HW03p

Joseph Peltroche

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`.

```
pacman::p_load(ggplot2, quantreg)
pacman::p_load(forcats, lazyeval, ggthemes)
```

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
```

```
## starting httpd help server ... done
```

```
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))
```

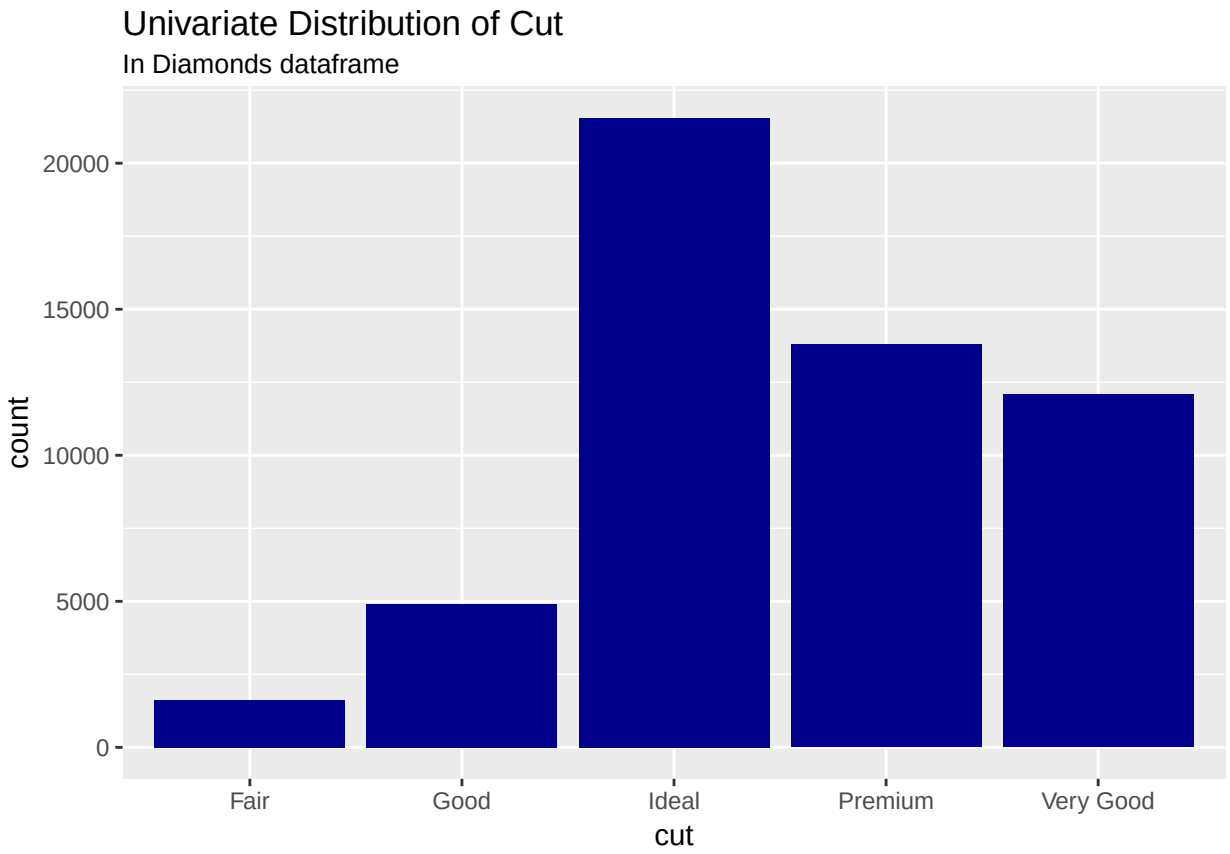
What is n, p, what do the features mean, what is the most likely response metric and why?

In this data set n is the number of diamonds, in this case it is 53940 diamonds and p the number of features or the dimension of each observation, i.e., $p = 10$. The features describe a different attribute of the diamond, beginning with simple concepts such as price (in US dollars) and weight of the diamond and extends to more complicated metrics such as the total depth percentage and the table. As shown by the str() function, the most likely response metric is numeric since features such as weight, dimensions of the diamond, price, depth, and table are defined on a continuum and outnumber the other features.

