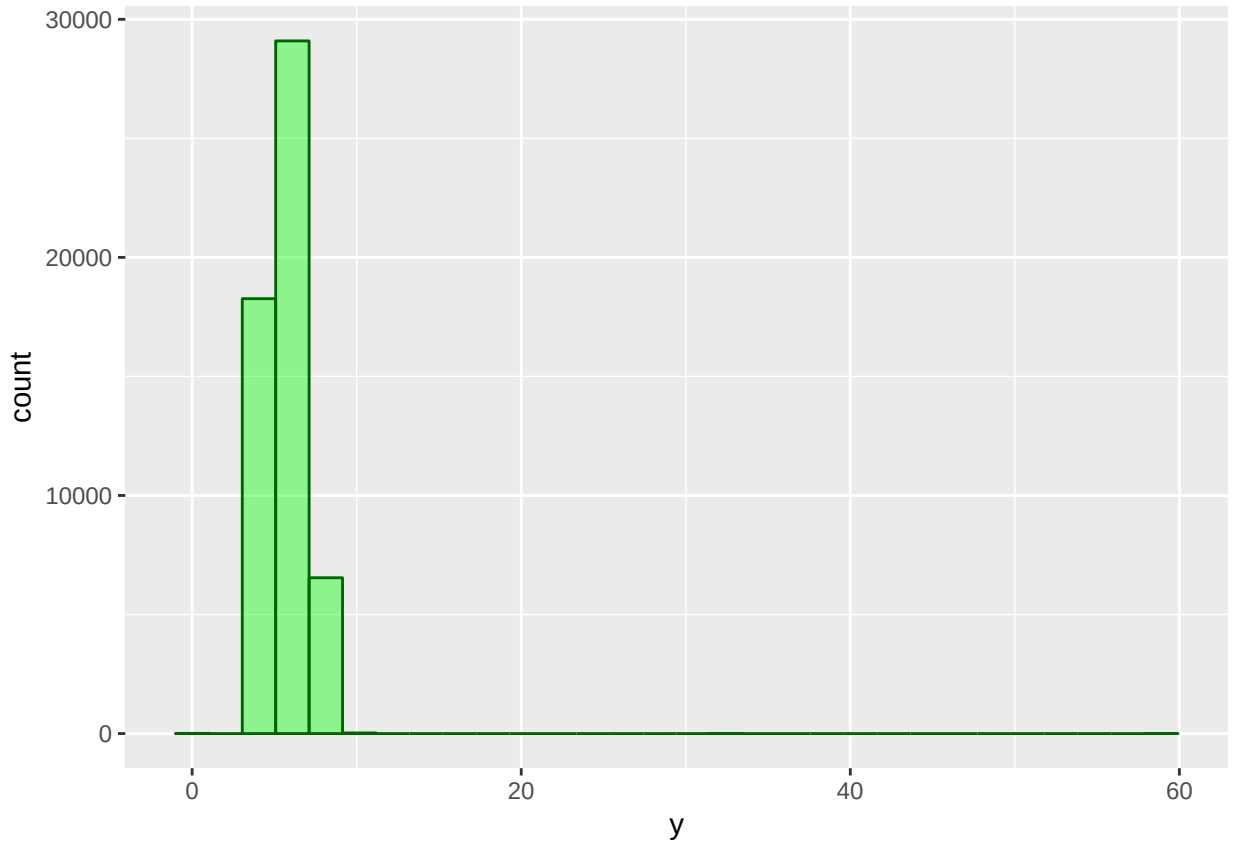

```
ggplot(diamonds, aes(x)) + geom_histogram( col = "blue", fill = "orange", alpha = 0.4)
```

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



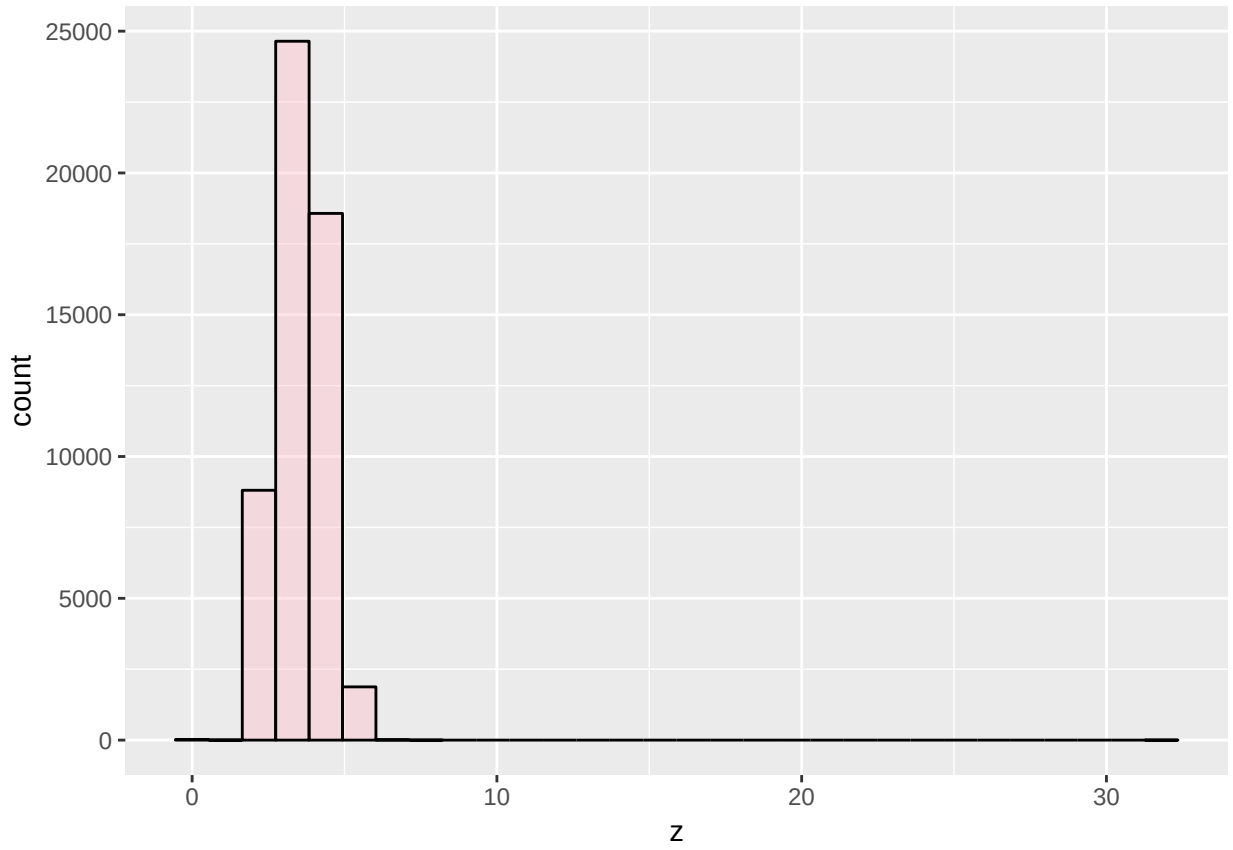

```
ggplot(diamonds, aes(y)) + geom_histogram(col = "darkgreen", fill = "green", alpha = 0.4)
```

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



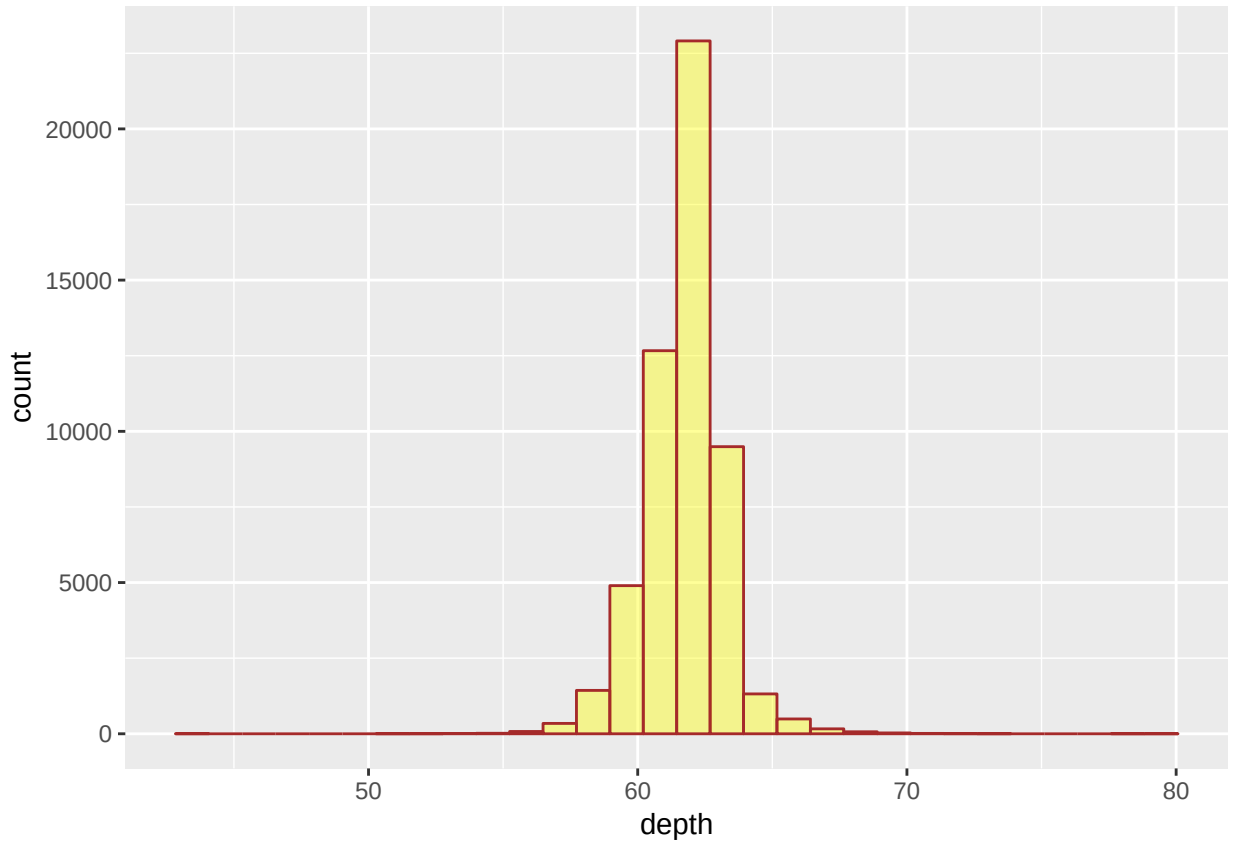

```
ggplot(diamonds, aes(z)) + geom_histogram(col = "black", fill = "pink", alpha = 0.4)
```

