HW03p

Darshan Patel

April 13, 2018

```
rm(list = ls())
if (!require("pacman")){install.packages("pacman")}
```

```
## Loading required package: pacman
```

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

1. Load pacakge `ggplot2` below using `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.

```
?diamonds
#to allow for readability of categorical variables
diamonds$cut = factor(as.character(diamonds$cut))
diamonds$color = factor(as.character(diamonds$color))
diamonds$clarity = factor(as.character(diamonds$clarity))
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    : Factor w/ 5 levels "Fair","Good",..: 3 4 2 4 2 5 5 5 1 5 ...
##  $ color  : Factor w/ 7 levels "D","E","F","G",..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Factor w/ 8 levels "I1","IF","SI1",..: 4 3 5 6 4 8 7 3 6 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 ...
```

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

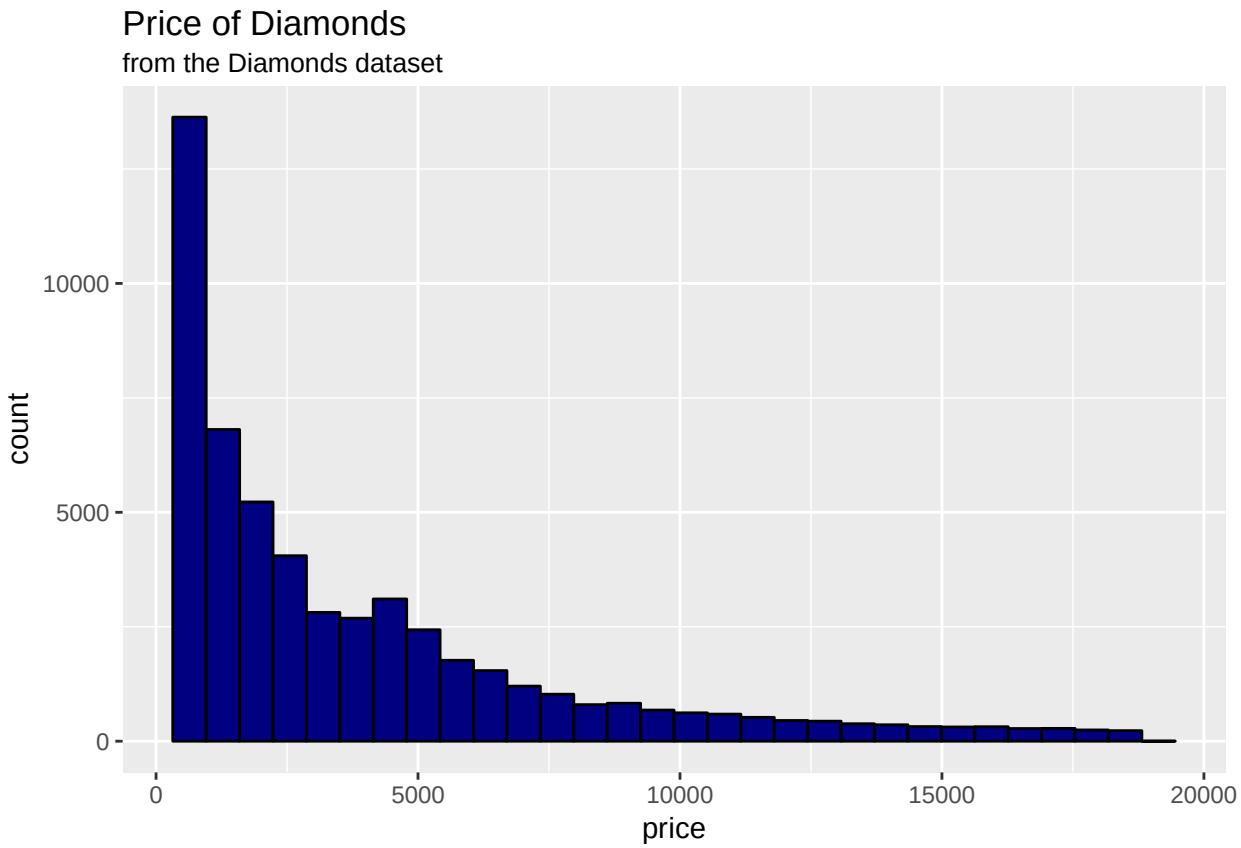
Answer: In this dataset, there are $53,940$ samples, or $n = 53,940$. To describe these samples, there are 10 features, or $p = 10$. The features describe the character of the diamond samples, such as carat (weight of the diamond), quality of the cut, diamond color, its dimensions and price. The most likely response metric is `price` because price is generally determined from other variables such as carat and cut.

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.

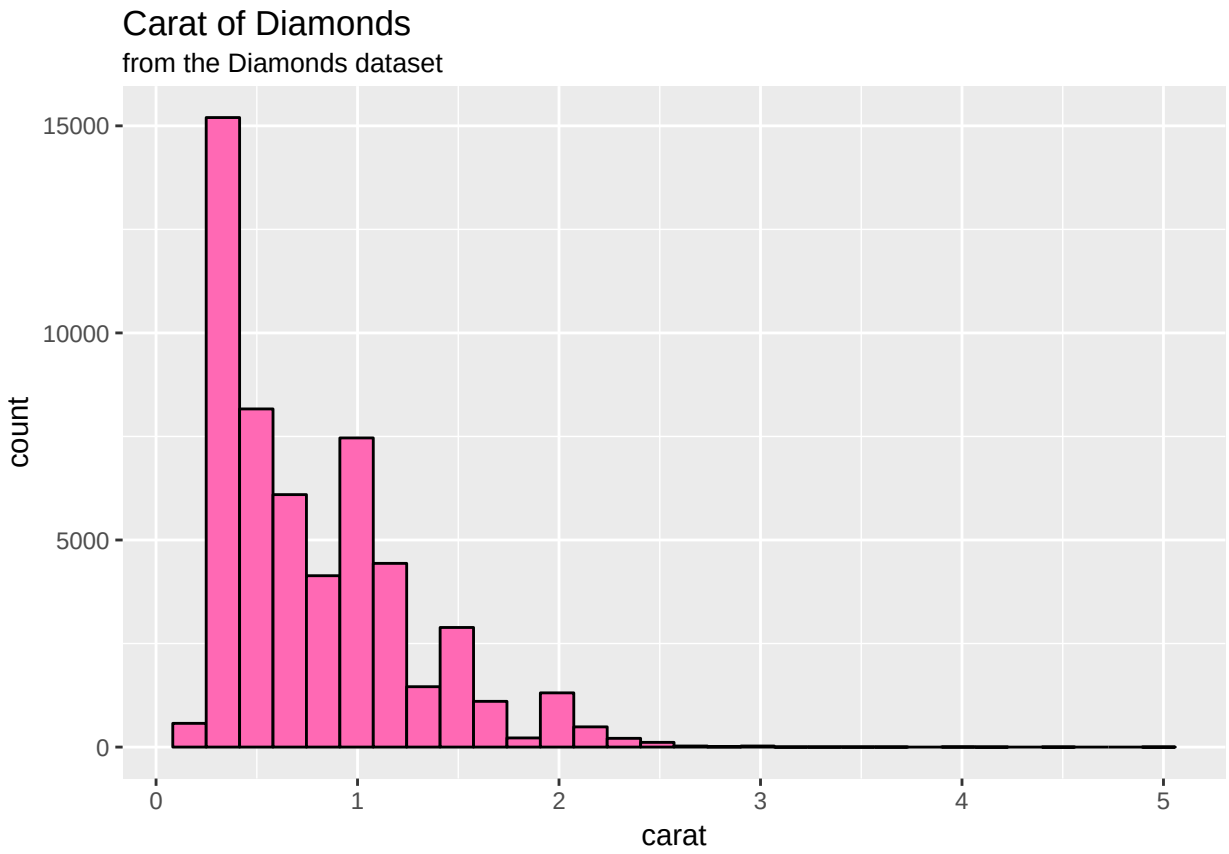
```
ggplot(diamonds, aes(price)) + geom_histogram(color = "black", fill = "navyblue") +
ggtitle("Price of Diamonds", subtitle = "from the Diamonds dataset")
```

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

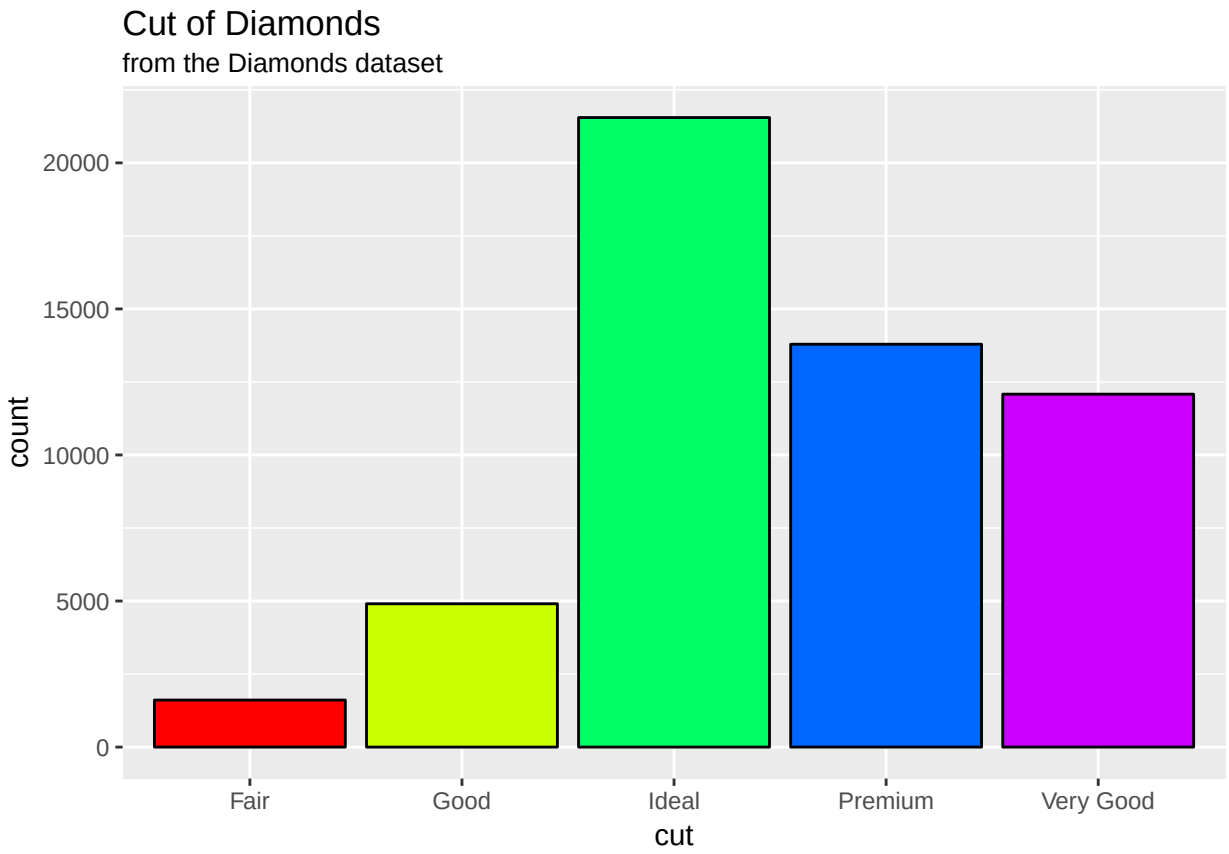



```
ggplot(diamonds, aes(carat)) + geom_histogram(color = "black", fill = "hotpink") +
ggtitle("Carat of Diamonds", subtitle = "from the Diamonds dataset")
```

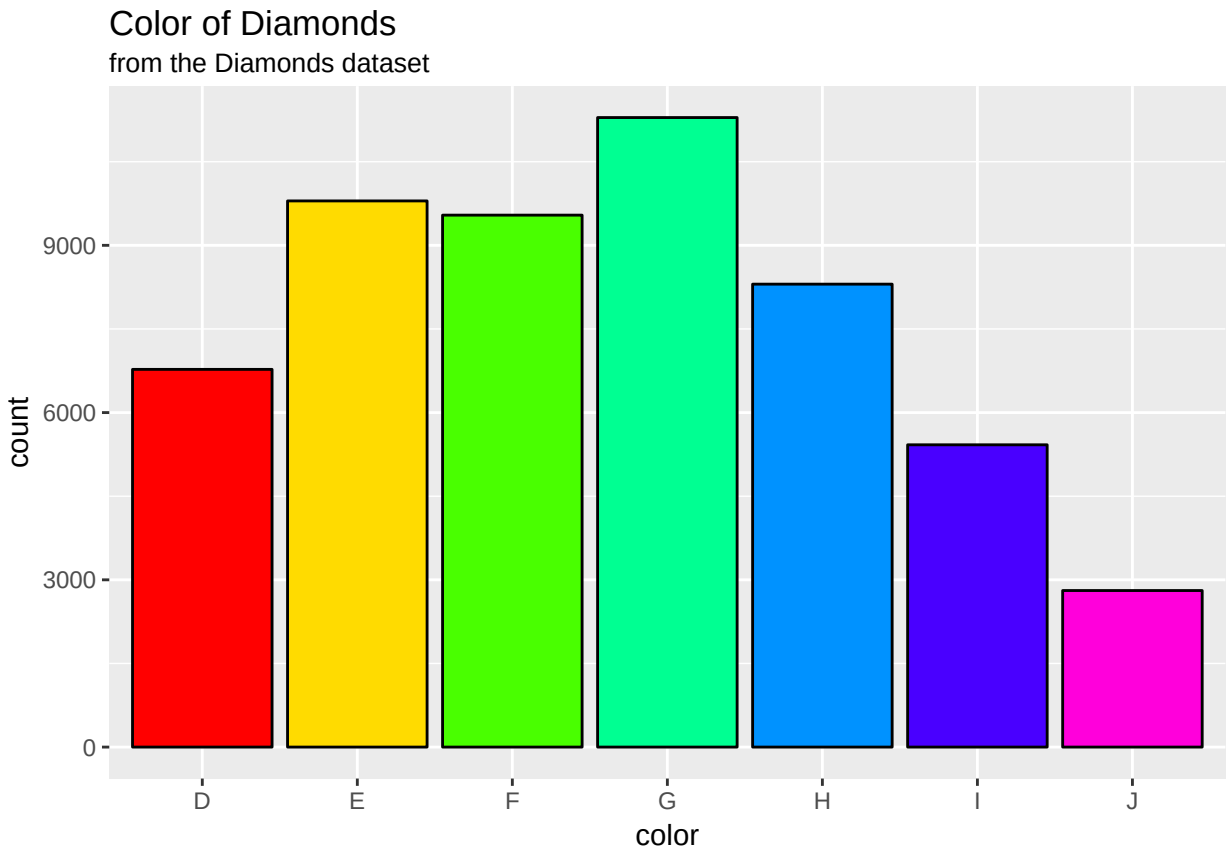
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