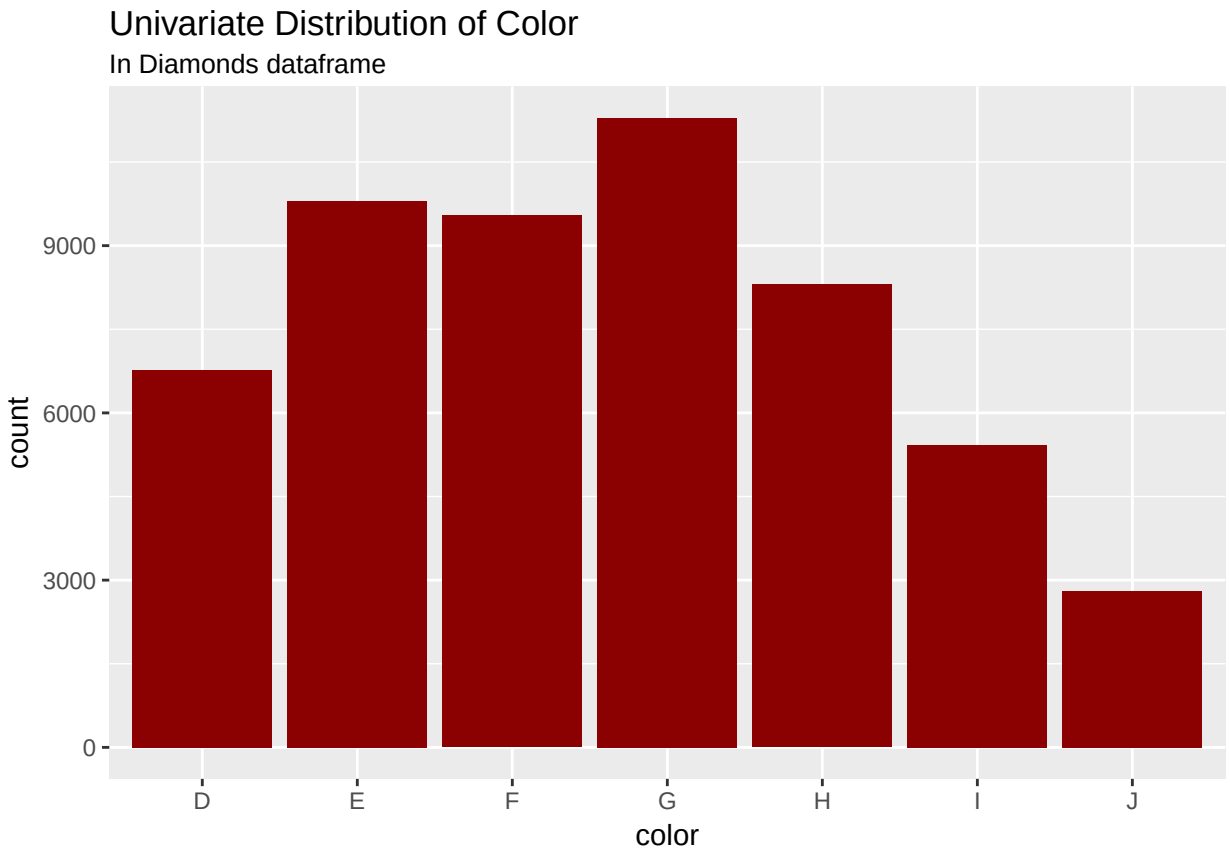
Regardless of what you wrote above, the variable `price` will be the response variable going forward.

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