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



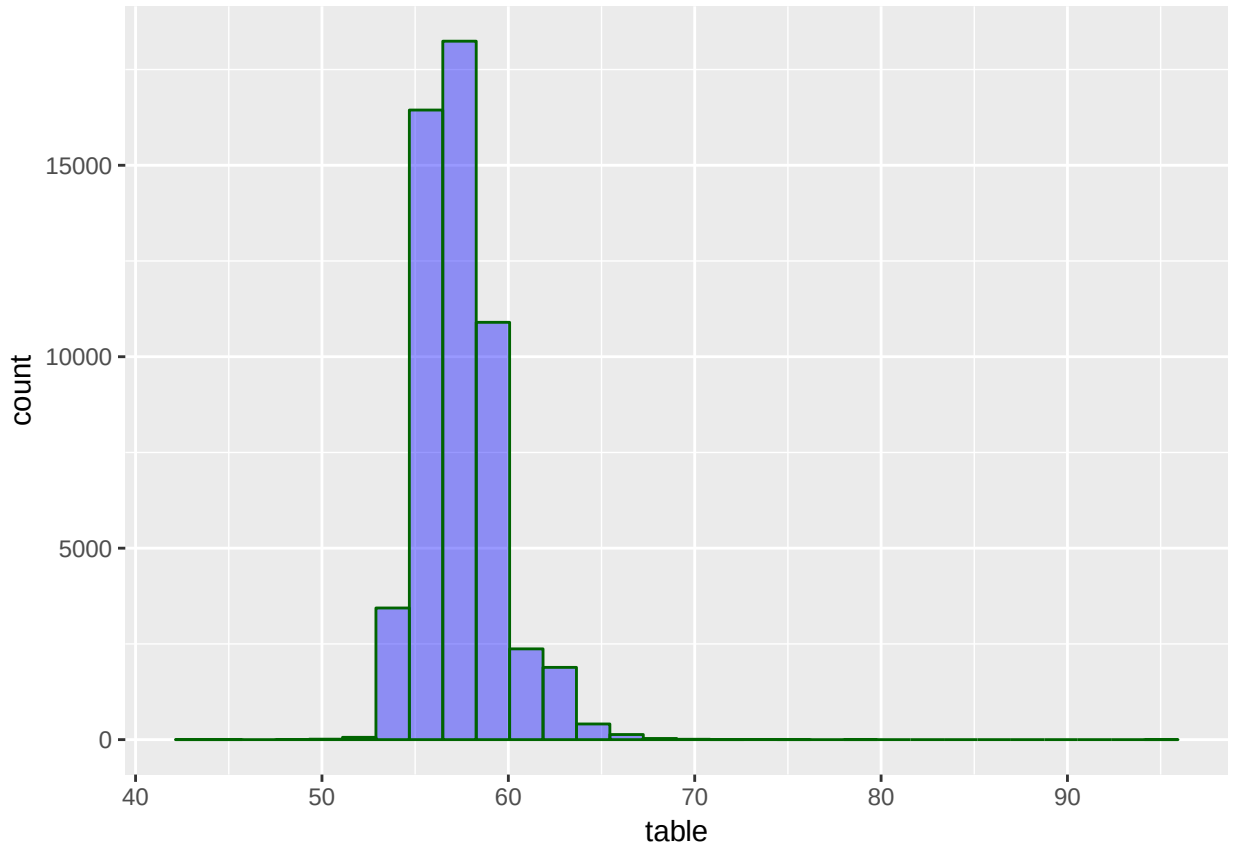

```
ggplot(diamonds, aes(depth)) + geom_histogram(col = "brown", fill = "yellow", alpha = 0.4)
```

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

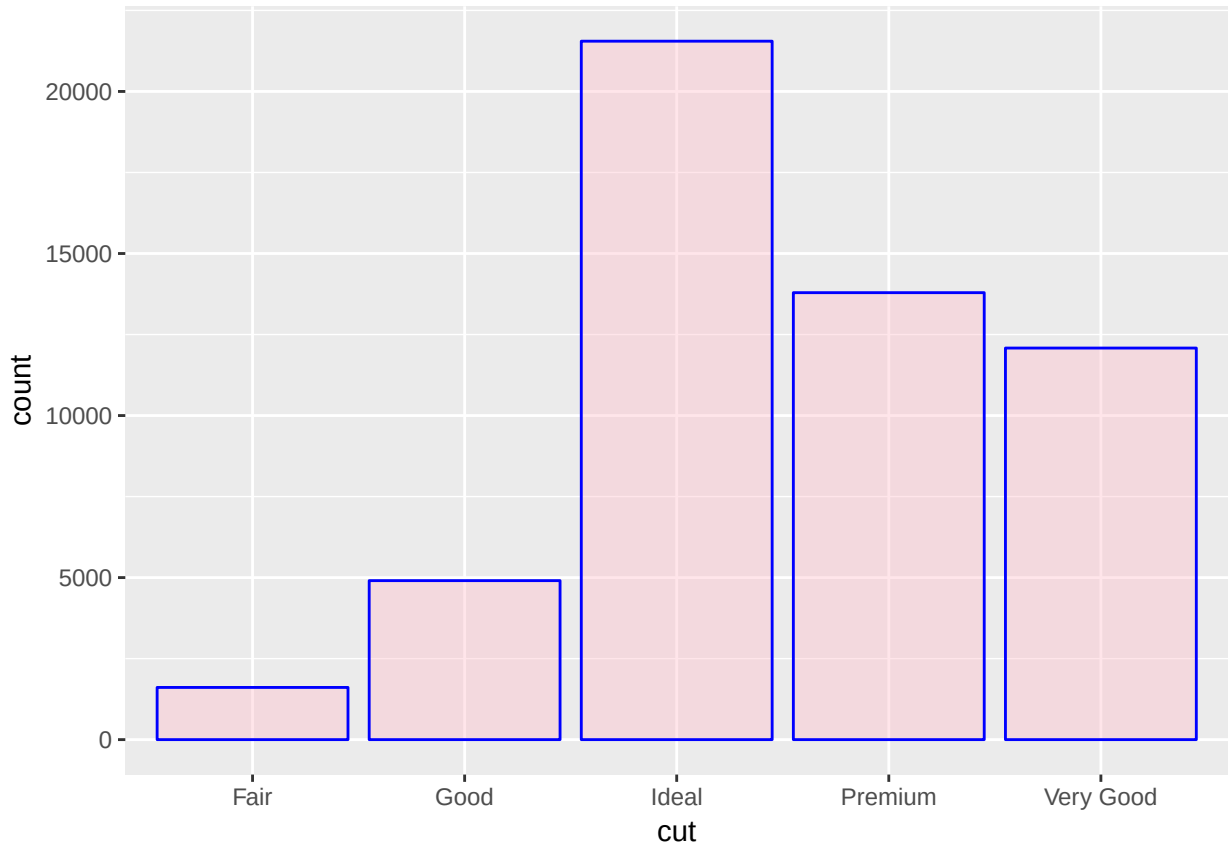



```
ggplot(diamonds, aes(table)) + geom_histogram(col = "darkgreen", fill = "blue", alpha = 0.4)
```

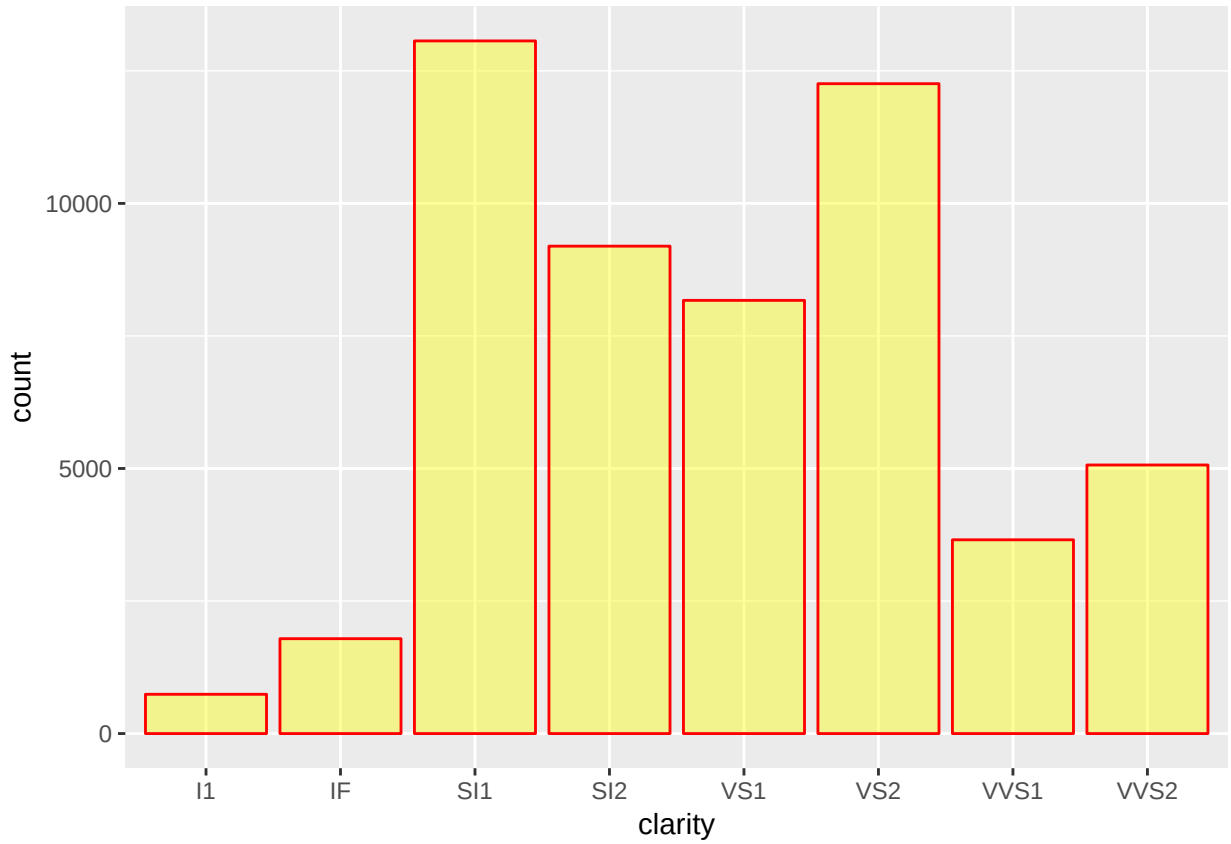
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



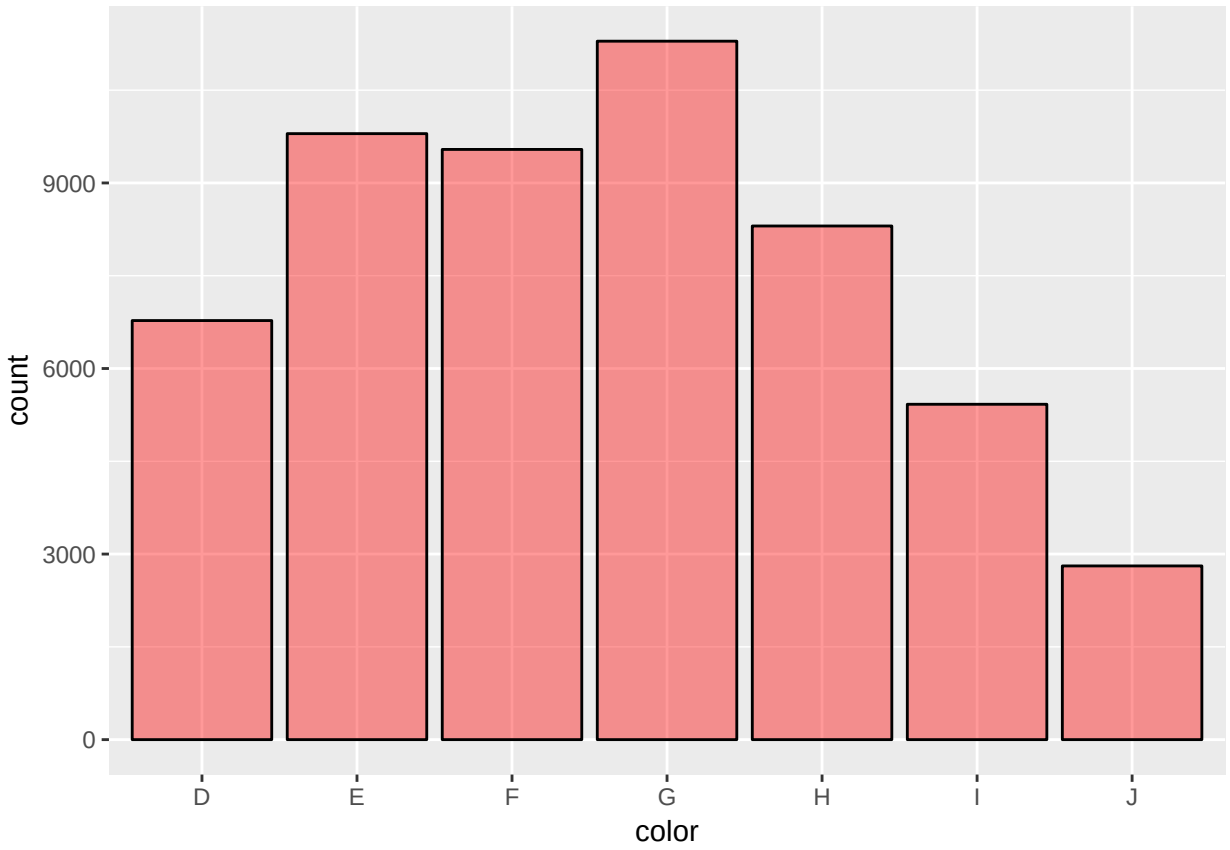

```
ggplot(diamonds, aes(cut)) + geom_bar(col= "blue", fill= "pink", alpha = 0.4)
```




```
ggplot(diamonds, aes(clarity)) + geom_bar(col= "red", fill= "yellow", alpha = 0.4)
```