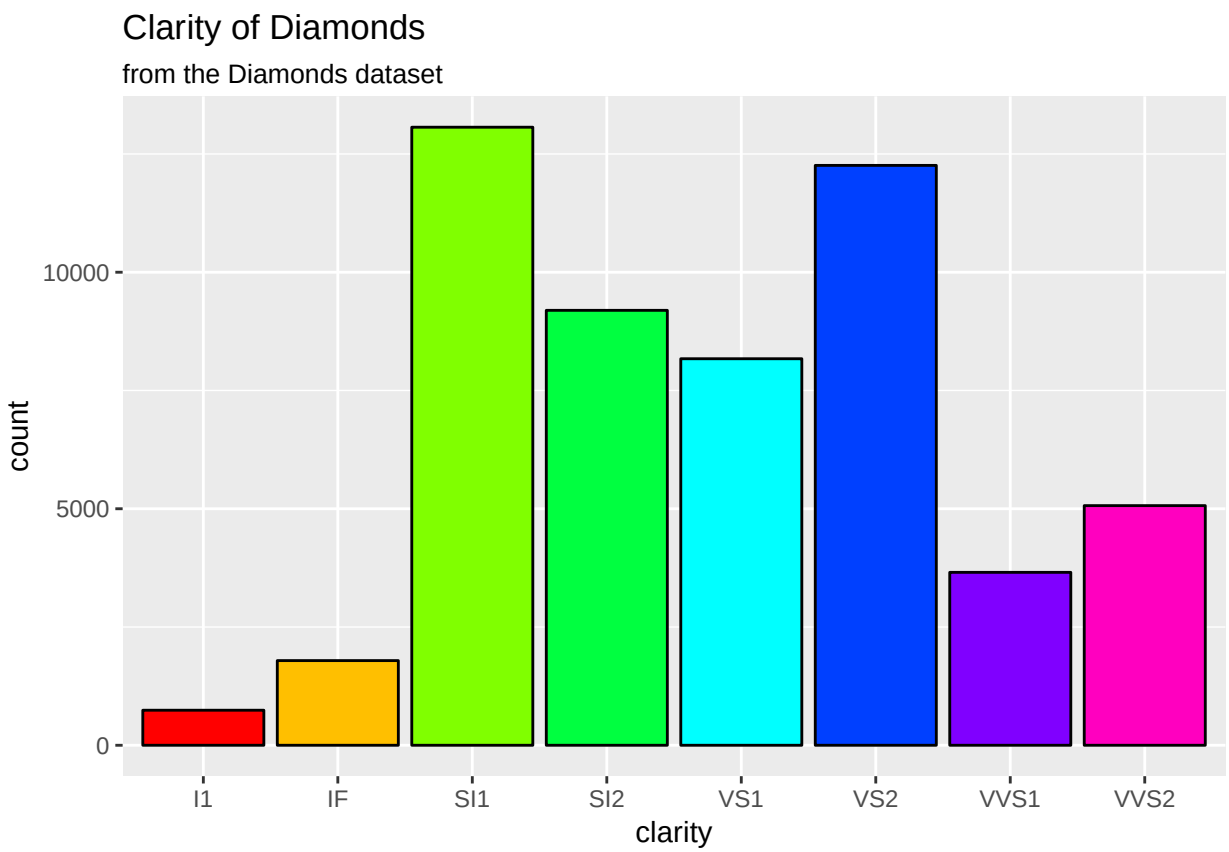

```
ggplot(diamonds, aes(cut)) + geom_bar(color = "black", fill = rainbow(5)) +
ggtitle("Cut of Diamonds", subtitle = "from the Diamonds dataset")
```




```
ggplot(diamonds, aes(color)) + geom_bar(color = "black", fill = rainbow(7)) +
ggtitle("Color of Diamonds", subtitle = "from the Diamonds dataset")
```

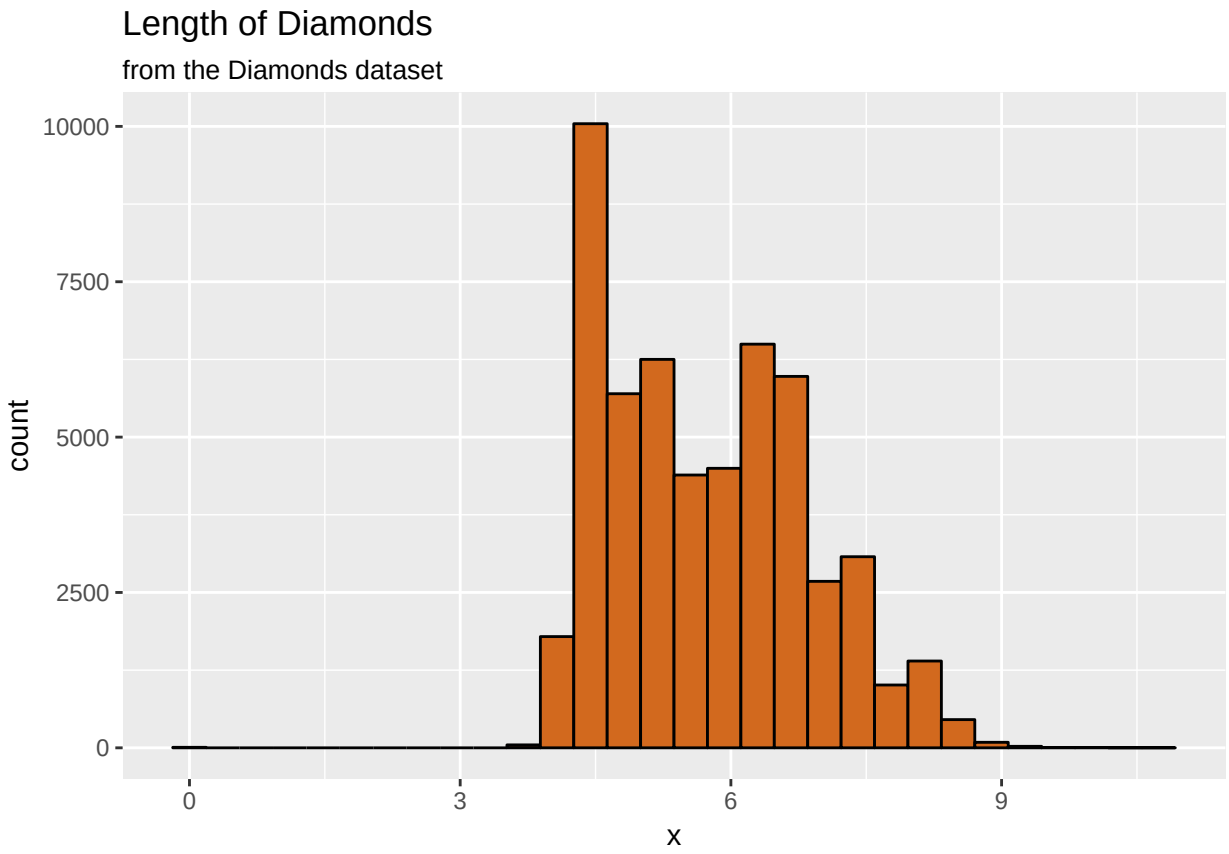



```
ggplot(diamonds, aes(clarity)) + geom_bar(color = "black", fill = rainbow(8)) +
  ggtitle("Clarity of Diamonds", subtitle = "from the Diamonds dataset")
```



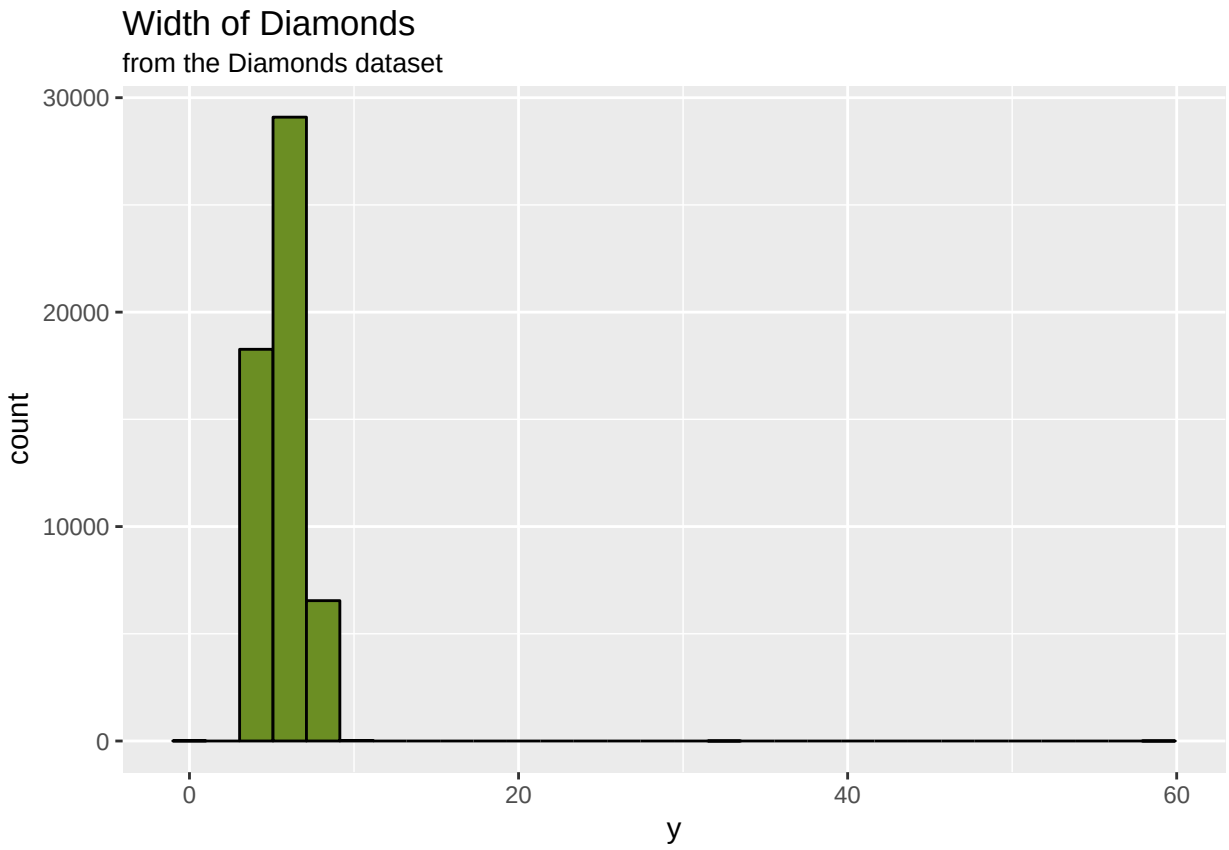

```
ggplot(diamonds, aes(x)) + geom_histogram(color = "black", fill = "chocolate") +
ggtitle("Length of Diamonds", subtitle = "from the Diamonds dataset")
```