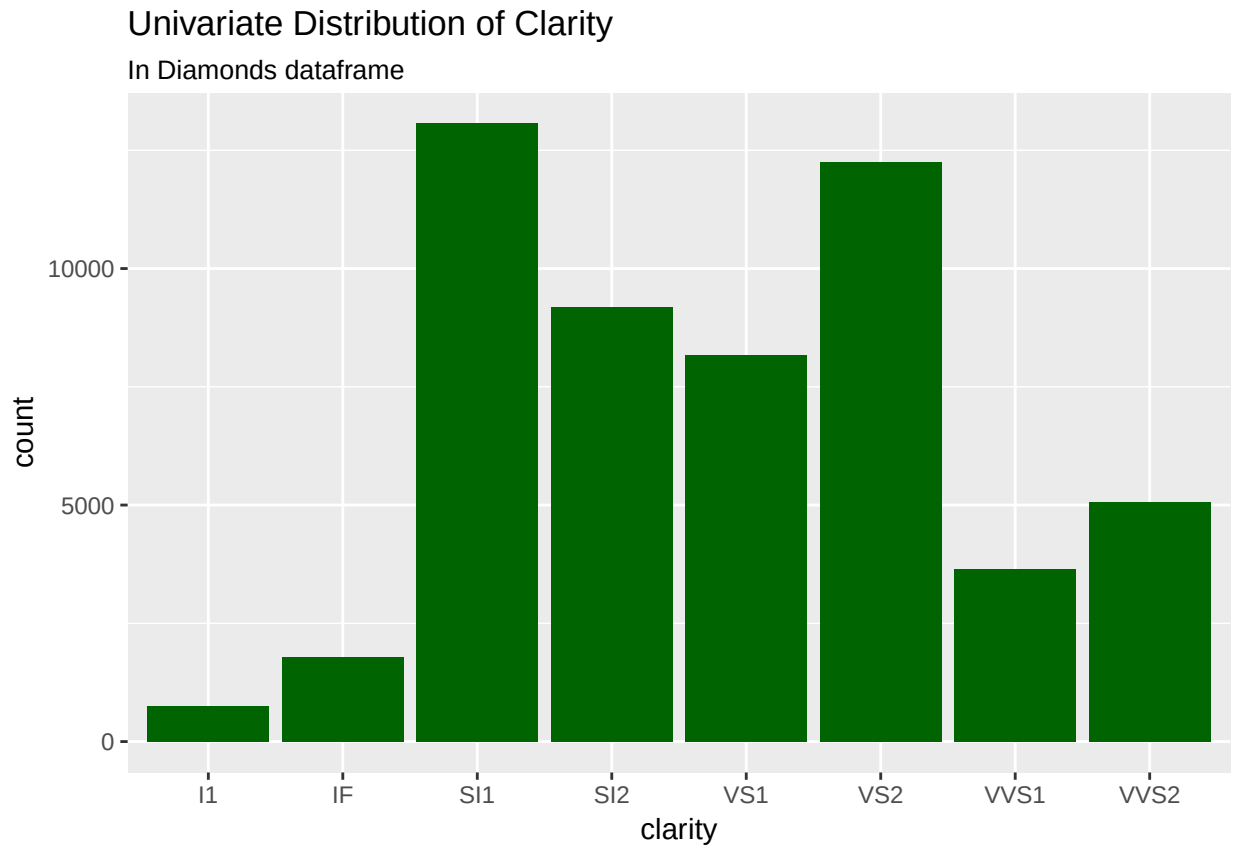
```
ggplot(diamonds) + geom_bar(aes(cut), fill="darkblue")+ggtitle("Univariate Distribution of Cut", subtitl
```




```
ggplot(diamonds) + geom_bar(aes(color), fill="