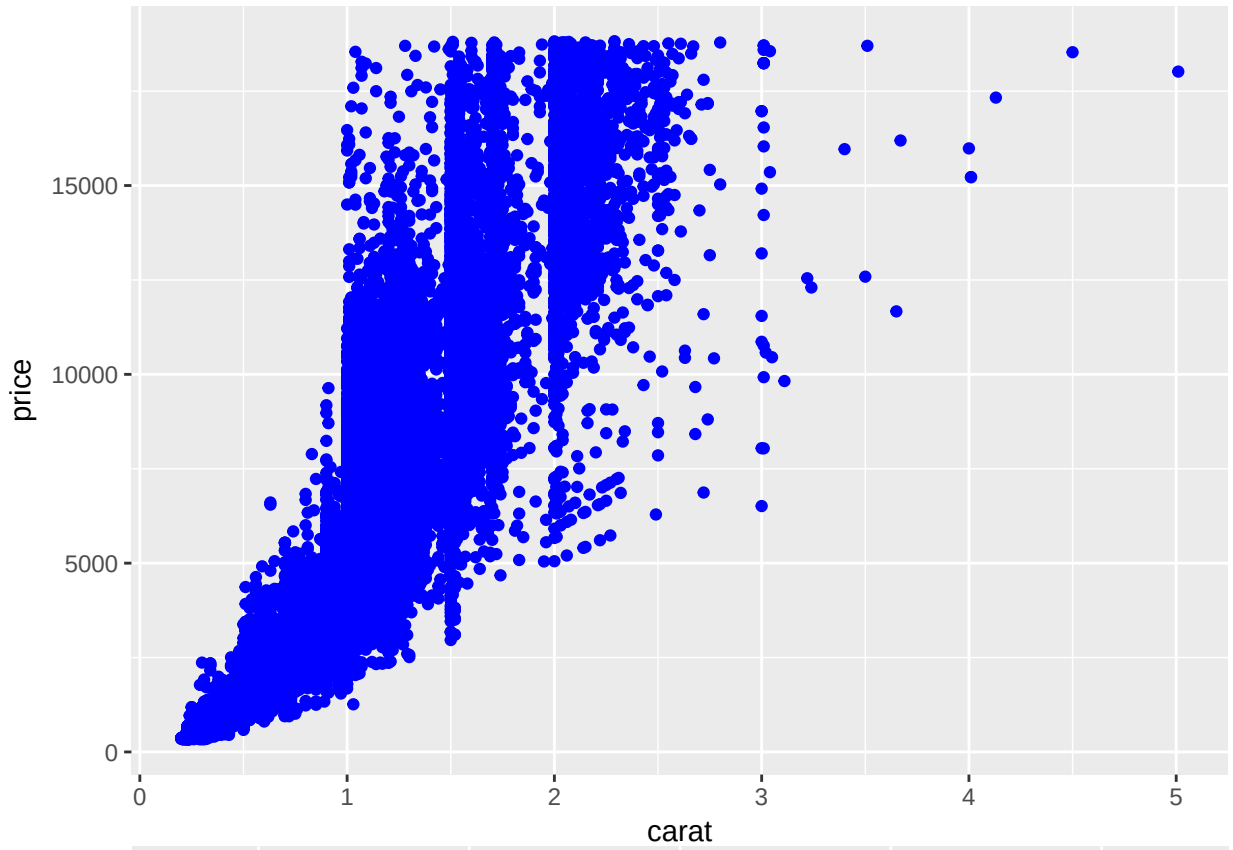


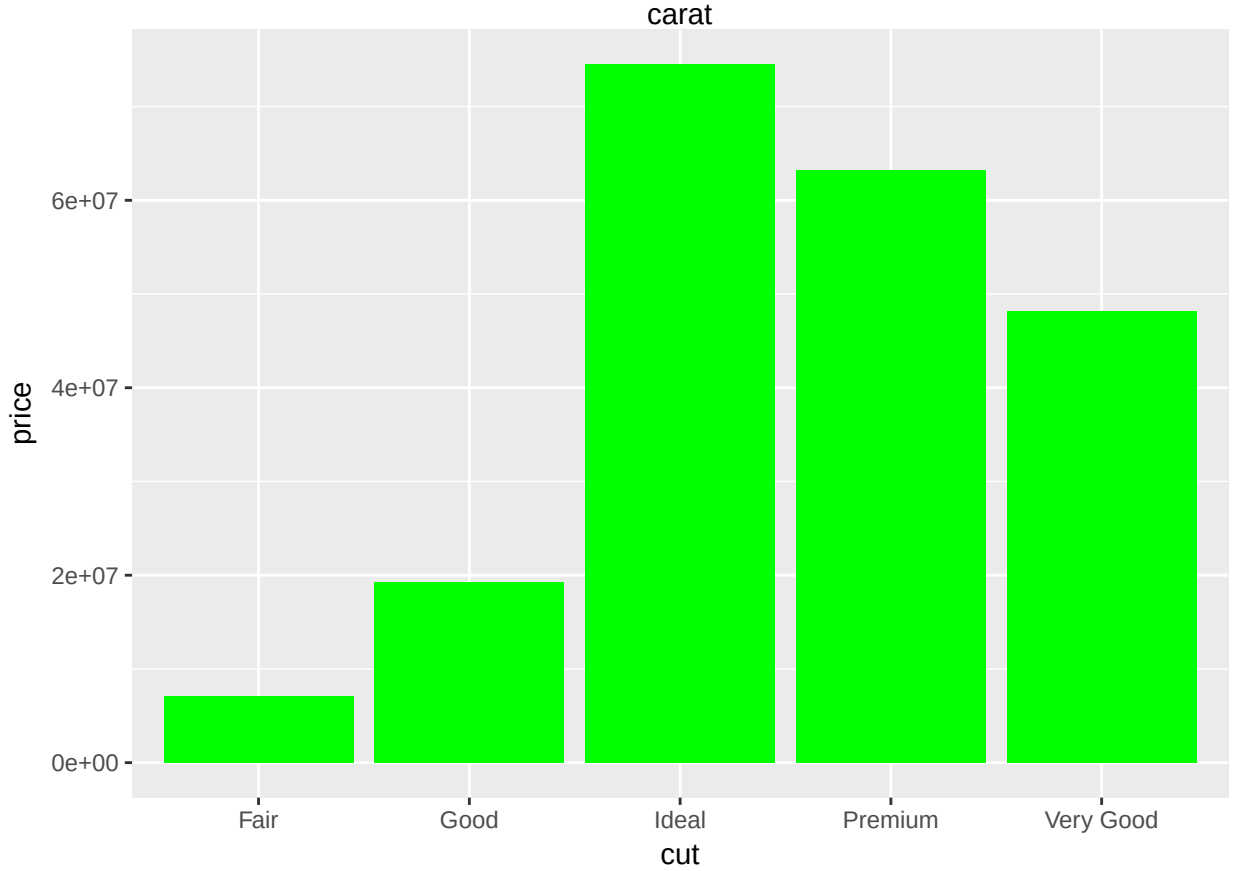

```
ggplot(diamonds, aes(color)) + geom_bar(col= "black", fill= "red", alpha = 0.4)
```