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



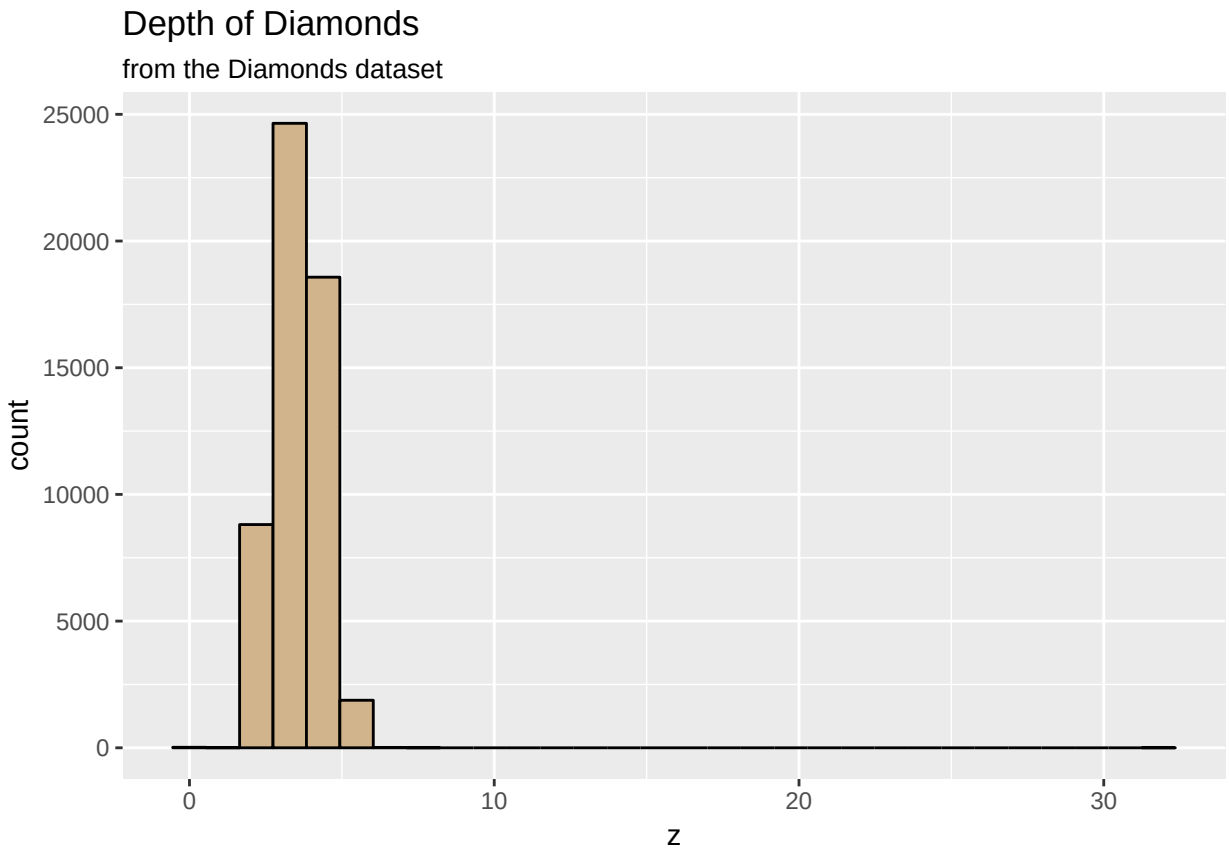

```
ggplot(diamonds, aes(y)) + geom_histogram(color = "black", fill = "olivedrab") +
ggtitle("Width of Diamonds", subtitle = "from the Diamonds dataset")
```

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



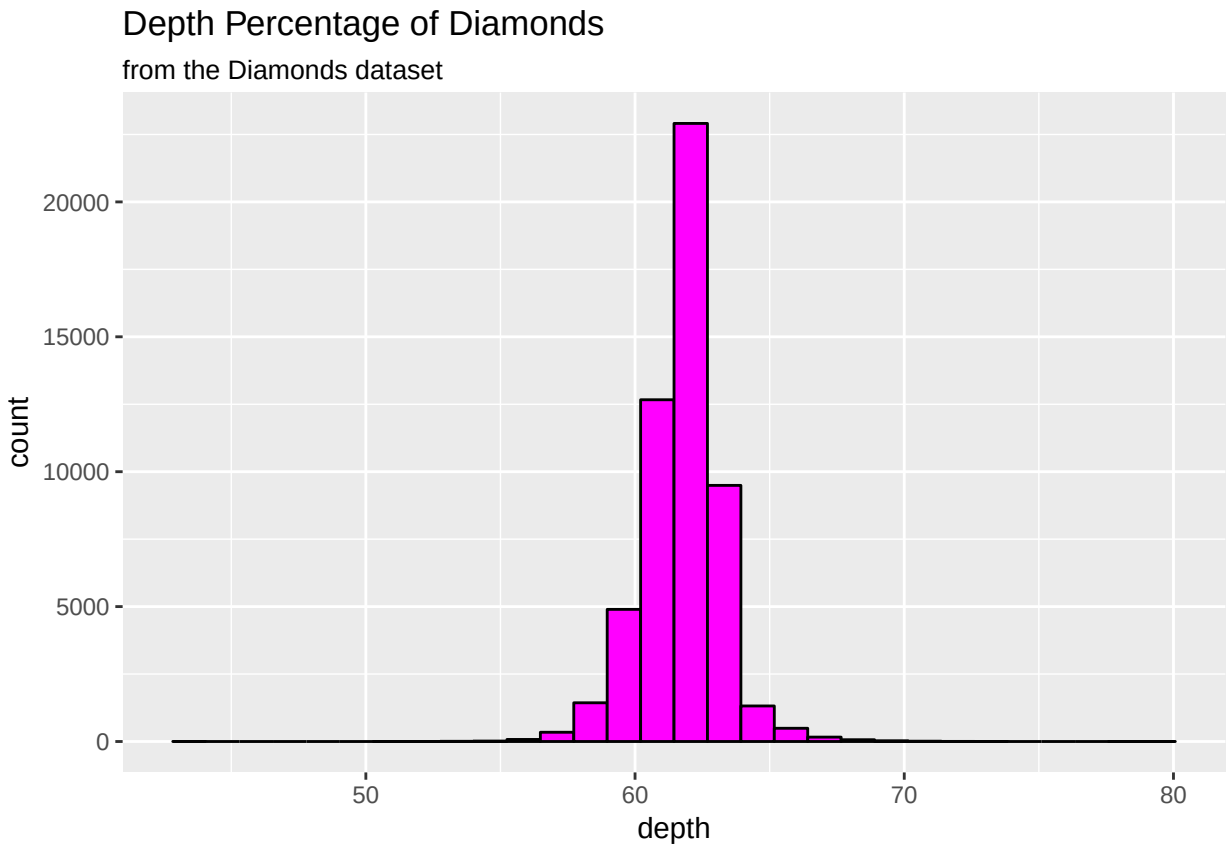

```
ggplot(diamonds, aes(z)) + geom_histogram(color = "black", fill = "tan") +
ggtitle("Depth of Diamonds", subtitle = "from the Diamonds dataset")
```

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



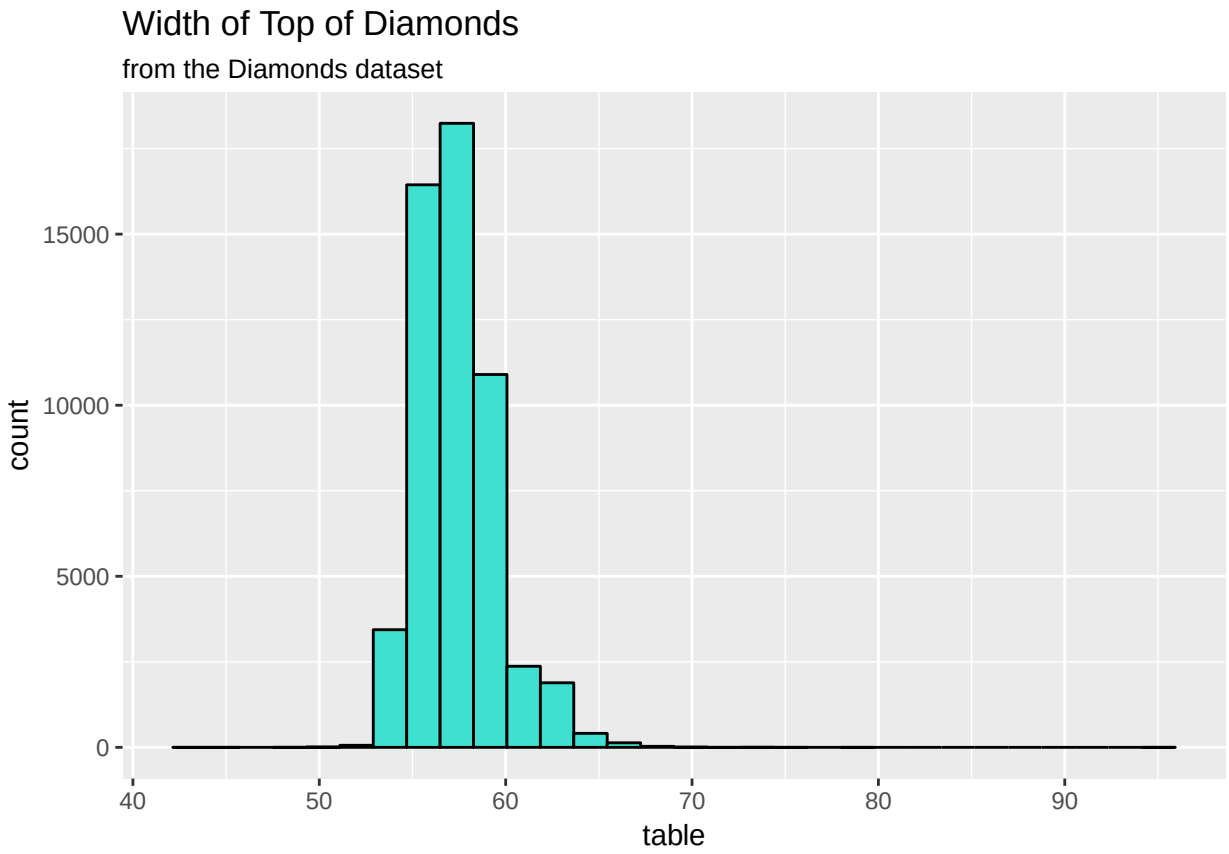

```
ggplot(diamonds, aes(depth)) + geom_histogram(color = "black", fill = "magenta") +
ggtitle("Depth Percentage of Diamonds", subtitle = "from the Diamonds dataset")
```

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



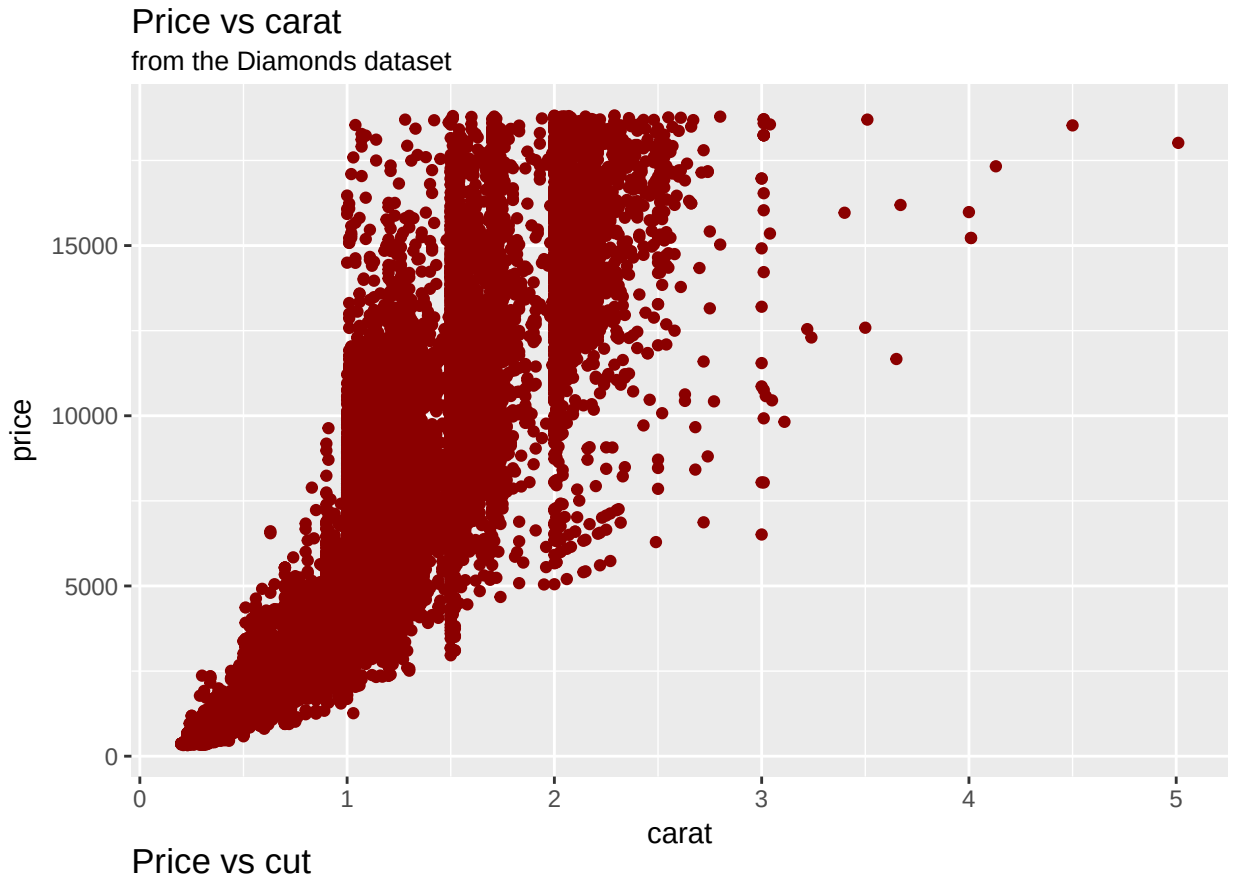

```
ggplot(diamonds, aes(table)) + geom_histogram(color = "black", fill = "turquoise") +
ggtitle("Width of Top of Diamonds", subtitle = "from the Diamonds dataset")
```