darkred")+ggtitle("Univariate Distribution of Color", sub
```

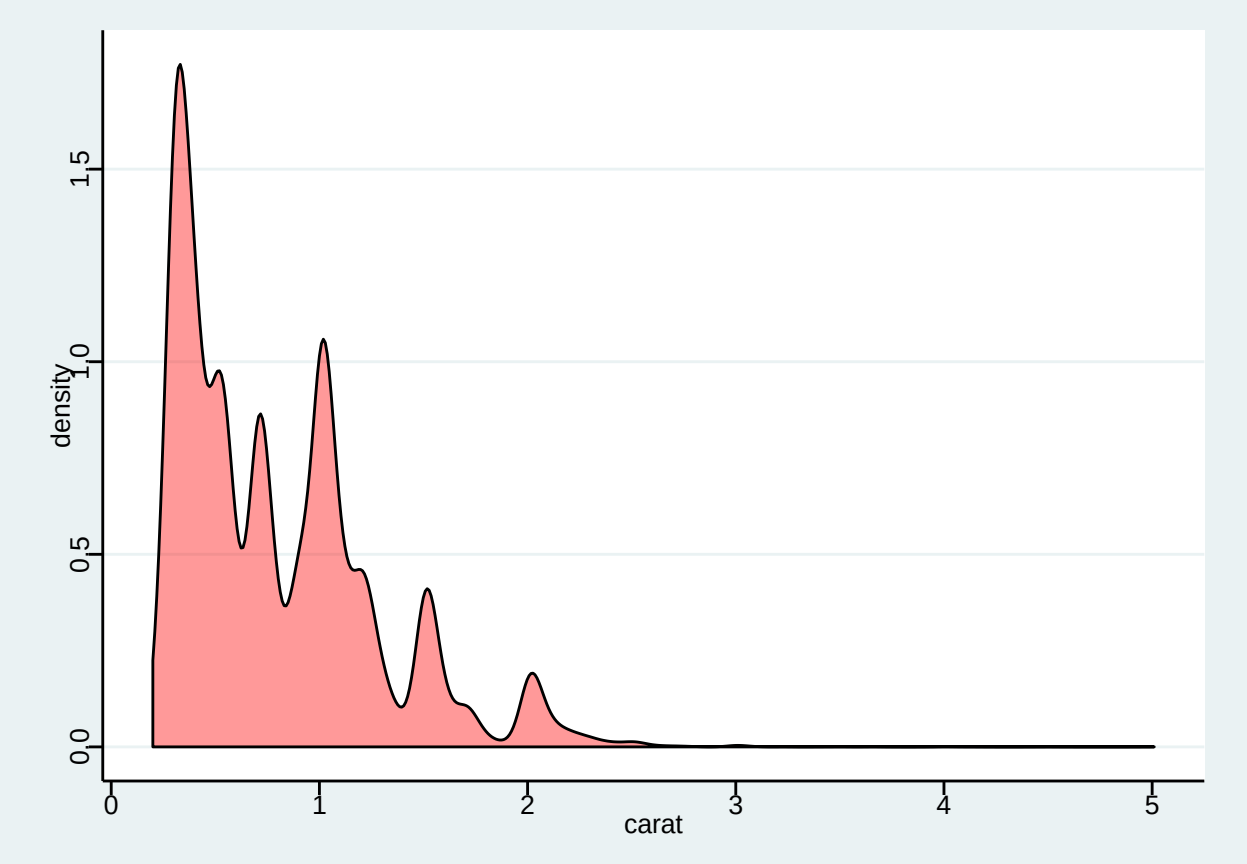



```
ggplot(diamonds) + geom_bar(aes(clarity), fill="darkgreen")+ggtitle("Univariate Distribution of Clarity