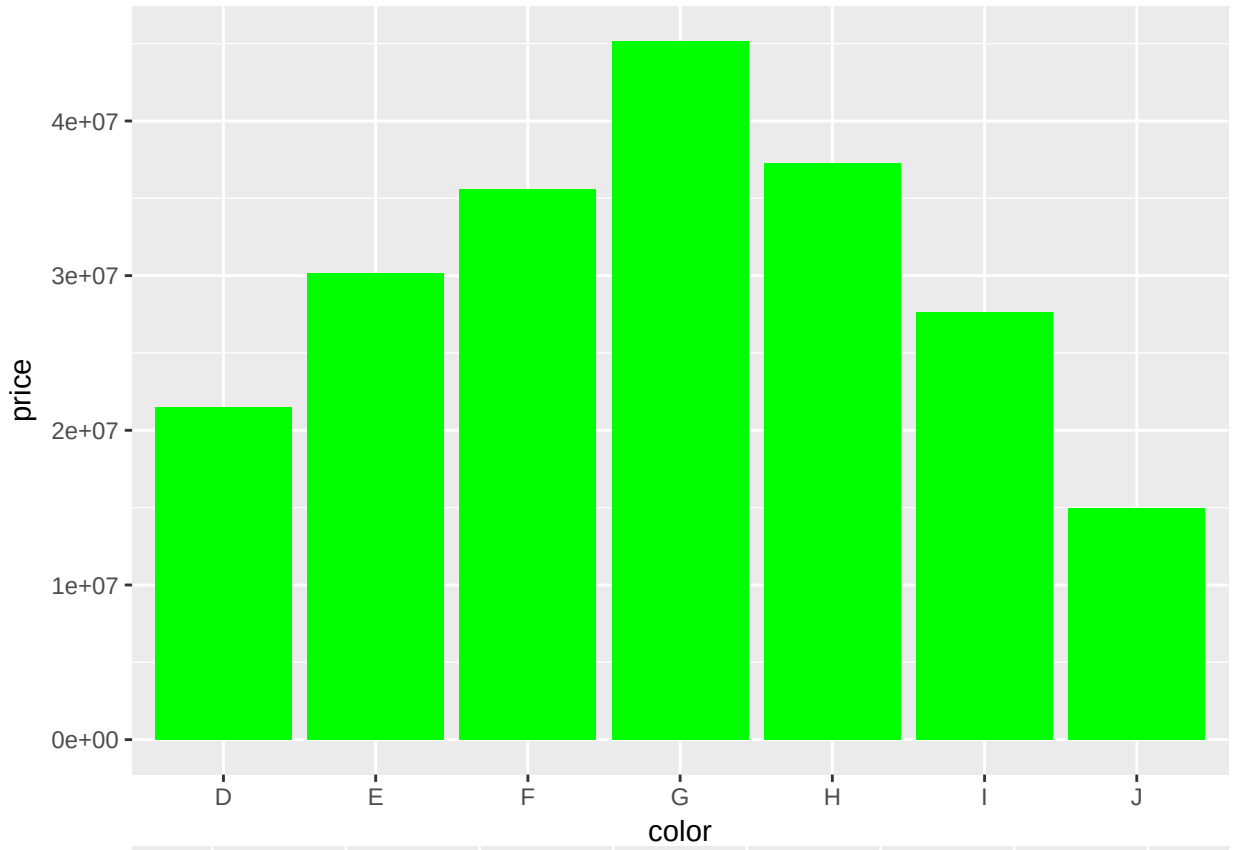


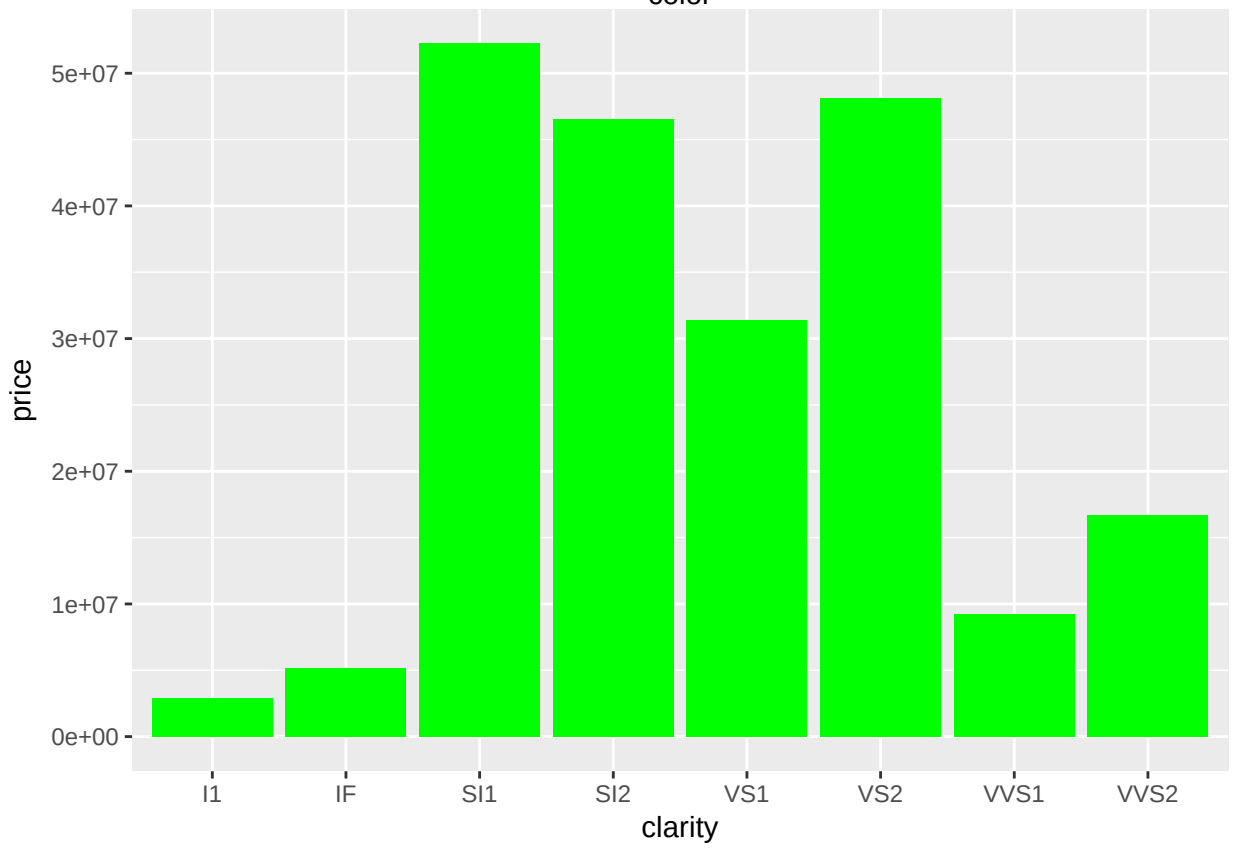
Use `ggplot` to look at the bivariate distributions of the response versus *all* predictors. Make sure you handle categorical predictors differently from continuous predictors. This time employ a for loop when an logic that handles the predictor type.

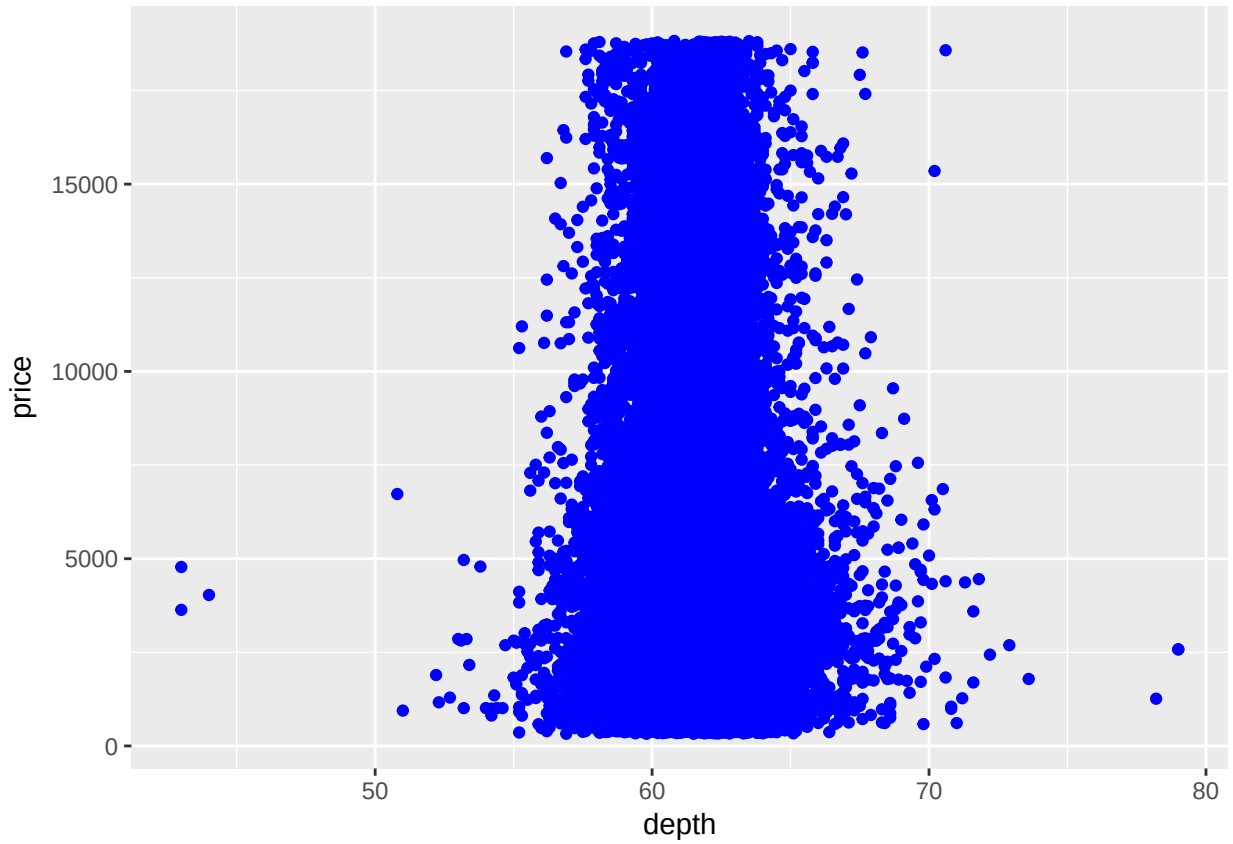
```
for (i in colnames(diamonds)){
  x2 = as.vector(as.matrix(diamonds[,i]))
  if (class(x2[[1]]) == "numeric"){
    print(ggplot(diamonds, aes(x = x2, y= price)) +geom_point(col= "blue") + xlab(i))

    } else { print(ggplot(diamonds, aes(x= x2, y = price))+ geom_bar(stat = "identity", fill= "green") +
  }
}
```