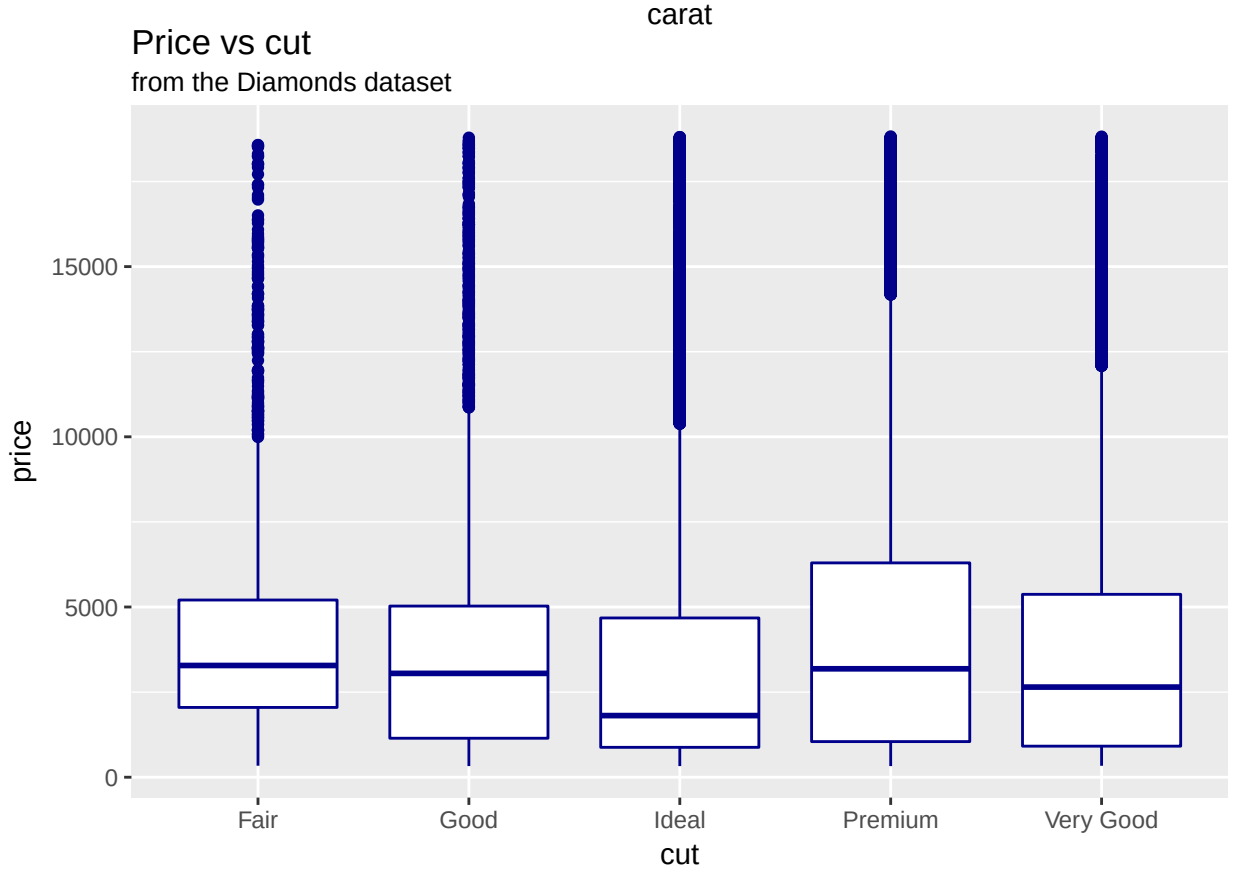
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

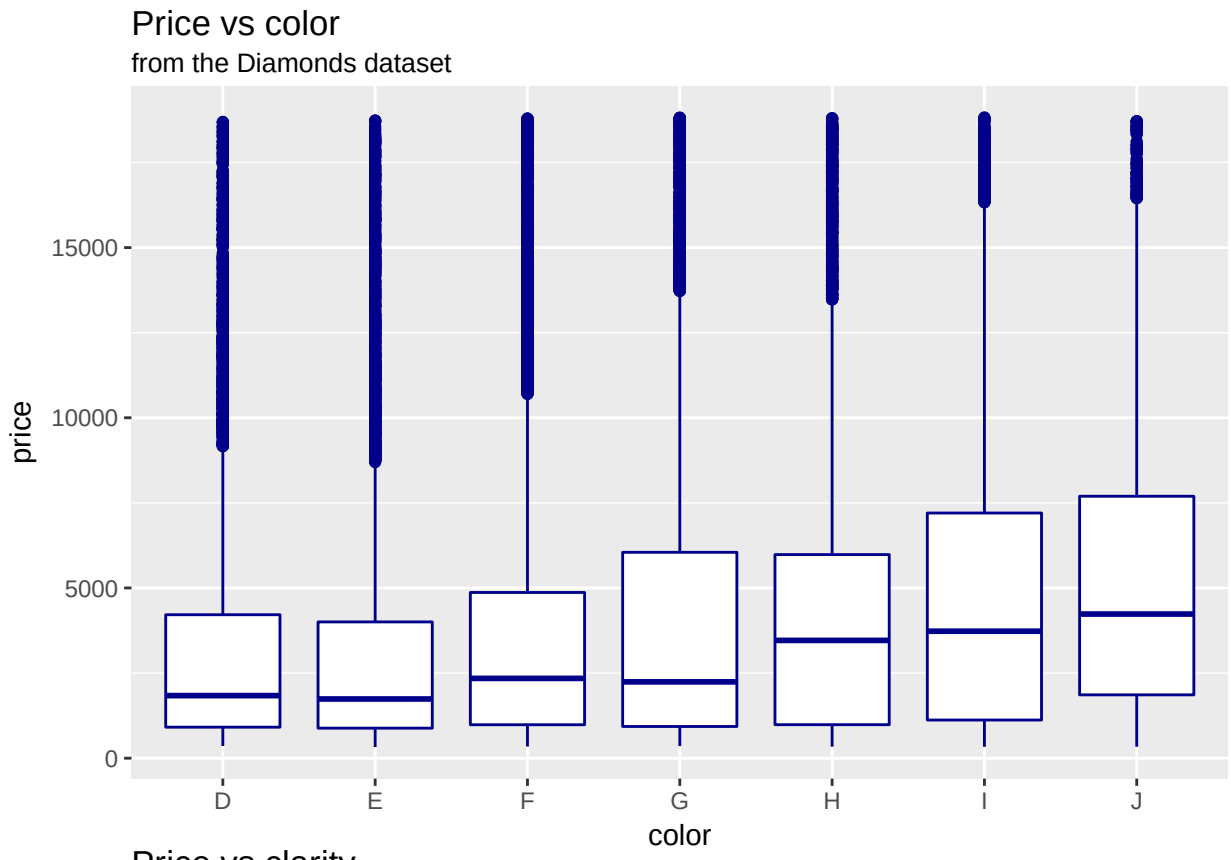


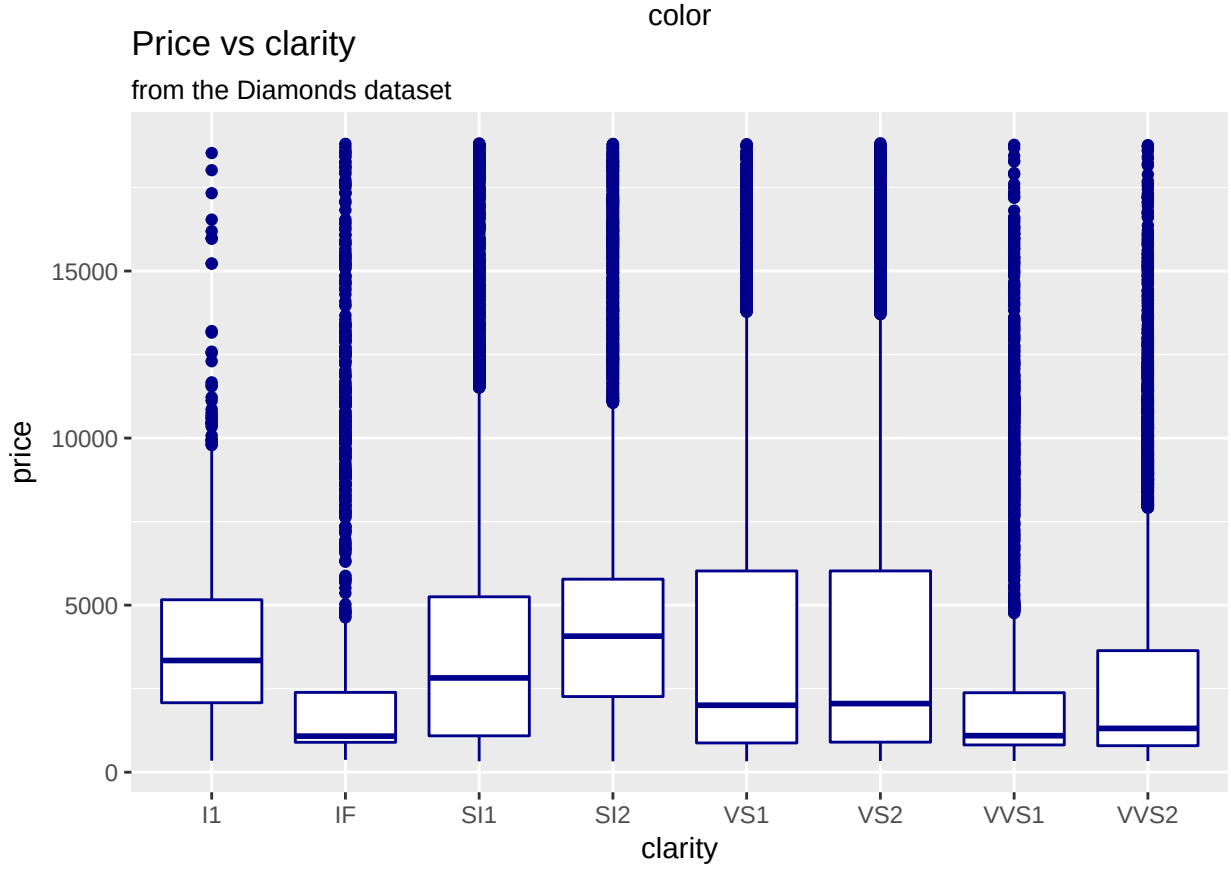
Use `ggplot` to look at the bivariate distributions of the response versus *all* predictors. Make sure you handle categorical predictors differently from continuous predictors. This time employ a for loop when an logic that handles the predictor type.

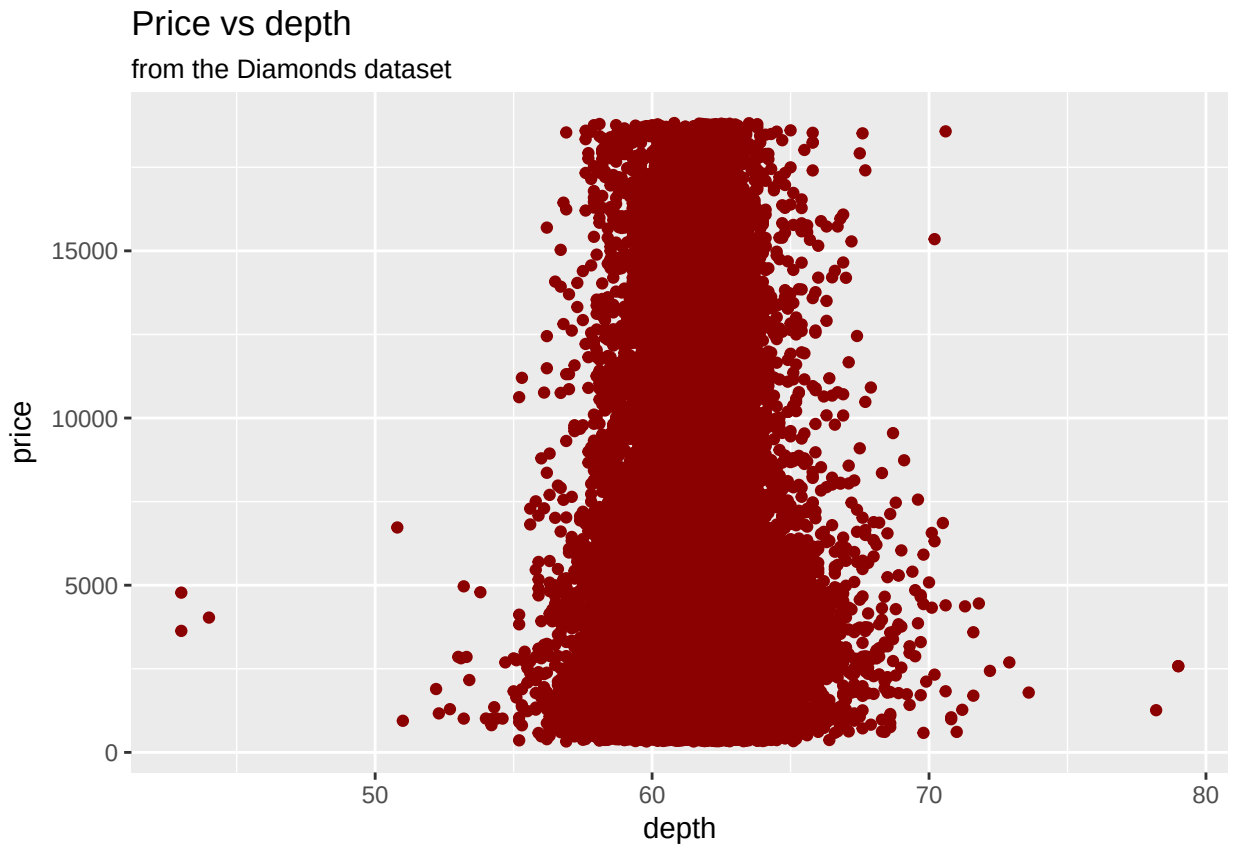
```
# a list of TRUES/FALSES if a column is numeric or not
df = sapply(diamonds, is.numeric)

for(i in 1:ncol(diamonds)){
  colname = as.vector(as.matrix(diamonds[,i]))
  variable = colnames(diamonds[,i])
  if(typeof(colname) == "character"){
    graphics = ggplot(diamonds, aes(x = colname, y = price)) +
               geom_boxplot(colour = "darkblue") +
               labs(x = colnames(diamonds[,i])) +
               ggtitle(paste("Price vs", variable),
                       subtitle = "from the Diamonds dataset")

  }
  else{
    graphics = ggplot(diamonds, aes(x = colname, y = price)) +
               geom_point(colour = "darkred", fill = "lightgreen") +
               labs(x = colnames(diamonds[,i]))  +
               ggtitle(paste("Price vs", variable),
                       subtitle = "from the Diamonds dataset")
  }
    plot(graphics)
}
```