```

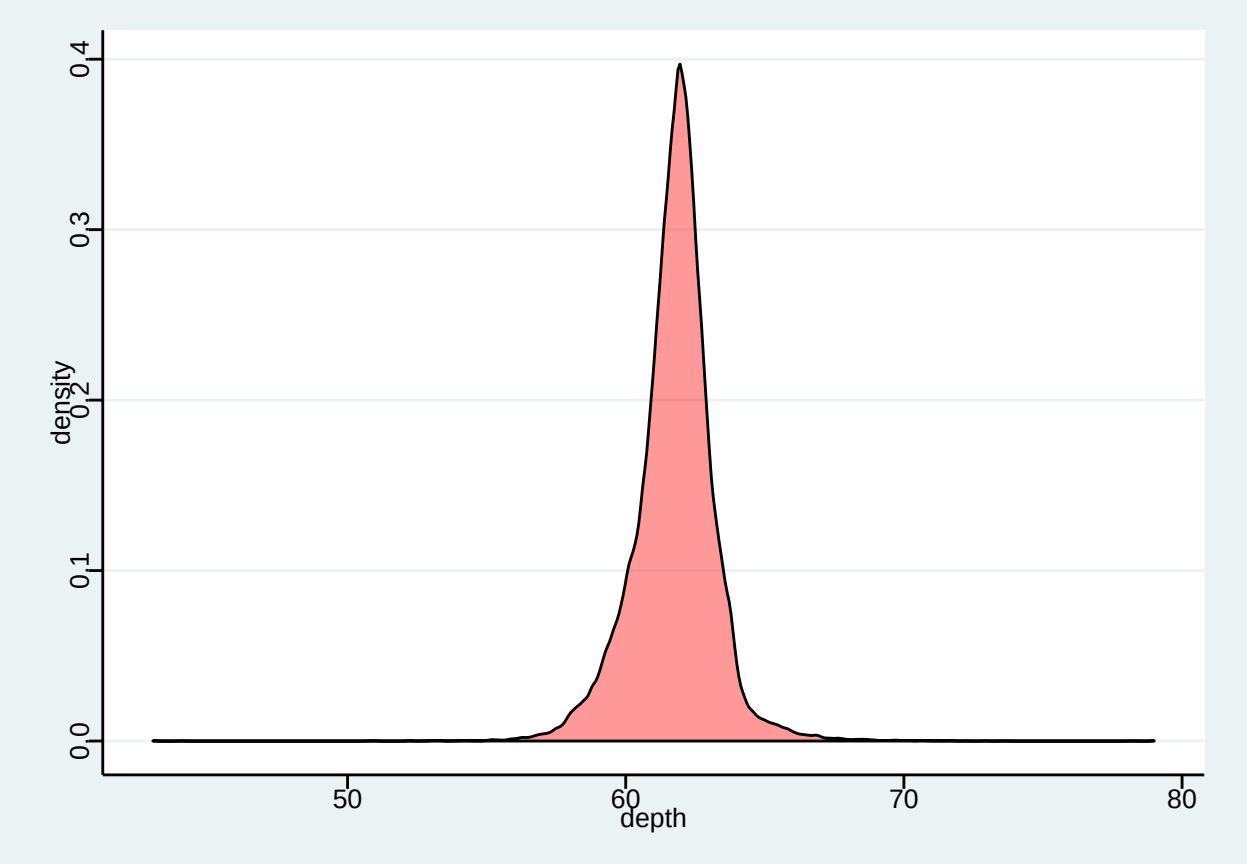



```
v=c(1,5:10)
for(i in v){
  gg_plot_all_predictors= ggplot(diamonds,aes(diamonds[i]))+geom_density(fill="red", alpha=0.4)+theme_s