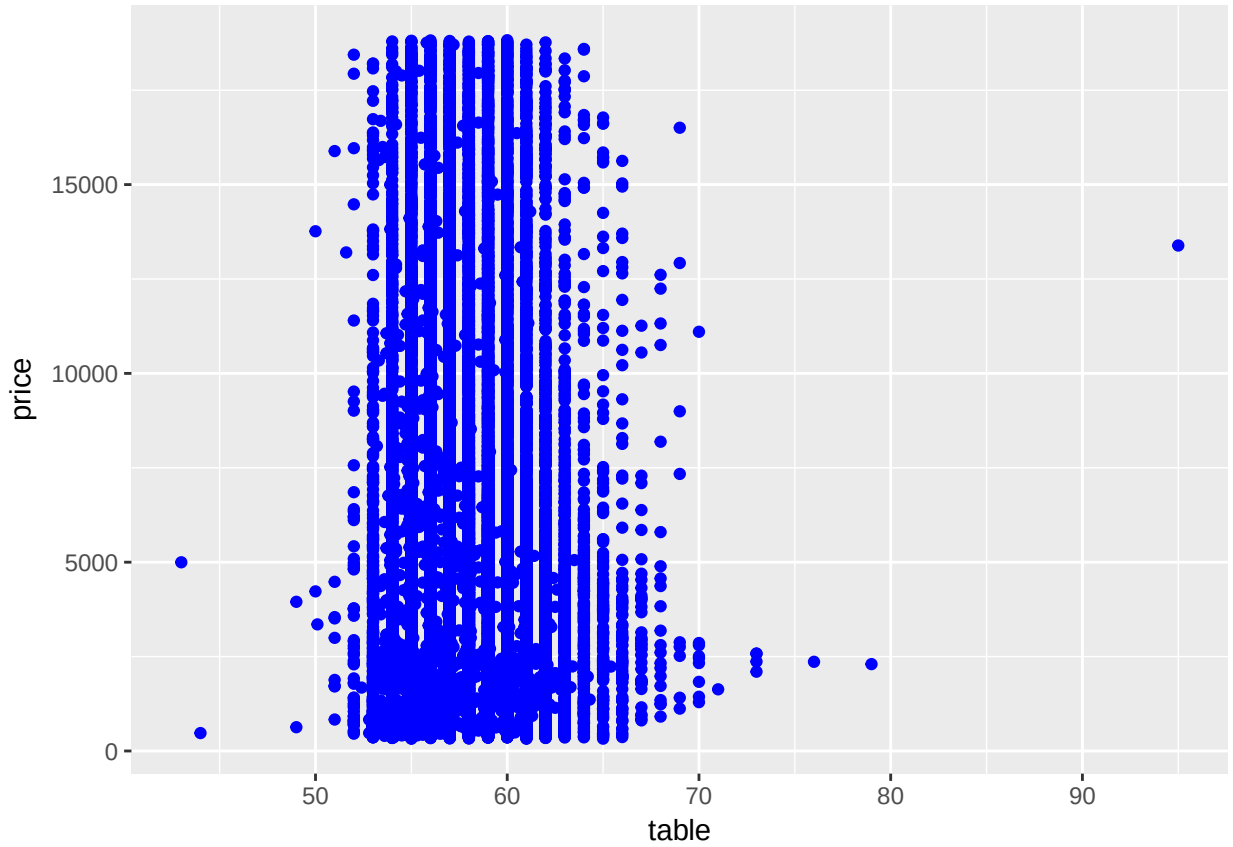

price
15000
10000
5000
0
0
1
2
3
4
5
carat

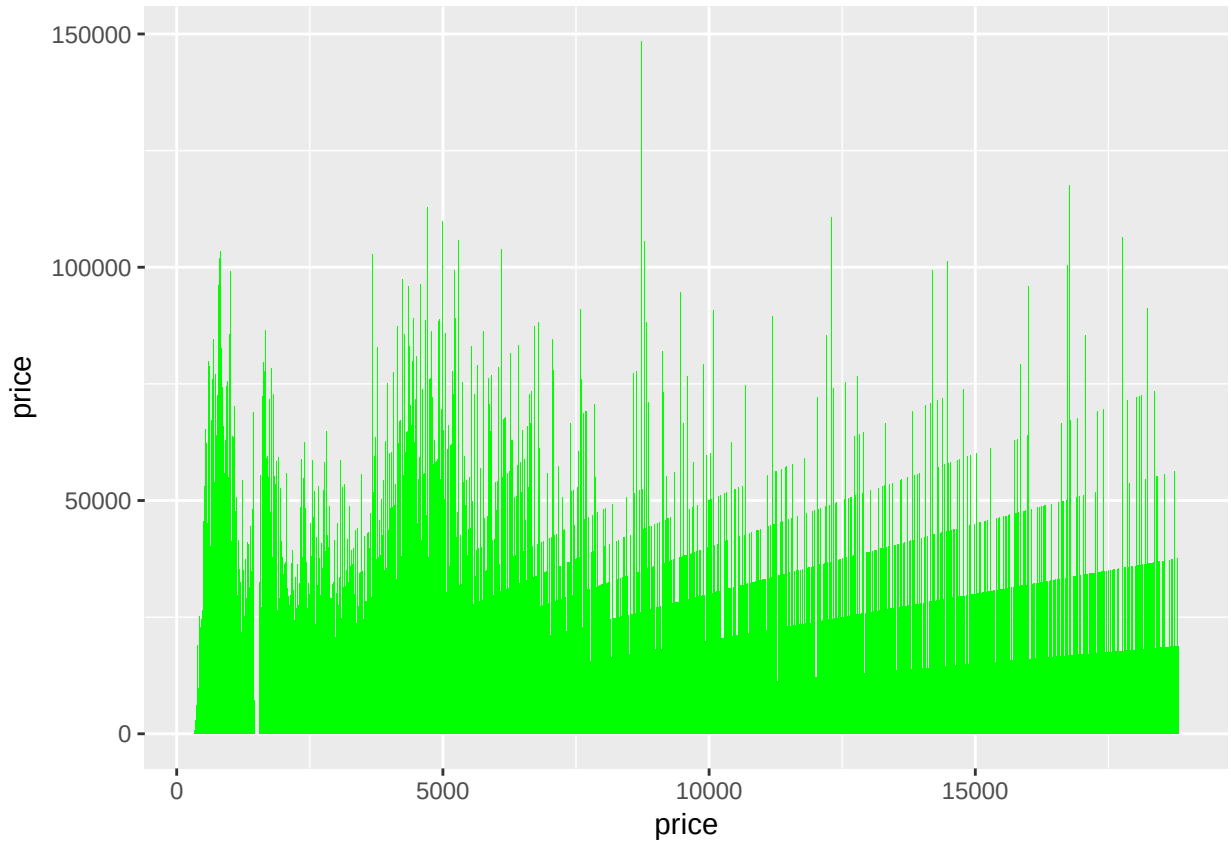

price
6e+07
4e+07
2e+07
0e+00
Fair
Good
Ideal
Premium
Very Good
cut

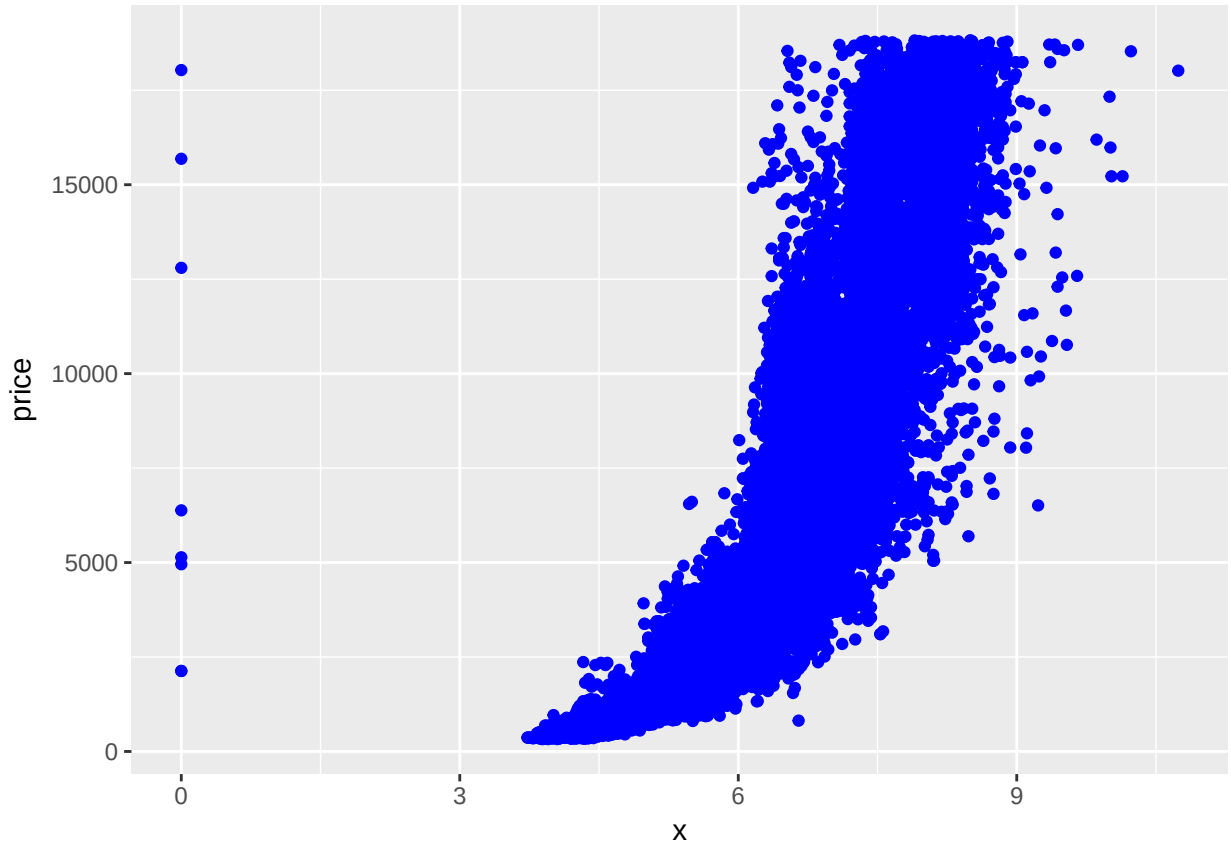

4e+07
3e+07
2e+07
1e+07
0e+00
price
D
E
F
G
H
I
J
color

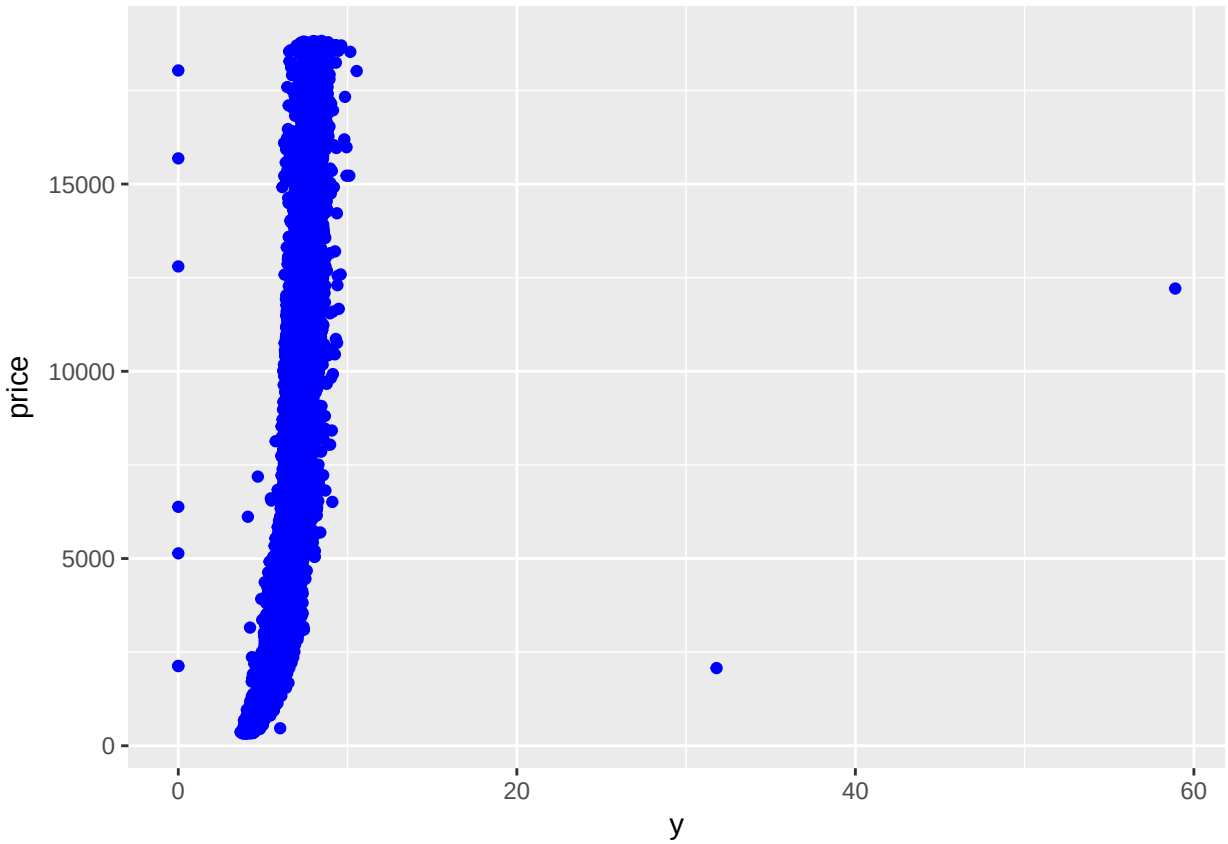

5e+07
4e+07
3e+07
2e+07
1e+07
0e+00
price
I1
IF
SI1
SI2
VS1
VS2
VVS1
VVS2
clarity