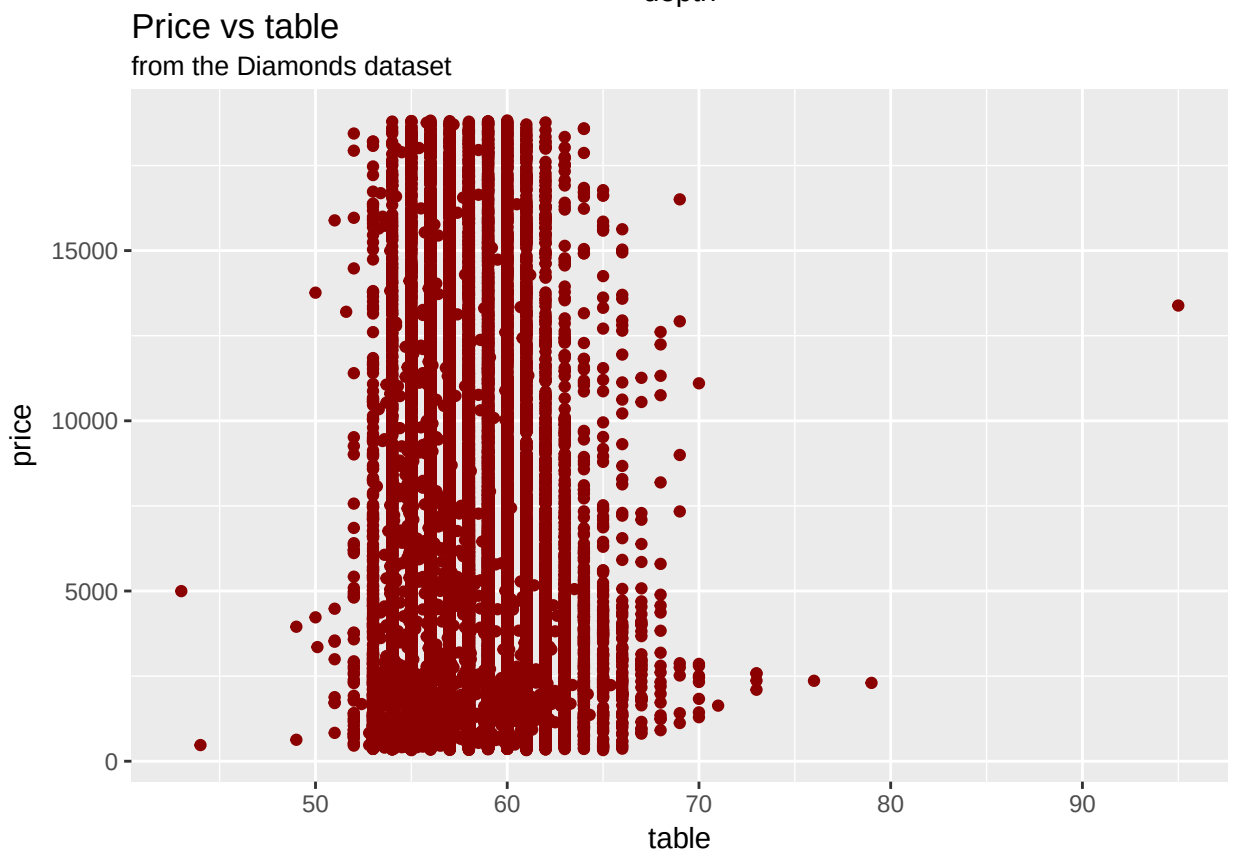

Price vs carat
from the Diamonds dataset
price
15000
10000
5000
0
0
1
2
3
4
5
carat

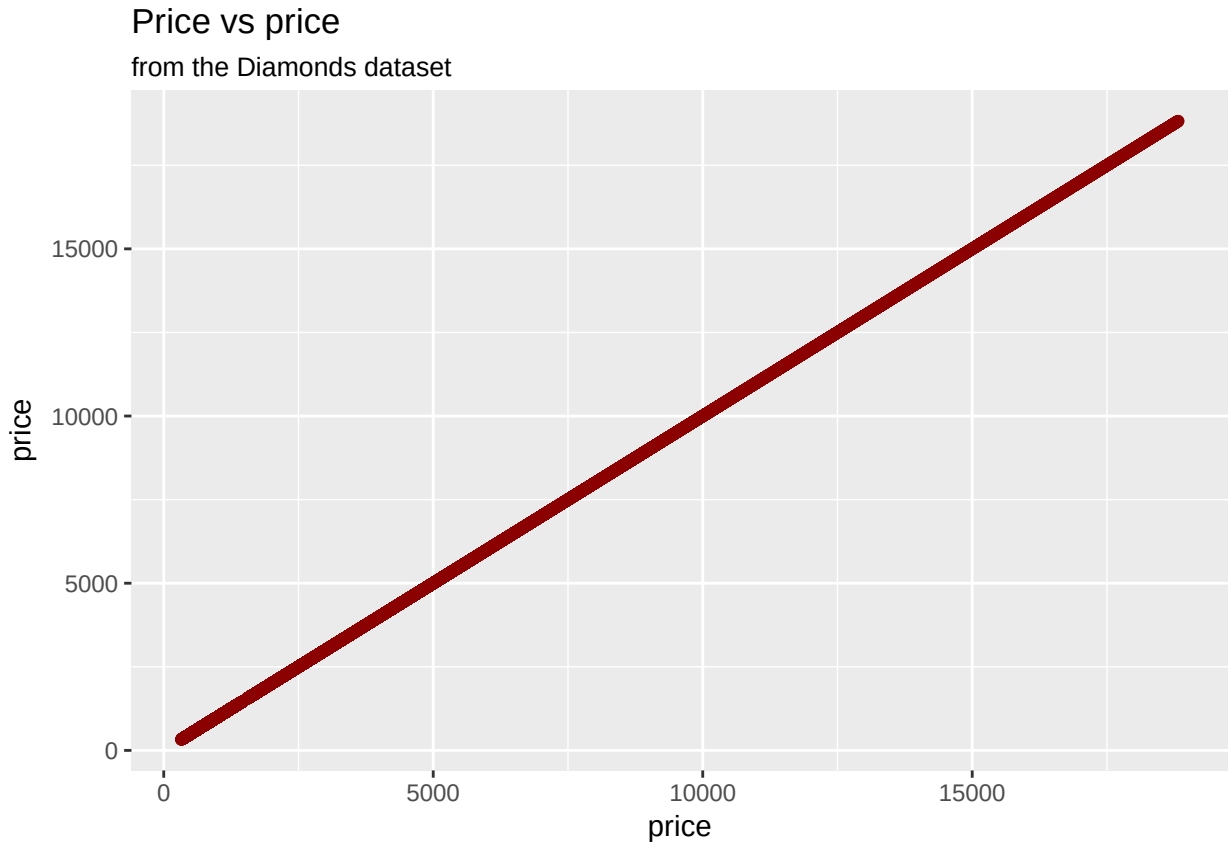

Price vs cut
from the Diamonds dataset
price
15000
10000
5000
0
Fair
Good
Ideal
Premium
Very Good
cut

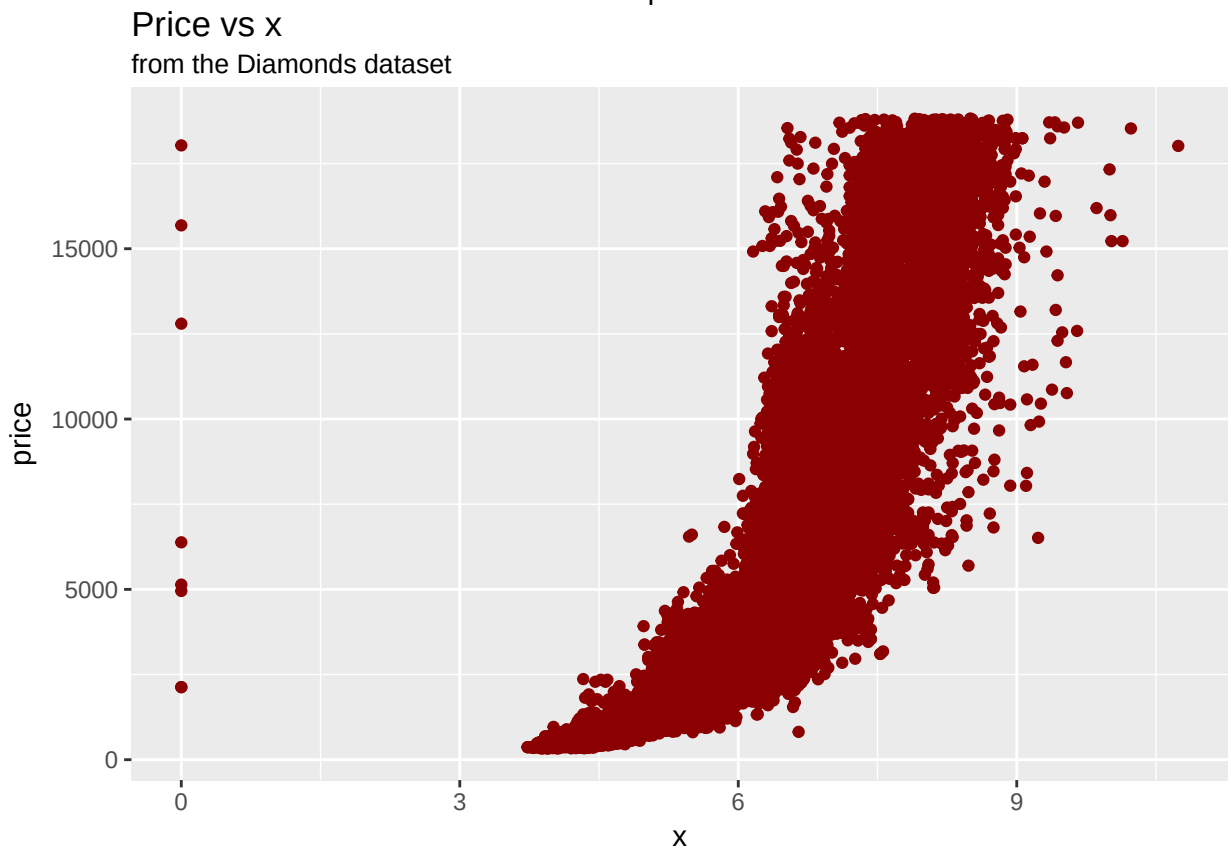

Price vs color
from the Diamonds dataset
price
15000
10000
5000
0
D
E
F
G
H
I
J
color

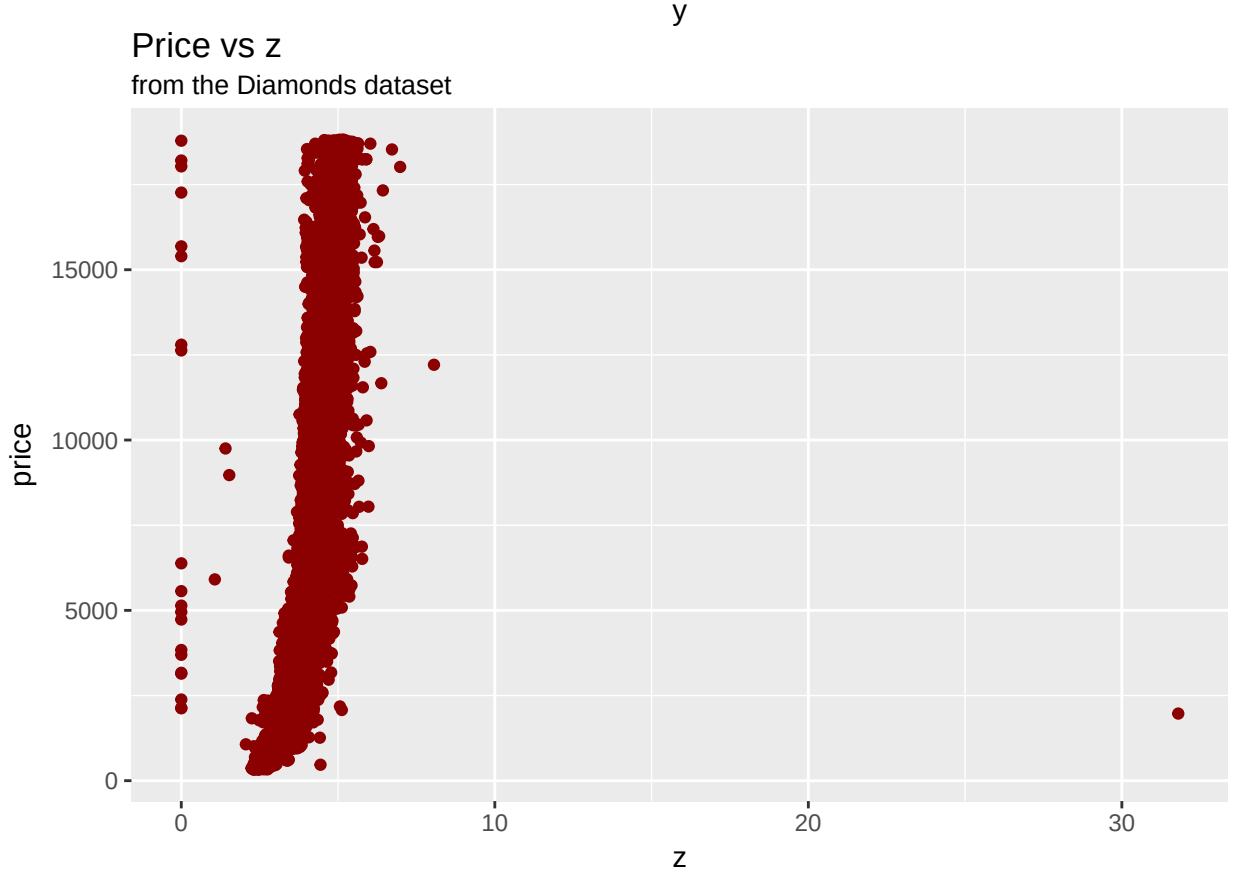

Price vs clarity
from the Diamonds dataset
price
15000
10000
5000
0
I1
IF
SI1
SI2
VS1
VS2
VVS1
VVS2
clarity