  plot(gg_plot_all_predictors)
}
```

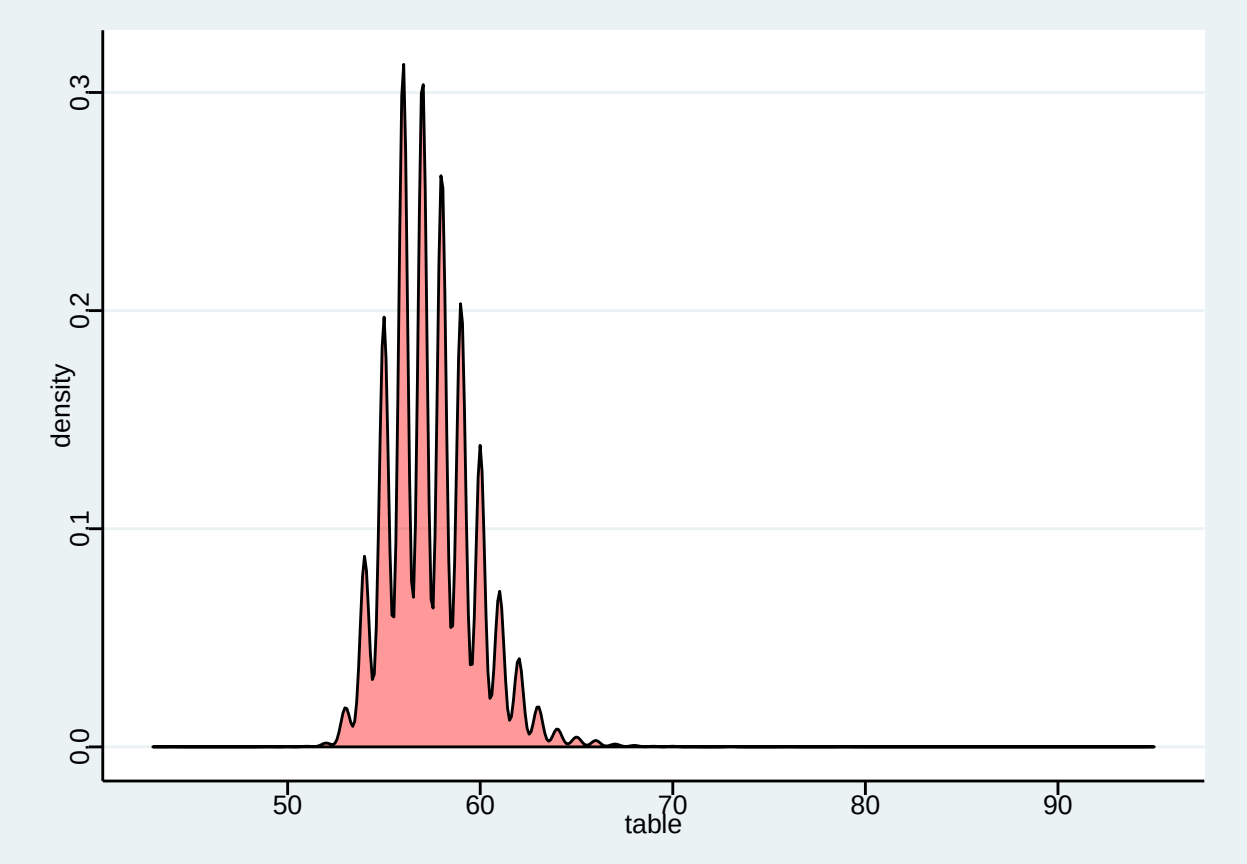
```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to c
```



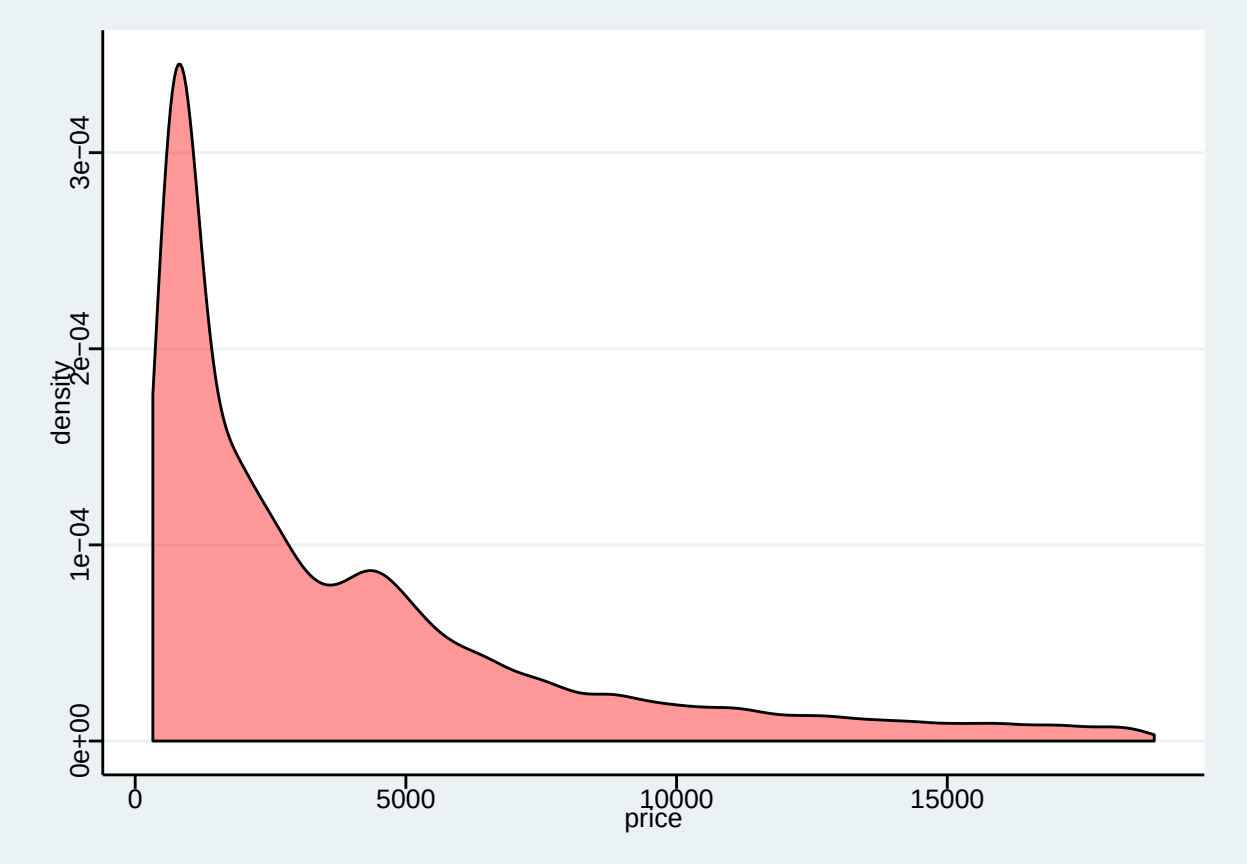