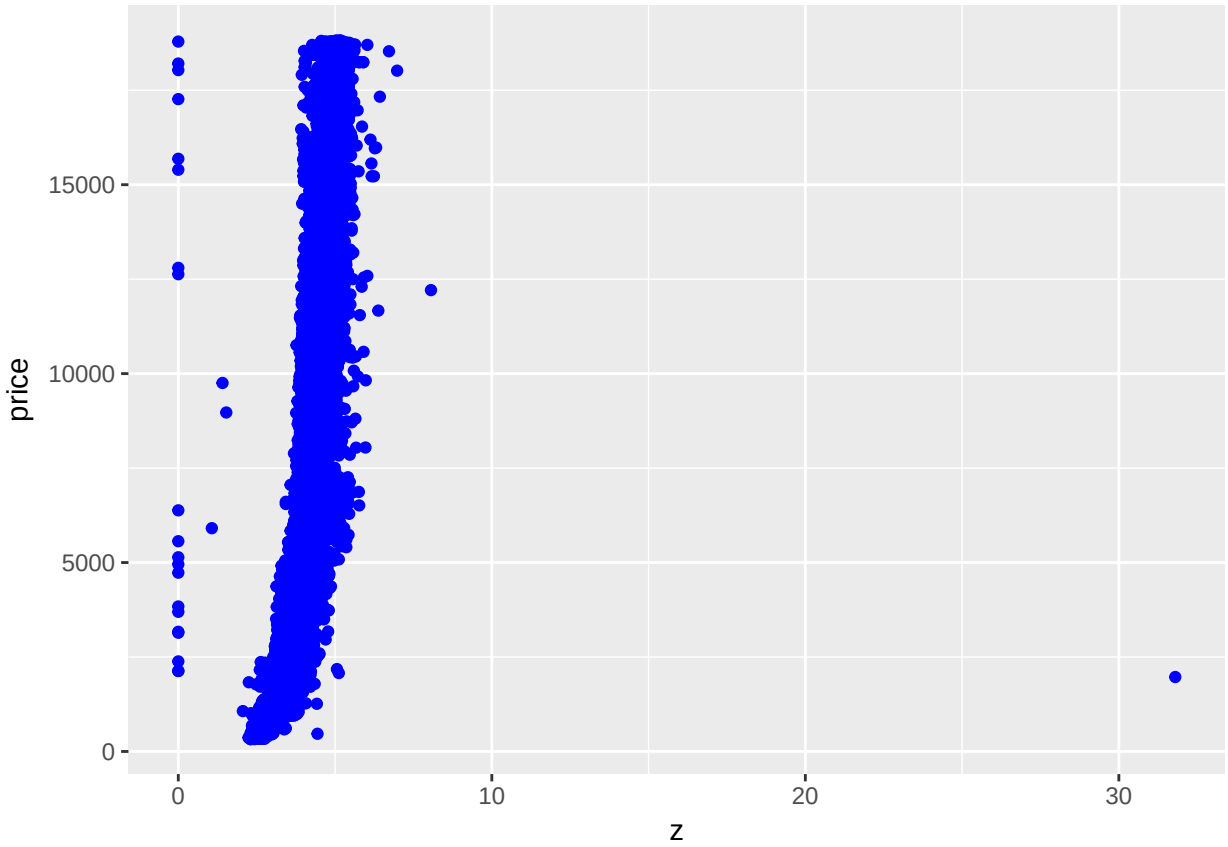

price
15000
10000
5000
0
50
60
70
80
depth


price
15000
10000
5000
0
50
60
70
80
90
table


150000
100000
50000
0
price
0
5000
10000
15000
price


15000
10000
5000
0
price
0
3
6
9
x



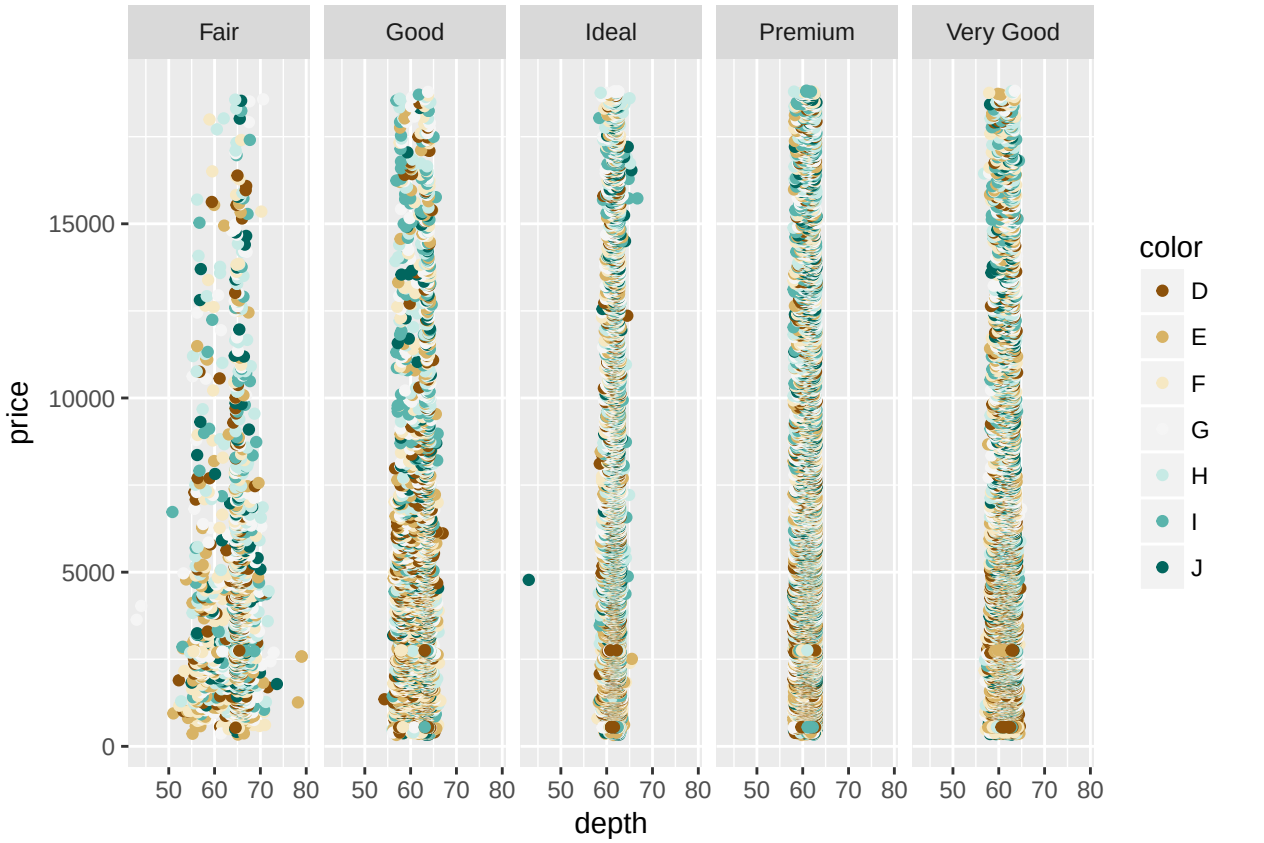


Does depth appear to be mostly independent of price?

yes.

Look at depth vs price by predictors cut (using faceting) and color (via different colors).

```
ggplot(diamonds, aes(x = depth, y = price)) +
  geom_point(aes(col= color)) + scale_color_brewer(type = "div") +
  facet_grid(.~ cut)
```



Does diamond color appear to be independent of diamond depth?

Yes, depth and color do not seem to be correlated.

Does diamond cut appear to be independent of diamond depth?

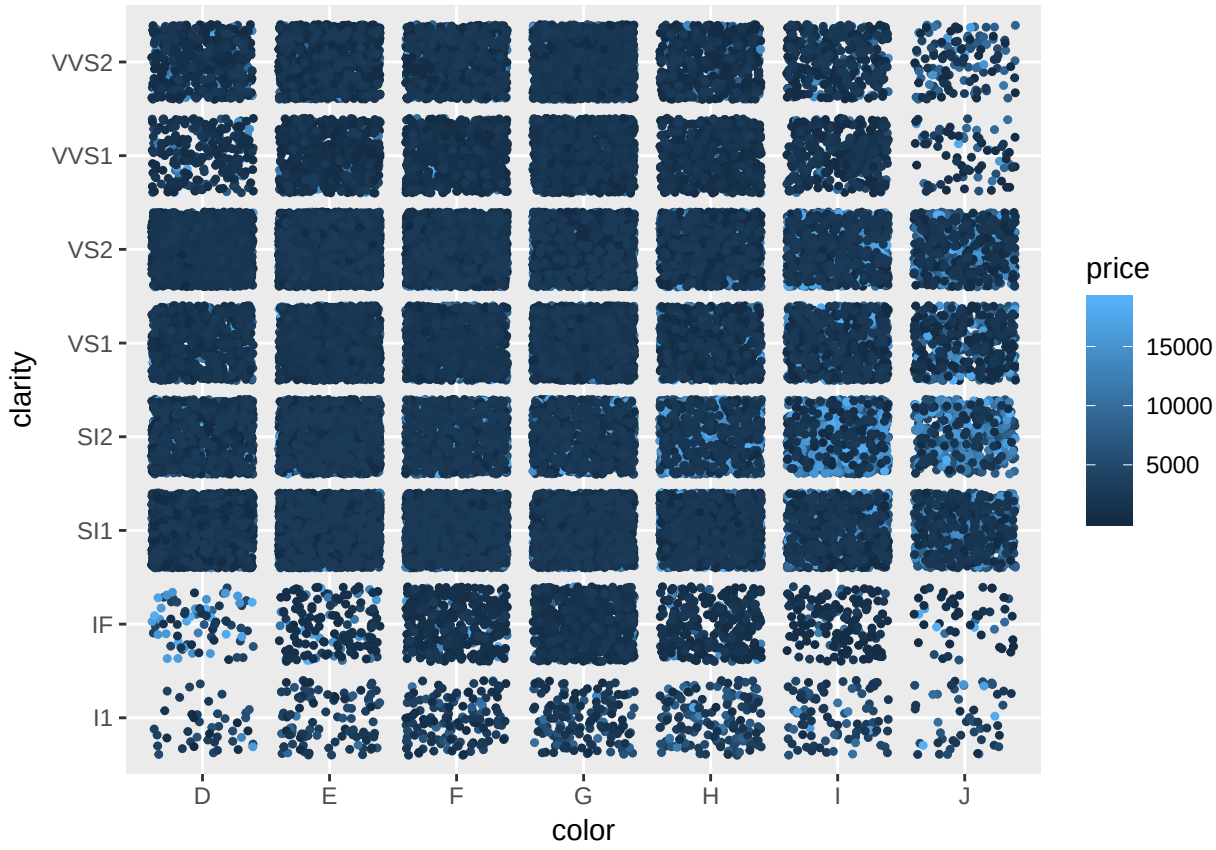
Not entirely. For fair cut, there's a lot of variance in the depth. As the cut becomes finer, the variance in the depth decreases.

Do these plots allow you to assess well if diamond cut is independent of diamond price? Yes / no

No.

We never discussed in class bivariate plotting if both variables were categorical. Use the geometry "jitter" to visualize color vs clarity. visualize price using different colors. Use a small sized dot.

```
ggplot(diamonds, aes(x = color, y= clarity)) +geom_jitter(aes(col = price), size=0.9)
```



Does diamond clarity appear to be mostly independent of diamond color?

Yes

2. Use `lm` to run a least squares linear regression using depth to explain price.

```
mod = lm(diamonds$price ~ diamonds$depth)
```

What is b, R^2 and the RMSE? What was the standard error of price originally?

```
coef(mod)
```

```
##    (Intercept) diamonds$depth
##     5763.66772      -29.64997
```

```
summary(mod)$r.squared
```

```
## [1] 0.0001133672
```

```
summary(mod)$sigma
```

```
## [1] 3989.251
```

```
sd(diamonds$price)
```

```
## [1] 3989.44
```

Are these metrics expected given the appropriate or relevant visualization(s) above?

Yes, we saw that price and depth are independent of each other. Here R^2 is almost 0, and RMSE is almost equal to the error in y.

Use `lm` to run a least squares linear regression using carat to explain price.

```
mod2 = lm(diamonds$price ~ diamonds$carat)
```

What is b, R^2 and the RMSE? What was the standard error of price originally?

```
coef(mod2)
```

```
##    (Intercept) diamonds$carat
##      -2256.361       7756.426
```

```
summary(mod2)$r.squared
```

```
## [1] 0.8493305
```

```
summary(mod2)$sigma
```

```
## [1] 1548.562
```

```
sd(diamonds$price)
```

```
## [1] 3989.44
```

Are these metrics expected given the appropriate or relevant visualization(s) above?

Yes, we saw a relationship between carat and price. The slope is positive because bigger diamonds cost more.

3. Use `lm` to run a least squares anova model using color to explain price.

```
dimond_color = as.factor(as.character(diamonds$color))
mod3= lm(diamonds$price ~ dimond_color)
```

What is b, R^2 and the RMSE? What was the standard error of price originally?

```
b = coef(mod3)
b
```

```
##   (Intercept) dimond_colorE dimond_colorF dimond_colorG dimond_colorH
##    3169.95410     -93.20162     554.93230     829.18158    1316.71510
## dimond_colorI dimond_colorJ
##    1921.92086    2153.86392
```

```
summary(mod3)$r.squared
```

```
## [1] 0.03127542
```

```
summary(mod3)$sigma
```

```
## [1] 3926.777
```

```
sd(diamonds$price)
```

```
## [1] 3989.44
```

Are these metrics expected given the appropriate or relevant visualization(s) above?