Price vs depth
from the Diamonds dataset
price
15000
10000
5000
0
50
60
70
80
depth


Price vs table
from the Diamonds dataset
price
15000
10000
5000
0
50
60
70
80
90
table


Price vs price
from the Diamonds dataset
price
15000
10000
5000
0
0
5000
10000
15000
price


Price vs x
from the Diamonds dataset
price
15000
10000
5000
0
0
3
6
9
x

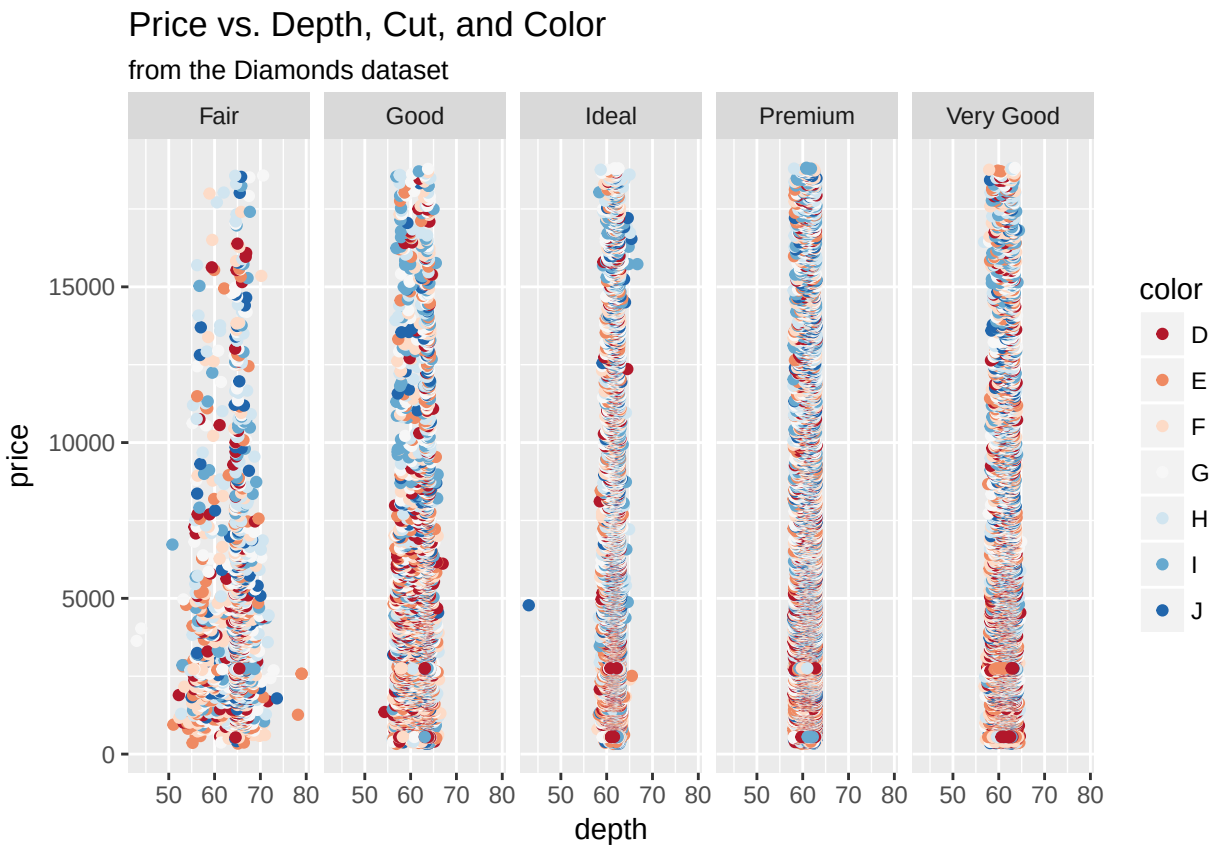


Does depth appear to be mostly independent of price?

Answer: Depth appears to be mostly independent of price as it does not show a consistent price at various depths or any correlation.

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

```
depth_price = ggplot(diamonds, aes(x = depth, y = price)) +
              geom_point(aes(col = color)) +
              ggtitle("Price vs. Depth, Cut, and Color",
                      subtitle = "from the Diamonds dataset") +
              scale_color_brewer(palette = "RdBu") +
              facet_grid(. ~ cut)
depth_price
```



Does diamond color appear to be independent of diamond depth?

Answer: Yes.

Does diamond cut appear to be independent of diamond depth?

Answer: No.

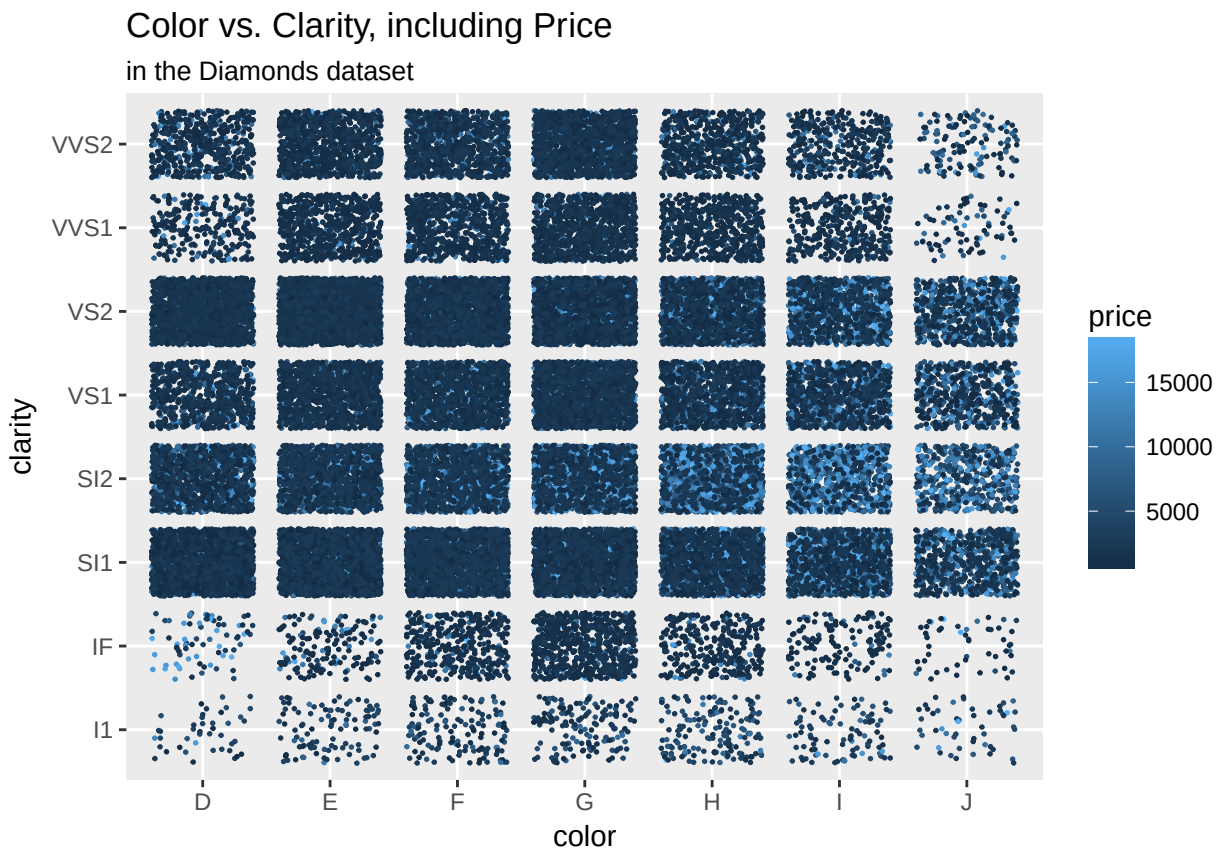
Do these plots allow you to assess well if diamond cut is independent of diamond price? Yes / no

Answer: 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.

```
color_clarity = ggplot(diamonds, aes(x = color, y = clarity, color = price)) +
                geom_jitter(size = 0.3) +
                ggtitle("Color vs. Clarity, including Price",
```

```
                      subtitle = "in the Diamonds dataset")
color_clarity
```



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

Answer: Diamond clarity appears to be mostly independent of diamond color.

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

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

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

Respectively, b_0 and b_1 is

```
coef(depth_price)
```

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

The R^2 value is

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

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

and the RMSE is

```
summary(depth_price)$sigma
```

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

The standard error of price originally was

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

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

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

Answer: Given the appropriate/revelant visualizations, these metrics are expected since there is no clear correlation between depth and price and R^2 was calculated to be nearly 0.

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

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

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

Respectively, b_0 and b_1 is

```
coef(carat_price)
```

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

The R^2 value is

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

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

and the RMSE is

```
summary(carat_price)$sigma
```

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

The standard error of price originally was

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

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

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

Answer: Given the appropriate/relevants visualizations, these metrics are expected because there does seem to be some correlation between carat and price.

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

```
color_price = lm(diamonds$price ~ diamonds$color)
```

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

b is

```
coef(color_price)
```

```
##     (Intercept) diamonds$colorE diamonds$colorF diamonds$colorG
##      3169.95410       -93.20162       554.93230       829.18158
## diamonds$colorH diamonds$colorI diamonds$colorJ
##      1316.71510      1921.92086      2153.86392
```

The R^2 value is

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

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

and the RMSE is

```
summary(color_price)$sigma
```

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

The standard error of price originally was

```
summary(color_price)$coefficients[1,2]
```

```
## [1] 47.70694
```

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

Answer: Given the appropriate/relevant visualizations, these metrics are expected since here does not seem to be a correlation between color and price hence a low R^2 value.

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

Answer: There are more two estimates in b because `color` is a categorical variable made up of 7 colors/variables/features.

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.

```
b0 = as.numeric(coef(color_price)[1])
mean_from_lm = c(b0)

for(i in 2:length(coef(color_price))){
  mean_from_lm[i] = b0 + as.numeric(coef(color_price)[i])
}

expect_equal(mean_from_lm[1], mean(diamonds$price[diamonds$color == "D"]))
expect_equal(mean_from_lm[2], mean(diamonds$price[diamonds$color == "E"]))
expect_equal(mean_from_lm[3], mean(diamonds$price[diamonds$color == "F"]))
expect_equal(mean_from_lm[4], mean(diamonds$price[diamonds$color == "G"]))
expect_equal(mean_from_lm[5], mean(diamonds$price[diamonds$color == "H"]))
expect_equal(mean_from_lm[6], mean(diamonds$price[diamonds$color == "I"]))
expect_equal(mean_from_lm[7], mean(diamonds$price[diamonds$color == "J"]))
```

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

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

```
## 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
```

```
# store the average of each colors' price into a vector and then test if
# the coefficients match up with the ones from the intercept-less model
val = aggregate(diamonds$price, by = list(diamonds$color), FUN = mean)[,2]

expect_equal(val, as.numeric(coef(color_price_no_intercept)))
```

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

Answer: Extrapolation would look like a diamond with no color would cost \$0, which makes sense. A "diamond" that is "colorless" is nothing and nothing does cost \$0, theoretically.

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

```
all_features_price = lm(price ~ ., diamonds)
```

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

b is

```
coef(all_features_price)
```

```
##  (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
```

The R^2 value is

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

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

The RMSE is

```
summary(all_features_price)$sigma
```

```
## [1] 1130.094
```

The 95% interval for predictions is

```
paste("±", 2*summary(all_features_price)$sigma)
```

```
## [1] "± 2260.18885197136"
```

Interpret all entries in the vector b.

Answer: The entries in the vector b gives the intercept and slope values for each predictor. With the categorical features, it is possible to understand which features give more of a boost to price. The `ideal cut` gives a price increase of \$832, the most from all the cut whereas the `good cut` gives the least price increase of \$579. The same analysis can be done with the colors. As for the numeric features, an increase of 1 mm in the `y`, or width of a diamond, brings an increase of \$9 whereas an increase of 1 mm in the `x` or `z`, or length or depth, causes a decrease of \$1008 and \$50 respectively.