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```



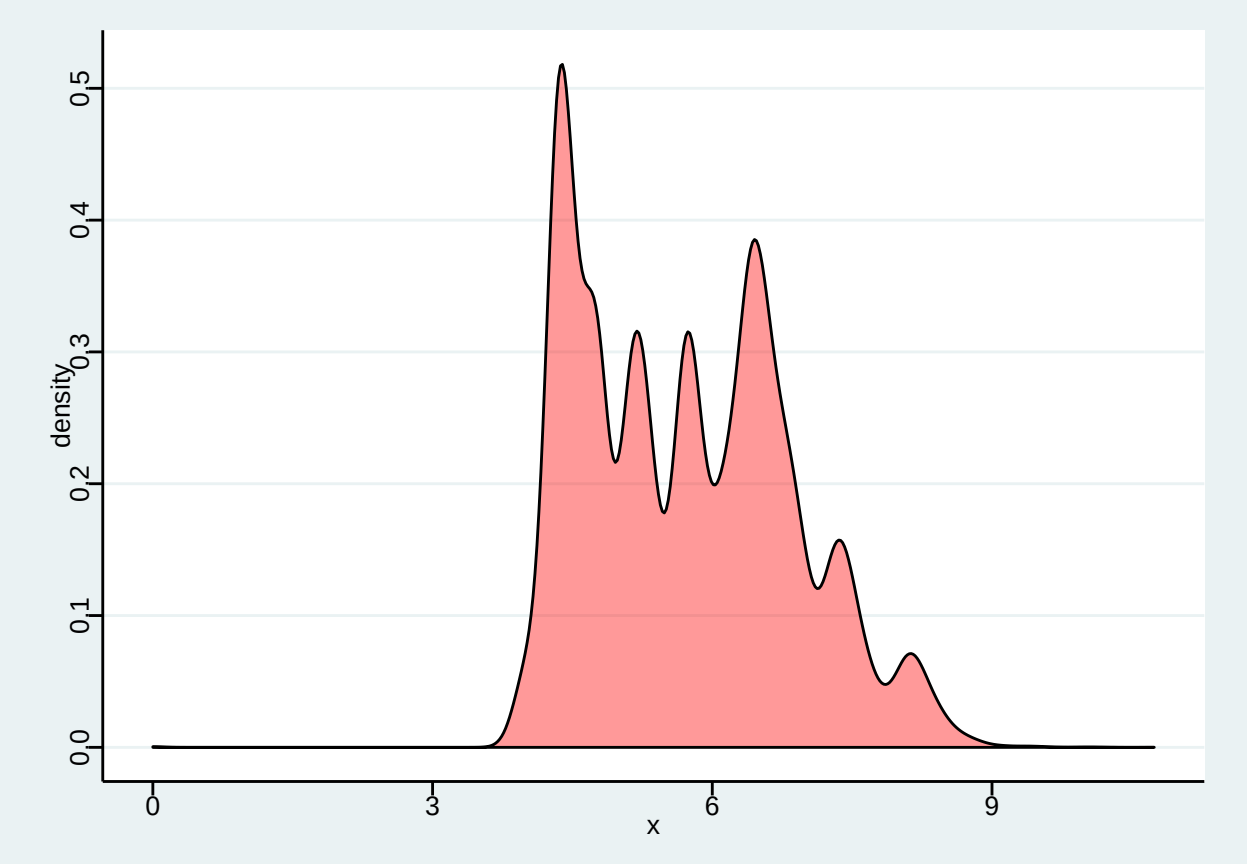

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```



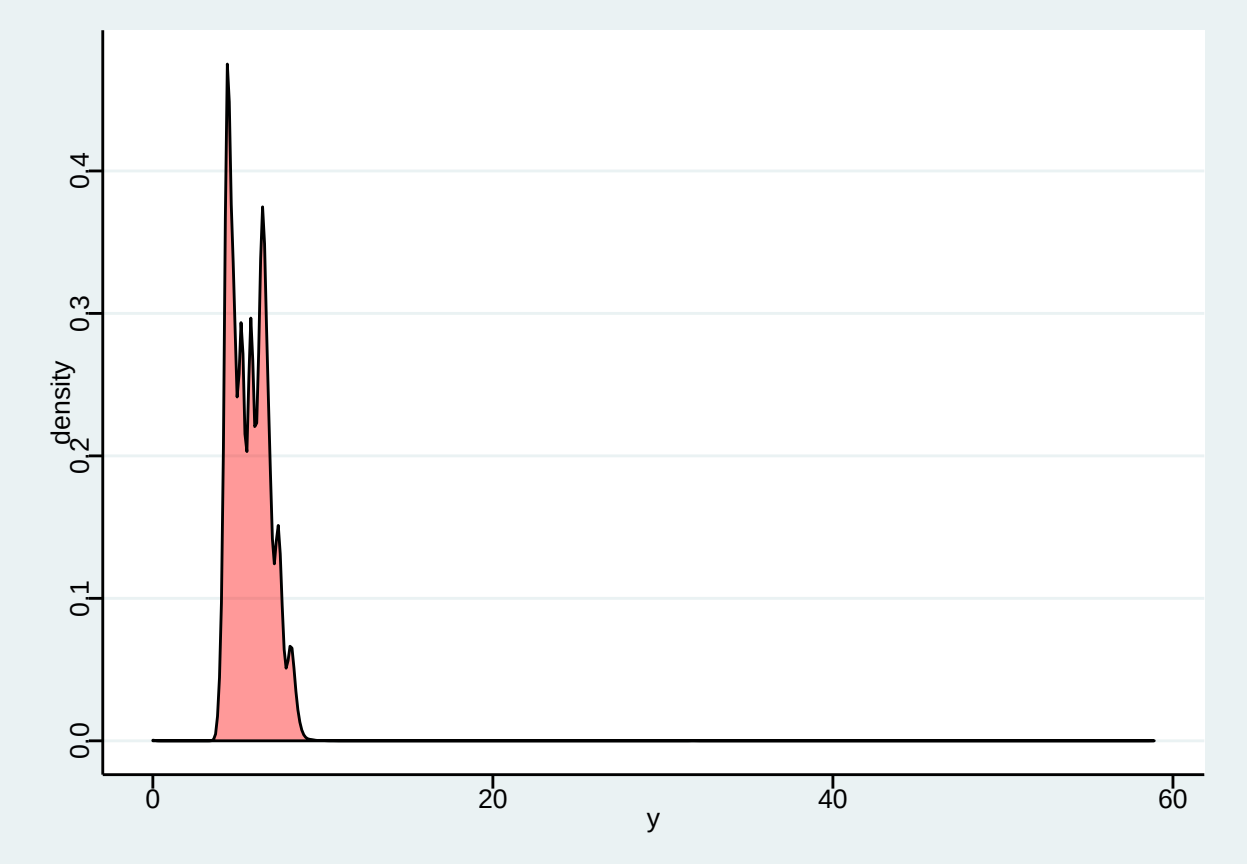