Yes. The color changes how fast the diamond increses in price as carat increases, but color alone is a pretty bad predictor.

Our model only included one feature - why are there more than two estimates in b?

The color variable underwent dummification. The linear model thinks of it as 7 dummy variables.

Verify that the least squares linear model fit gives the sample averages of each price given color combination. Make sure to factor in the intercept here.

```
ave_price = c()
for (let in c('D', 'E', 'F', 'G', 'H', 'I', 'J')){
  ave_price = c(ave_price, mean(diamonds[diamonds$color == let, ]$price))
}

ave_price_from_mod = b[1]
for ( i in 2:7){
  ave_price_from_mod = c(ave_price_from_mod, b[1] + b[i])
}
ave_price_from_mod = as.vector(ave_price_from_mod)

pacman::p_load(testthat)
expect_equal(ave_price, ave_price_from_mod)
```

Fit a new model without the intercept and verify the sample averages of each colors' prices *directly* from the entries of vector b.

```
mod4= lm(diamonds$price ~ 0 + diamonds$color)

b= coef(mod4)
b
```

```
## diamonds$colorD diamonds$colorE diamonds$colorF diamonds$colorG
##        3169.954        3076.752        3724.886        3999.136
## diamonds$colorH diamonds$colorI diamonds$colorJ
##        4486.669        5091.875        5323.818
```

What would extrapolation look like in this model? We never covered this in class explicitly.

There's no real way to extrapolate on this, since this isn't a continuous variable.

4. Use `lm` to run a least squares linear regression using all available features to explain diamond price.

```
diamonds$color = as.factor(as.character(diamonds$color))
diamonds$cut = as.factor(as.character(diamonds$cut))
diamonds$clarity = as.factor(as.character(diamonds$clarity))
mod5 = lm(price ~ ., diamonds)
```

What is b, R^2 and the RMSE? Also - provide an approximate 95% interval for predictions using the empirical rule.

```
b = coef(mod5)
b
```

```
##  (Intercept)        carat      cutGood     cutIdeal   cutPremium
##  2184.477350 11256.978307   579.751446   832.911845   762.143950
## cutVery Good       colorE       colorF       colorG       colorH
##   726.782591  -209.118085  -272.853832  -482.038904  -980.266675
##       colorI       colorJ    clarityIF   claritySI1   claritySI2
## -1466.244474 -2369.398063  5345.102246  3665.472080  2702.586294
##   clarityVS1   clarityVS2  clarityVVS1  clarityVVS2        depth
##  4578.397915  4267.223565  5007.759045  4950.814072   -63.806100
##        table            x            y            z
##   -26.474085 -1008.261098     9.608886   -50.118891
```

```
summary(mod5)$r.squared
```

```
## [1] 0.9197915
```

```
summary(mod3)$sigma
```

```
## [1] 3926.777
```

```
approx_interval = 2*summary(mod3)$sigma
```

Interpret all entries in the vector b.

The intercept refers to fair cut color D and clarity I1 with 0 in the continuous variables (carat, x, y, z, depth, and table). The slope for each variable is the increas in price for a unit increas in that variable. Cut, color, and clarity were dummified. the slopes for x, z, depth, and table are negative. Thus, in increase in those variables, will decrease the price of the diamond.

Are these metrics expected given the appropriate or relevant visualization(s) above? Can you tell from the visualizations?

Yes, (ish)

Comment on why R^2 is high. Think theoretically about diamonds and what you know about them.

Diamond prices are man-made. The price of a diamond is based on the size, cut, clarity, and color. If you have all of these variables, you're getting very close to having the z's. Thus, R^2 is high.

Do you think you overfit? Comment on why or why not but do not do any numerical testing or coding.

No. n = 53,940 and p+1 = 10. Since n is much greater than p+1, there's no concern of overfitting.

Create a visualization that shows the "original residuals" (i.e. the prices minus the average price) and the model residuals.

```
original_rediduals= data.frame(diamonds$price - mean(diamonds$price))
yhat = predict(mod5, diamonds)
model_residuals = data.frame(yhat - diamonds$price)

ggplot() + geom_density(aes(original_rediduals), data = original_rediduals, col = "green", fill = "gree
```

```
## Don't know how to automatically pick scale for object of type data.frame. Defaulting to continuous.
```

```
## Error: Column `x` must be a 1d atomic vector or a list
```

5. Reference your visualizations above. Does price vs. carat appear linear?

Not really, it seems quadratic.

Upgrade your model in #4 to use one polynomial term for carat.

```
#carat_sqr = diamonds[, "carat"]^2
#colnames(carat_sqr) = "carat_sqr"
#X_new = cbind(diamonds, carat_sqr)
mod6 = lm(price ~. +  poly(carat, 2) - carat, data = diamonds)
```

What is b, R^2 and the RMSE?

```
b = coef(mod6)
b
```

```
##     (Intercept)         cutGood        cutIdeal      cutPremium
##     21804.30931       538.33407       807.51616       747.69518
##    cutVery Good          colorE          colorF          colorG
##       678.31993      -209.43992      -284.54706      -496.84716
##          colorH          colorI          colorJ       clarityIF
##      -997.60127     -1469.25151     -2357.79746      5243.52276
##      claritySI1      claritySI2      clarityVS1      clarityVS2
##      3565.41193      2605.54013      4475.44424      4163.34947
##     clarityVVS1     clarityVVS2           depth           table
##      4904.22750      4843.80493      -116.22729       -36.37384
##               x               y               z poly(carat, 2)1
##     -2123.00617       -23.46172       -83.11272   1536647.48327
## poly(carat, 2)2
```

```
##    -76105.45945
```

```
summary(mod6)$r.squared
```

```
## [1] 0.9214777
```

```
summary(mod6)$sigma
```

```
## [1] 1118.162
```

```
length(b)
```

```
## [1] 25
```

Interpret each element in b just like previously. You can copy most of the text from the previous question but be careful. There is one tricky thing to explain.

Like before, the intercept is the price of a fair cut, D color, I1 clarity diamons with 0 in the continuous variables. Carat_sqr is negative, so the concatity in that variable is negative. Eventually diamond prices will stop increasing with increas in carat.

Is this an improvement over the model in #4? Yes/no and why.

Yes, by adding only one more predictor, the RMSE was reduced to less than half of its value in problem 4.

Define a function g that makes predictions given a vector of the same features in $\mathbb{D}$.

```
g = function(x_star){
  d1 = as.character(x_star[["cut"]])
  d2 = as.character(x_star[["color"]])
  d3 = as.character(x_star[["clarity"]])
  yhat =  as.numeric(b["(Intercept)"]) + as.numeric(b[paste0("cut", d1)]) + as.numeric(b[paste0("color"
 return(yhat)
}

g(diamonds[267,])
```

```
## [1] 1043156
```

```
predict(mod6, diamonds[267,])
```

```
##        1
## 3595.324
```

6. Use `lm` to run a least squares linear regression using a polynomial of color of degree 2 to explain price.

```
mod7 = lm(price ~ carat + cut + table + x +y + z + depth +clarity +poly(color, 2), data = diamonds)
```

```
## Warning in mean.default(x): argument is not numeric or logical: returning
## NA
```

```
## Warning in Ops.factor(x, xbar): '-' not meaningful for factors
```

```
## Error in qr.default(X): NA/NaN/Inf in foreign function call (arg 1)
```

Why did this throw an error?

Color is a categorical variable. Squaring it is meaningless.

7. Redo the model fit in #4 without using `lm` but using the matrix algebra we learned about in class. This is hard and requires many lines, but it's all in the notes.

```
data("diamonds")
diamonds1 = diamonds
```

```
#diamonds$cutFair = ifelse(diamonds$cut =="Fair", 1,0)
diamonds1$cutGood = ifelse(diamonds1$cut =="Good", 1,0)
diamonds1$cutVery_good = ifelse(diamonds1$cut =="Very Good", 1,0)
diamonds1$cutPremium = ifelse(diamonds1$cut =="Premium", 1,0)
diamonds1$cutIdeal = ifelse(diamonds1$cut =="Ideal", 1,0)
#diamonds$colorD = ifelse(diamonds$color =="D", 1,0)
diamonds1$colorE = ifelse(diamonds1$color =="E", 1,0)
diamonds1$colorE = ifelse(diamonds1$color =="F", 1,0)
diamonds1$colorH = ifelse(diamonds1$color =="H", 1,0)
diamonds1$colorI = ifelse(diamonds1$color =="I", 1,0)
diamonds1$colorJ = ifelse(diamonds1$color =="J", 1,0)
#diamonds$clarityIF = ifelse(diamonds$clarity =="IF",1,0)
diamonds1$clarityVVS1 = ifelse(diamonds1$clarity =="VVS1", 1,0)
diamonds1$clarityVVS2 = ifelse(diamonds1$clarity =="VVS2", 1,0)
diamonds1$clarityVS1 = ifelse(diamonds1$clarity =="VS1", 1,0)
diamonds1$clarityVS2 = ifelse(diamonds1$clarity =="VS2", 1,0)
diamonds1$claritySI1 = ifelse(diamonds1$clarity =="SI1", 1,0)
diamonds1$claritySI2 = ifelse(diamonds1$clarity =="SI2", 1,0)
diamonds1$cut = NULL
diamonds1$color = NULL
diamonds1$clarity = NULL

diamonds1 = diamonds1[sample(nrow(diamonds1), 0.01*nrow(diamonds1)), ]
y = diamonds1$price
diamonds1$price = NULL

X = as.matrix(cbind(1, diamonds1))
XtX = t(X) %*% X
XtXinv = solve(XtX)
H = X %*% XtXinv %*% t(X)
yhat = H %*% y

head(yhat)
```

```
##           [,1]
## [1,]  895.1403
## [2,] 3201.0603
## [3,]  674.1909
## [4,] 1194.1327
## [5,] 6670.6344
## [6,] 1219.4170
```

```
mod8 = XtXinv %*% t(X) %*% y
```

What is b, R^2 and the RMSE?

```
mod8
```

```
##                    [,1]
## 1           -6630.34033
## carat       12434.52027
## depth         210.64390
## table         -69.72016
## x            -888.02703
## y            2988.16194
```

```
## z            -6007.63753
## cutGood        834.72150
## cutVery_good   591.04929
## cutPremium     736.68550
## cutIdeal       772.83628
## colorE        -362.90808
## colorH        -789.67785
## colorI       -1240.75543
## colorJ       -1954.11211
## clarityVVS1   1358.85271
## clarityVVS2   1354.42521
## clarityVS1     951.64680
## clarityVS2     867.82447
## claritySI1     275.32668
## claritySI2    -741.53471
```

```
SSE = sum((y - yhat)^2 )
SST = sum((y - mean(y))^2)
SSR = sum((yhat - mean(y))^2)
R_sqr = (SST - SSE)/SST
R_sqr
```

```
## [1] 0.9445613
```

```
RMSE = sqrt(SSE/(nrow(X) -(ncol(X)+1)))
RMSE
```

```
## [1] 942.1474
```

Are they the same as in #4?

Close but no, because I used considerably less data points.