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

Answer: Given the appropriate/relevant visualizations, it is impossible to make a conclusion about the metrics since the metrics are given from an analysis of all features instead of just one. The bivariate visualizations from above will not help analyze the metrics here.

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

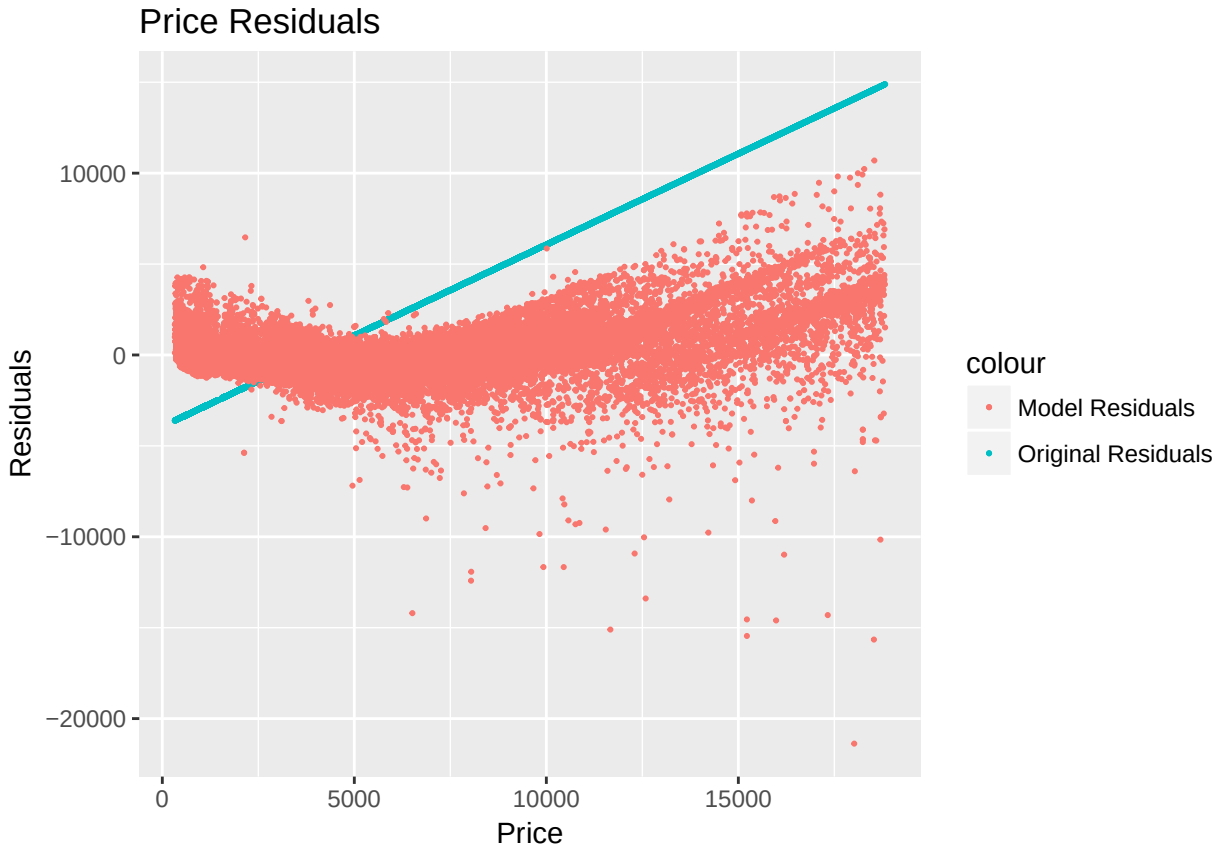
Answer: R^2 is high because diamonds prices are usually calculated per carat. Since carat is provided, it does not take much effort to create a linear model comparing the two, albeit some other features do play a smaller role. Therefore the other features give supplementary information on the price to help create a fuller explanation of the phenomenon.

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

Answer: No overfit was done because $n >> p$. In layman terms, the sample size is much larger than the number of features therefore there is no outrageous attempt to fit through all the data points.

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

```
df = data.frame(diamonds$price,
                diamonds$price - mean(diamonds$price),
                all_features_price$residuals)
ggplot(df, aes(x = df[,1], y = value, color = "variable")) +
  geom_point(aes(y = df[,2], col = "Original Residuals"), size = 0.4) +
  geom_point(aes(y = df[,3], col = "Model Residuals"), size = 0.4) +
  labs(x = "Price", y = "Residuals") + ggtitle("Price Residuals")
```



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

Answer: Price and carat does appear to be linear.

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

```
poly_carat_features_price = lm(price ~ poly(carat, 2, raw = TRUE) + ., diamonds)
```

What is b, R^2 and the RMSE?

b is

```
coef(poly_carat_features_price)
```

```
##                 (Intercept) poly(carat, 2, raw = TRUE)1
##                  9807.97904                 16144.75809
```

```
## poly(carat, 2, raw = TRUE)2                     carat
##                  -1028.81806                        NA
##                      cutGood                   cutIdeal
##                    538.33407                  807.51616
##                   cutPremium              cutVery Good
##                    747.69518                  678.31993
##                       colorE                     colorF
##                   -209.43992                 -284.54706
##                       colorG                     colorH
##                   -496.84716                 -997.60127
##                       colorI                     colorJ
##                  -1469.25151                -2357.79746
##                    clarityIF                  claritySI1
##                   5243.52276                 3565.41193
##                   claritySI2                  clarityVS1
##                   2605.54013                 4475.44424
##                   clarityVS2                 clarityVVS1
##                   4163.34947                 4904.22750
##                  clarityVVS2                       depth
##                   4843.80493                 -116.22729
##                        table                           x
##                    -36.37384                -2123.00617
##                            y                           z
##                    -23.46172                  -83.11272
```

The R^2 value is

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

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

and the RMSE is

```
summary(poly_carat_features_price)$sigma
```

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

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.

Answer: The entries in the vector b gives the intercept and slope values for each predictor. The intercept value gives a base price of the diamond, which is \$9807. The intercept value gives the price of a diamond that is considered reference variable for the categorical variables, in this case which is color D, fair cut, and clarity of I1. By modifying the carat predictor to have a polynomial term, it shows that price increases by \$16144 when carat goes up by 1 while also decreasing by \$1028 when carat squared goes up by 1.

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

Answer: Compared to the model in #4, the R^2 went slighty up while the RMSE went slightly down. In my judgement, I would say this is an improvement to the model because more variance is explained by the model and smaller residuals are formed.

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

```
g = function(x){

  b = coef(poly_carat_features_price)

  yhat = b["(Intercept)"] +
```

```
        (x["carat"] * b["poly(carat, 2, raw = TRUE)1"]) +
        (x["carat"]^2 * b["poly(carat, 2, raw = TRUE)2"]) +
        switch(as.numeric(x["clarity"]),
               1 , 0,
               2 , b["claritySI2"],
               3 , b["claritySI1"],
               4 , b["clarityVS2"],
               5 , b["clarityVS1"],
               6 , b["clarityVVS2"],
               7 , b["clarityVVS1"],
               8 , b["clarityIF"]) +
        switch(as.numeric(x["color"]),
               1 , 0,
               2 , b["colorE"],
               3 , b["colorF"],
               4 , b["colorG"],
               5 , b["colorH"],
               6 , b["colorI"],
               7 , b["colorJ"]) +
        switch(as.numeric(x["cut"]),
               1 , 0,
               2 , b["cutGood"],
               3 , b["cutVery Good"],
               4 , b["cutPremium"],
               5 ,b["cutIdeal"]) +
        (x["x"]*b["x"]) +
        (x["y"]*b["y"]) +
        (x["z"]*b["z"]) +
        (x["table"]*b["table"]) +
        (x["depth"]*b["depth"])
  as.numeric(yhat)
}
```

Performing a test trial on a random row from the diamond set, we see that for a certain diamond, it predicts a price of

```
rand = sample(1:nrow(diamonds), 1)
g(diamonds[rand,])
```

```
## [1] 5116.684
```

when the actual price is

```
as.numeric(diamonds[rand, 7])
```

```
## [1] 5164
```

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

```
color_price = lm(price ~ poly(color, 2, raw = TRUE), diamonds)
```

```
## Warning in Ops.factor(X, Y, ...): '^' not meaningful for factors
```

```
## Error in lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...): 0 (non-NA) cases
```

Why did this throw an error?

Answer: This threw an error because a categorical value cannot be raised to a power. (e.g, `red` squared makes no sense).

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.

```
cat_matrix = model.matrix(diamonds$price ~ diamonds$cut +
                            diamonds$color +
                            diamonds$clarity)

X = cbind(diamonds[,1],cat_matrix,diamonds[,5:6], diamonds[,8:10])
y = diamonds$price

indices = sample(1 : nrow(X), 2000)
X = as.matrix(X[indices, ])
y = as.matrix(diamonds[indices,7])

Xt = t(X)
XtX = Xt %*% X
XtXinv = solve(XtX)
b = XtXinv %*% Xt %*% y
```

What is b, R^2 and the RMSE?

b is

```
b
```

```
##                               price
## carat                  12822.123557
## (Intercept)             5600.059174
## diamonds$cutGood         208.425227
## diamonds$cutIdeal        679.827969
## diamonds$cutPremium      509.228300
## diamonds$cutVery Good    533.524925
## diamonds$colorE         -126.001616
## diamonds$colorF         -177.850855
## diamonds$colorG         -273.824335
## diamonds$colorH         -928.568116
## diamonds$colorI        -1401.126595
## diamonds$colorJ        -2102.102021
## diamonds$clarityIF      4512.701429
## diamonds$claritySI1     2839.535335
## diamonds$claritySI2     1783.067778
## diamonds$clarityVS1     3850.438614
## diamonds$clarityVS2     3385.120824
## diamonds$clarityVVS1    4017.964967
## diamonds$clarityVVS2    3977.844573
## depth                    -92.158977
## table                      1.562167
## x                      -3369.142899
## y                       1712.931276
## z                         13.557866
```

The R^2 value is

```
yhat = X %*% b
e = y - yhat
Rsq = (var(y) - var(e)) / var(y)
Rsq
```

```
##           price
## price 0.9316222
```

and the RMSE is

```
SSE = t(e) %*% e
MSE = 1 / (ncol(X)) * SSE
RMSE = sqrt(MSE)
RMSE
```

```
##          price
## price 9688.759
```

Are they the same as in #4?

Answer: No. We did not use the entire $\mathcal{D}$. However, if we redo #4 using the randomized 2000 points and create a linear model from it,

```
subset_diamond = data.frame(diamonds[indices,])
subset_price = lm(price ~ .,subset_diamond)
coef(subset_price)
```

```
##  (Intercept)        carat      cutGood     cutIdeal   cutPremium
##  5600.059173 12822.123557   208.425227   679.827969   509.228300
## cutVery Good       colorE       colorF       colorG       colorH
##   533.524925  -126.001616  -177.850855  -273.824335  -928.568116
##       colorI       colorJ    clarityIF   claritySI1   claritySI2
## -1401.126595 -2102.102021  4512.701429  2839.535335  1783.067778
##   clarityVS1   clarityVS2  clarityVVS1  clarityVVS2        depth
##  3850.438614  3385.120824  4017.964967  3977.844573   -92.158977
##        table            x            y            z
##     1.562167 -3369.142899  1712.931276    13.557866
```

We see that the values are roughly the same whether done via matrix algebra or linear models.

```
expect_equal(as.numeric(coef(subset_price)[3:24]), as.numeric(b[3:24]))
```

For some reason the intercept and carat coefficients are in different locations for both lists, so..

```
expect_equal(as.numeric(coef(subset_price)[1]), as.numeric(b[2]))
expect_equal(as.numeric(coef(subset_price)[2]), as.numeric(b[1]))
```

Still good.