Redo the model fit using matrix algebra by projecting onto an orthonormal basis for the predictor space Q and the Gram-Schmidt "remainder" matrix R. Formulas are in the notes. Verify b is the same.

```
qrX = qr(X)
Q = qr.Q(qrX)
R = qr.R(qrX)


yhat_via_Q = Q %*% t(Q) %*% y

expect_equal(yhat, yhat_via_Q)
```

Generate the vectors $\hat{y}$, e and the hat matrix H.

```
H = X %*% XtXinv %*% t(X)
yhat = H %*% y
e = y - yhat
```

In one line each, verify that (a) $\hat{y}$ and e sum to the vector y (the prices in the original dataframe), (b) $\hat{y}$ and e are orthogonal (c) e projected onto the column space of X gets annhilated, (d) $\hat{y}$ projected onto the column space of X is unaffected, (e) $\hat{y}$ projected onto the orthogonal complement of the column space of X is annhilated (f) the sum of squares residuals plus the sum of squares model equal the original (total) sum of squares

```
pacman::p_load(testthat)
```

```
#Showing yhat + e= y
expect_equal(y, as.vector(yhat+e), tol= 1e-2)

# showing yhat orthogonal to e
expect_equal((t(e) %*% yhat)[1, 1], 0, tol= 1e-1)
```

```
## Error: (t(e) %*% yhat)[1, 1] not equal to 0.
## 1/1 mismatches
## [1] 0.152 - 0 == 0.152
```

```
#show that projection by H annhilates e
expect_equal( H%*%e, matrix(0, nrow =length(e), ncol= 1), tol = 1e-2)

# showing that yhat is unaffected by projection
expect_equal(H%*% yhat, yhat, tol = 1e-4)


# Showing that yhat projected to (I-H) gets annhilated
expect_equal((diag(length(y))- H)%*%yhat, matrix(0, nrow =length(yhat), ncol = 1), tol= 1e-5)

expect_equal(SSR + SSE,SST)
```

8. Fit a linear least squares model for price using all interactions and also 5-degree polynomials for all continuous predictors.

```
data(diamonds)

diamonds$cut = factor(as.character(diamonds$cut))
diamonds$color = factor(as.character(diamonds$color))
diamonds$clarity = factor(as.character(diamonds$clarity))

mod9 = lm(price ~ .*. + poly(carat, 5) + poly(table, 5)+ poly(depth, 5) + poly(x, 5) + poly(y, 5) + poly
```

Report R^2, RMSE, the standard error of the residuals (s_e) but you do not need to report b.

```
summary(mod9)$r.squared
```

```
## [1] 0.9728255
```

```
summary(mod9)$sigma
```

```
## [1] 659.2249
```

```
sd(summary(mod9)$residuals)
```

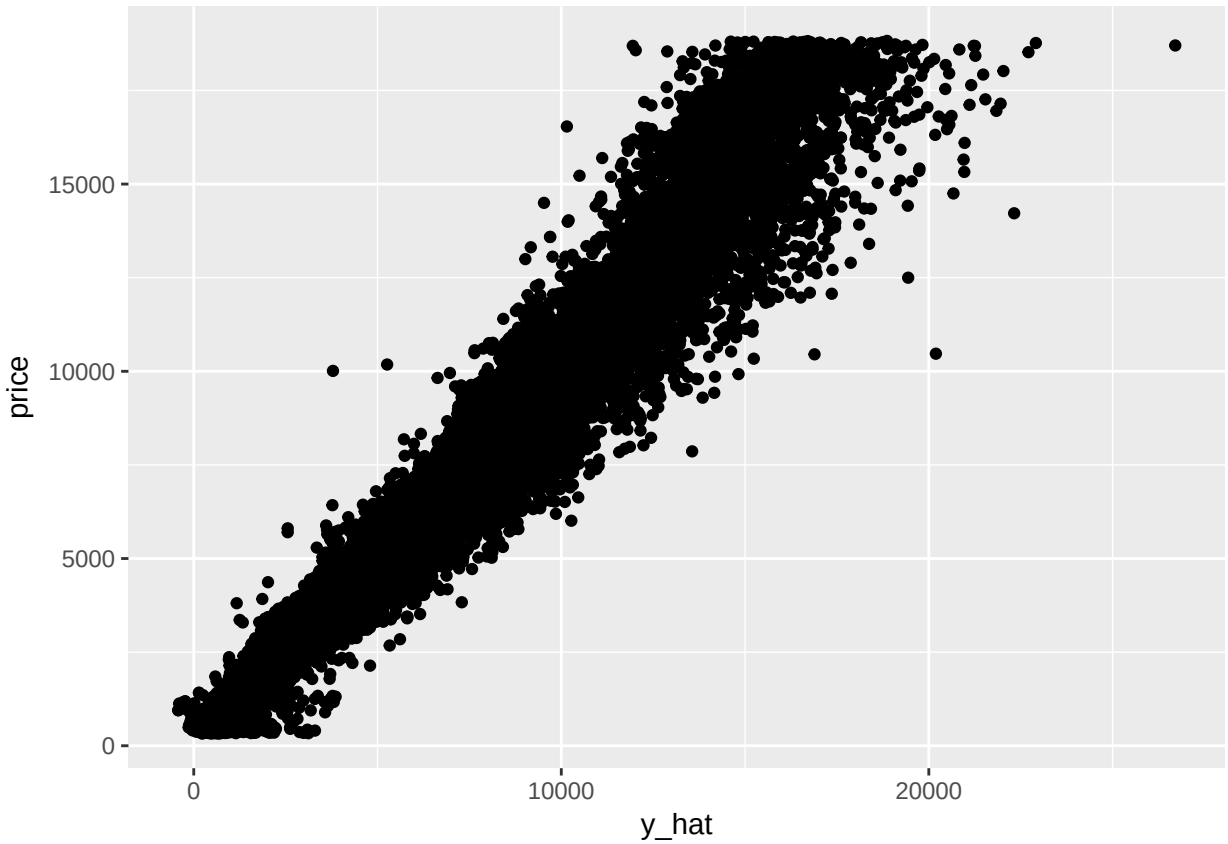
```
## [1] 657.6464
```

Create an illustration of y vs. $\hat{y}$.

```
y_hat = predict(mod9, diamonds)
```

```
## Warning in predict.lm(mod9, diamonds): prediction from a rank-deficient fit
## may be misleading
```

```
ggplot(diamonds, aes(x = y_hat, y = price)) + geom_point()
```



How many diamonds have predictions that are wrong by $1,000 or more ?

```
sum(abs(diamonds$price - y_hat) >= 1000)
```

```
## [1] 4583
```

R^2 now is very high and very impressive. But is RMSE impressive? Think like someone who is actually using this model to e.g. purchase diamonds.

95% of prediuctions will fall within $1,320 above or below the true value. I never bougt a diamond, but I'd imagine diamond shoppers wouldn't tolerate such a margin of error.

What is the degrees of freedom in this model?

```
length(coef(mod9))
```

```
## [1] 260
```

Do you think g is close to h^* in this model? Yes / no and why?

Yes, estimation error is minimized because of the large n.

Do you think g is close to f in this model? Yes / no and why?

Probably. Our model is very flexible, so it fits f's curves well.

What more degrees of freedom can you add to this model to make g closer to f?

Who knows?!

Even if you allowed for so much expressivity in $\mathcal{H}$ that f was an element in it, there would still be error due to ignorance of relevant information that you haven't measured. What information do you think can help? This is not a data science question - you have to think like someone who sells diamonds.

Market conditions, supply, demand.

9. Validate the model in #8 by reserving 10% of $\mathbb{D}$ as test data. Report oos standard error of the residuals

```
k = 10
n = nrow(diamonds)
test_indices = sample(1:n, n/k)

train_indices = setdiff(1:n, test_indices)
X_train = diamonds[train_indices, ]
X_test = diamonds[test_indices, ]

mod10 = lm(price ~ .*. + poly(carat, 5) + poly(table, 5)+ poly(depth, 5) + poly(x, 5) + poly(y, 5) + pol

yhat = predict(mod10, X_test)
```

```
## Warning in predict.lm(mod10, X_test): prediction from a rank-deficient fit
## may be misleading
```

```
oose = sd(yhat - X_test$price)

oose
```

```
## [1] 3680796
```

Compare the oos standard error of the residuals to the standard error of the residuals you got in #8 (i.e. the in-sample estimate). Do you think there's overfitting?

Yes, the oos s_e is a few orders of nagnitude greater than the in sample s_e. This is definately overfit.

Extra-credit: validate the model via cross validation.

```
#TO-DO if you want extra credit

k = 10

n = nrow(diamonds)

X_shuffled = diamonds[sample(1:n), ]
osse_vec = c()

for (i in 1 : k){
  test_indices = (i - 1)*(n / k) + 1:(n / k)
  train_indices = setdiff(1:n, test_indices)

  X_train = diamonds[train_indices, ]
  X_test = diamonds[test_indices, ]

  y_train = diamonds[train_indices, "price"]

  mod10 = lm(price[train_indices] ~ .*. + poly(carat, 5) + poly(table, 5)+ poly(depth, 5) + poly(x, 5) +

  yhat = predict(mod10, X_test)
  oose = sd(yhat - X_test$price)

  osse_vec =  c(osse_vec, oose)

}
```

```
## Warning in predict.lm(mod10, X_test): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(mod10, X_test): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(mod10, X_test): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(mod10, X_test): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(mod10, X_test): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(mod10, X_test): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(mod10, X_test): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(mod10, X_test): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(mod10, X_test): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(mod10, X_test): prediction from a rank-deficient fit
## may be misleading
```

```
oose_vec
```

```
## Error in eval(expr, envir, enclos): object 'oose_vec' not found
```

Is this result much different than the single validation? And, again, is there overfitting in this model?

Not much different. Yes, this is overfit.

10. The following code (from plec 14) produces a response that is the result of a linear model of one predictor and random ϵ.

```
rm(list = ls())
set.seed(1003)
n = 100
beta_0 = 1
beta_1 = 5
xmin = 0
xmax = 1
x = runif(n, xmin, xmax)
#best possible model
h_star_x = beta_0 + beta_1 * x

#actual data differs due to information we don't have
epsilon = rnorm(n)
y = h_star_x + epsilon
```

We then add fake predictors. For instance, here is the model with the addition of 2 fake predictors:

```
p_fake = 2
X = matrix(c(x, rnorm(n * p_fake)), ncol = 1 + p_fake)
mod = lm(y ~ X)
```

Using a test set hold out, find the number of fake predictors where you can reliably say "I overfit". Some example code is below that you may want to use:

```
#TO-DO

p_fake = 0
k = 5
X = x

osse_vec = c()
ise_vec = c()

for (i in 1: 78){
  p_fake = p_fake + i

  X = cbind(X, rnorm(n))
  test_indices = sample(1:n, n/k)
  train_indices = setdiff(1:n, test_indices)
  X_train = X[train_indices, ]
  X_test = X[test_indices, ]
  y_train = y[train_indices,]
  y_test = y[test_indices, ]

  mod = lm(y_train ~ .,  data.frame(X_train))
  ise = sd(mod$residuals)
  y_hat_oss = predict(mod, data.frame(X_test))

  oose = sd(y_hat_oss - y[test_indices, ])

  osse_vec = c(osse_vec, oose)
  ise_vec = c(ise_vec, ise)

}
```

```
## Error in y[train_indices, ]: incorrect number of dimensions
```

```
error = data.frame(iter = 1:78, oose = osse_vec, ise = ise_vec)
```

```
## Error in data.frame(iter = 1:78, oose = osse_vec, ise = ise_vec): arguments imply differing number o
```

```
ggplot(error) +geom_line(aes(x = iter, y = oose), col= 'red') + geom_line(aes(x = iter, y= ise), col= '
```

```
## Error in ggplot(error): object 'error' not found
```

The overfitting can be seen when the out of sample error and the in sample error diverge from each other.