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)

z = t(Q) %*% y
b_QR = solve(R) %*% z
```

b is

```
b_QR
```

```
##                             price
## carat                12822.123557
## (Intercept)           5600.059173
```

```
## diamonds$cutGood        208.425227
## diamonds$cutIdeal       679.827969
## diamonds$cutPremium     509.228300
## diamonds$cutVery Good   533.524925
## diamonds$colorE        -126.001616
## diamonds$colorF        -177.850855
## diamonds$colorG        -273.824335
## diamonds$colorH        -928.568116
## diamonds$colorI       -1401.126595
## diamonds$colorJ       -2102.102021
## diamonds$clarityIF     4512.701429
## diamonds$claritySI1    2839.535335
## diamonds$claritySI2    1783.067778
## diamonds$clarityVS1    3850.438614
## diamonds$clarityVS2    3385.120824
## diamonds$clarityVVS1   4017.964967
## diamonds$clarityVVS2   3977.844573
## depth                   -92.158977
## table                     1.562167
## x                     -3369.142899
## y                      1712.931276
## z                        13.557866
```

and it is the same as b from before

```
expect_equal(b_QR, b)
```

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

```
yhat_via_Q = Q %*% t(Q) %*% y
e = y - yhat_via_Q
H = Q %*% t(Q)
```

In one line each, verify that (a) $\hat{y}$ and e sum to the vector y (the prices in the original dataframe),

```
expect_equal(y, yhat_via_Q + e)
```

(b) $\hat{y}$ and e are orthogonal

```
expect_equal(as.vector(t(yhat_via_Q) %*% e), 0, tol = 1e-1)
```

(c) e projected onto the column space of X gets annhilated,

```
expect_equal(sum(H %*% e), 0, tol = 1e-1)
```

(d) $\hat{y}$ projected onto the column space of X is unaffected,

```
expect_equal(H %*% yhat_via_Q, yhat_via_Q)
```

(e) $\hat{y}$ projected onto the orthogonal complement of the column space of X is annhilated

```
expect_equal(sum((diag(nrow(X))-H) %*% yhat_via_Q), 0, tol = 1e-1)
```

(f) the sum of squares residuals plus the sum of squares model equal the original (total) sum of squares

```
ybar = mean(y)
SST = sum((y - ybar)^2)
SSR = sum((yhat_via_Q - ybar)^2)
SSE = sum(e^2)
expect_equal(SST, SSR + SSE)
```

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

```
five_deg_poly = lm(price ~ .*.
                   + poly(carat, 5, raw = TRUE)
                   + poly(depth,5,raw = TRUE) + poly(table, 5, raw = TRUE)
                   + poly(x, 5, raw = TRUE) + poly(y, 5, raw = TRUE)
                   + poly(z, 5, raw = TRUE), diamonds)
```

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

The R^2 value is

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

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

The RMSE is

```
summary(five_deg_poly)$sigma
```

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

and the standard error of the residuals is

```
sd(five_deg_poly$residuals)
```

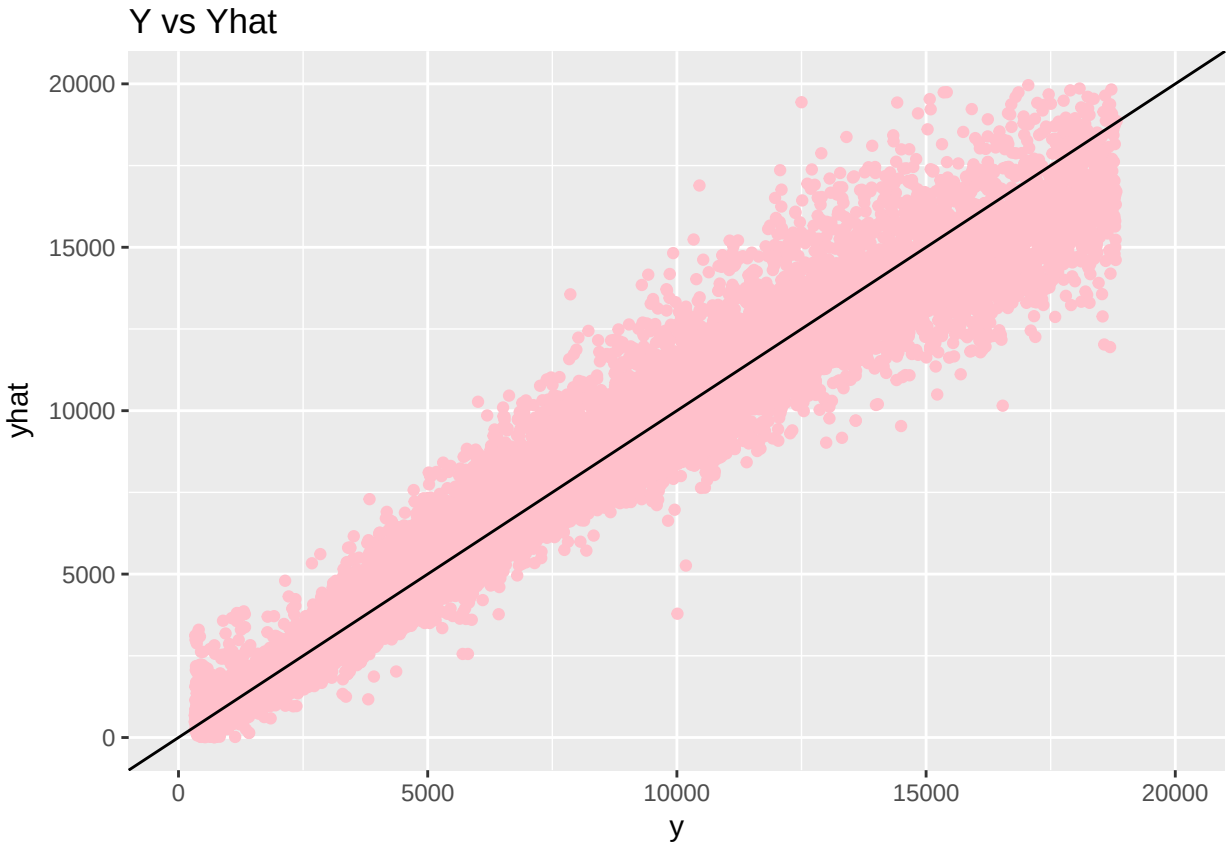
```
## [1] 657.6464
```

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

```
y_vs_yhat = ggplot(diamonds, aes(x = diamonds$price,
                                 y = predict(five_deg_poly, diamonds))) +
            geom_jitter(colour = "pink") +
            xlim(0, 20000) +
            ylim(0,20000) +
            geom_abline(slope = 1) +
            xlab("y") +
            ylab("yhat") +
            ggtitle("Y vs Yhat")
y_vs_yhat
```

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

```
## Warning: Removed 53 rows containing missing values (geom_point).
```

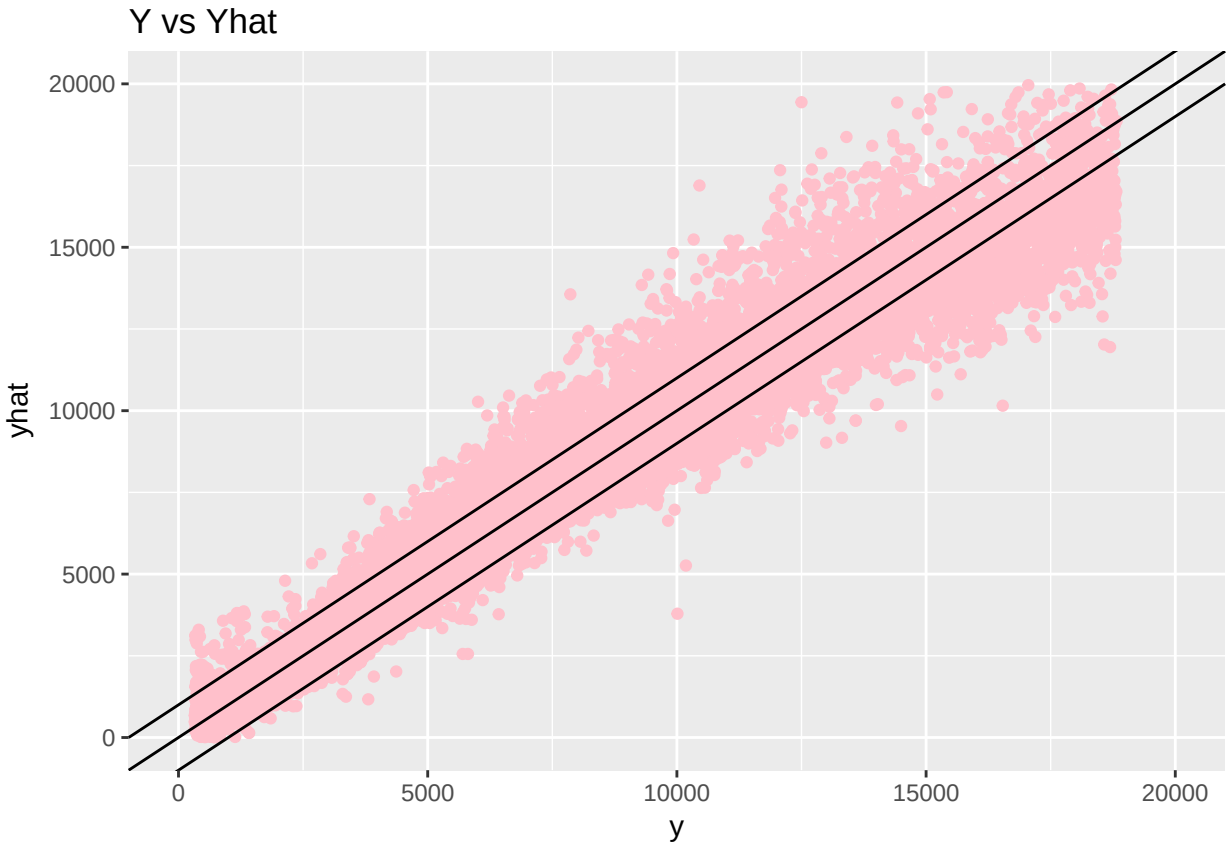


How many diamonds have predictions that are wrong by $1,000 or more ?

```
y_vs_yhat + geom_abline(intercept = -1000) + geom_abline(intercept = 1000)
```

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

```
## Warning: Removed 53 rows containing missing values (geom_point).
```



It is a lot of diamonds, in fact, it is

```
differences = abs(diamonds$price - predict(five_deg_poly, diamonds))
```

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

```
length(which(differences >= 1000))
```

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

diamonds.

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.

Answer: The RMSE is not impressive. Using the empirical formula, it is possible to be off by almost $1300 for the price of a diamond which is not good for a diamond buyer.

What is the degrees of freedom in this model?

```
length(coef(five_deg_poly))
```

```
## [1] 265
```

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

Answer: g is close to h^* in this model because there are more degrees of freedom.

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

Answer: g is not close to f in this model because g does not contain all the features that can capture the true phenomenon f.

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

Answer: To make g closer to f, more polynomial terms of varying degrees can be added to this model.

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.

Answer: Information that can help diminish ignorance of relevant information includes: crown angle, crown depth, culet size/angle, place of purchase and authenticity.

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

```
n = nrow(diamonds)
K = 10
test_indices = sample(1 : n, size = (n / K))
train_indices = setdiff(1 : n, test_indices)
test_data = diamonds[test_indices, ]
train_data = diamonds[train_indices, ]

poly_fit_test = lm(price ~ .*. + poly(carat, 5, raw = TRUE)
                   + poly(depth,5,raw = TRUE) + poly(table, 5, raw = TRUE)
                   + poly(x, 5, raw = TRUE) + poly(y, 5, raw = TRUE)
                   + poly(z, 5, raw = TRUE), train_data)

y_hat_oos = predict(poly_fit_test, test_data)
```

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

```
oos_resids = test_data$price - y_hat_oos
sd(oos_resids)
```

```
## [1] 652.8752
```

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?

Answer: The oos standard error of the residuals is a bit greater than the standard error of the residuals of the in-sample estimation. However I do not think there is overfitting.

Extra-credit: validate the model via cross validation. Note: The following block of code is time-expensive.

```
n = nrow(diamonds)
K = 10
size = n/K
cv_resids = c()

for(i in 1:K){
  init = (i-1)*size + 1
  final = i*size
  test_indices = c(seq(init:final))
  train_indices = setdiff(1:n, test_indices)
  X_train = diamonds[train_indices, ]
  X_test = diamonds[test_indices, ]
  poly_fit_test = lm(price ~ .*. + poly(carat, 5, raw = TRUE)
                     + poly(depth,5,raw = TRUE) + poly(table, 5, raw = TRUE)
                     + poly(x, 5, raw = TRUE) + poly(y, 5, raw = TRUE)
                     + poly(z, 5, raw = TRUE), X_train)
```

```
  y_hat_oos = predict(poly_fit_test, X_test)
  oos_resids = X_test$price - y_hat_oos
  cv_resids[i] = sd(oos_resids)
}
```

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

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

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

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

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

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

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

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

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

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

```
cv_resids
```

```
##  [1] 528.5943 528.5943 528.5943 528.5943 528.5943 528.5943 528.5943
##  [8] 528.5943 528.5943 528.5943
```

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

Answer: This result is different from the one in the single validation case. There is no overfitting in this model.

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:

```
p = 2
sds = c(5)

X = matrix(c(x, rnorm(n*p)), ncol = 1 + p)
test_indices = sample(1:n, (1/10)*n)
train_indices = setdiff(1:n, test_indices)

for(i in 2:n){
  p = p+1
  X = matrix(c(x, rnorm(n*p)), ncol = 1 + p)
  X_test = X[test_indices,]
  X_train = X[train_indices,]
  Y_test = y[test_indices]
  Y_train = y[train_indices]
  mod = lm(Y_train ~ X_train)
  y_hat_oos = predict(mod, data.frame(X_test))
  sds[i] = sd(Y_test - y_hat_oos)
}
```

```
## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows
```

```
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
```

```
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning: 'newdata' had 10 rows but variables found have 90 rows
```

```
## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows

## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning in predict.lm(mod, data.frame(X_test)): prediction from a rank-
## deficient fit may be misleading

## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning in predict.lm(mod, data.frame(X_test)): prediction from a rank-
## deficient fit may be misleading

## Warning: 'newdata' had 10 rows but variables found have 90 rows
## Warning in predict.lm(mod, data.frame(X_test)): prediction from a rank-
```

```
## deficient fit may be misleading

## Warning: 'newdata' had 10 rows but variables found have 90 rows

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

## Warning: 'newdata' had 10 rows but variables found have 90 rows

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

## Warning: 'newdata' had 10 rows but variables found have 90 rows

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

## Warning: 'newdata' had 10 rows but variables found have 90 rows

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

## Warning: 'newdata' had 10 rows but variables found have 90 rows

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

## Warning: 'newdata' had 10 rows but variables found have 90 rows

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

## Warning: 'newdata' had 10 rows but variables found have 90 rows

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

## Warning: 'newdata' had 10 rows but variables found have 90 rows

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

## Warning: 'newdata' had 10 rows but variables found have 90 rows

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

## Warning: 'newdata' had 10 rows but variables found have 90 rows

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

Overfit occurs when the number of fake predictors becomes

```r
which.min(sds)+1
```

```
## [1] 33
```