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```



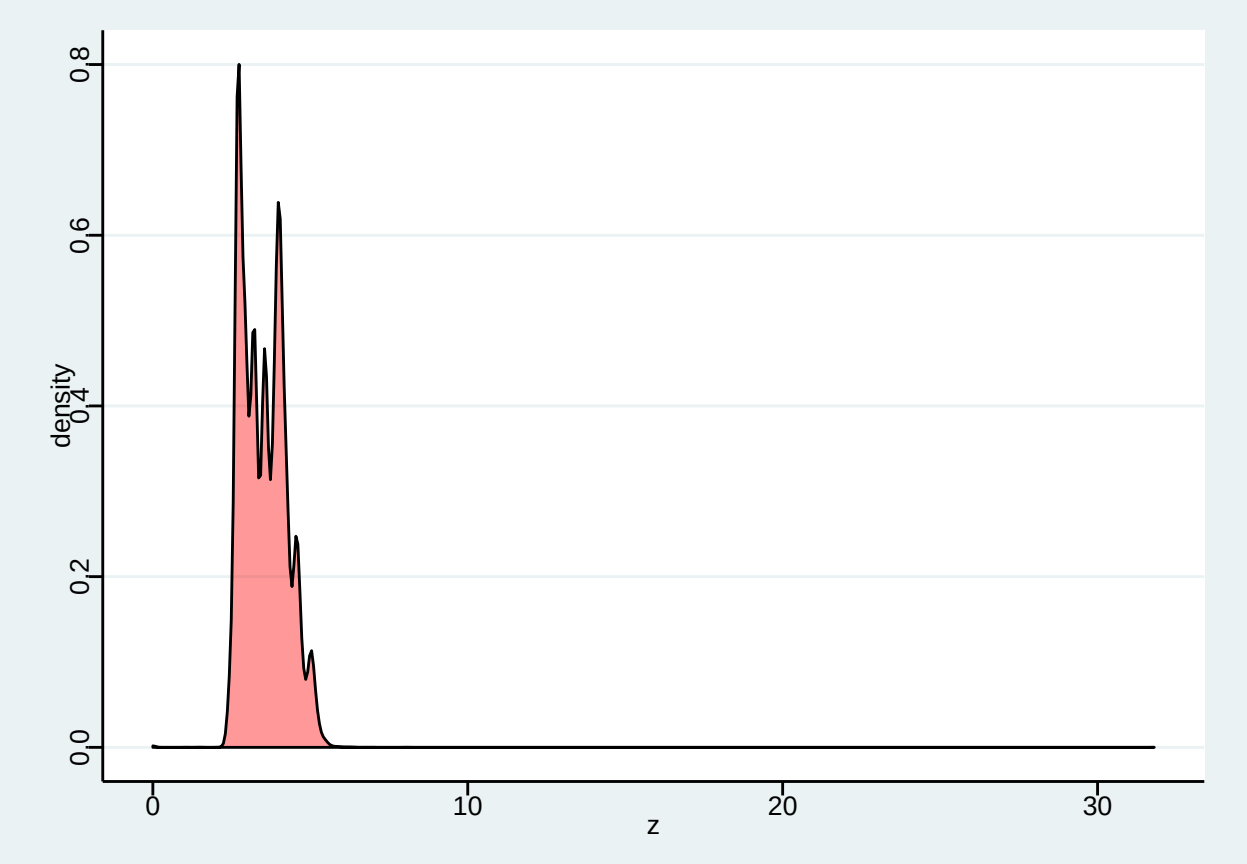

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```




```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```



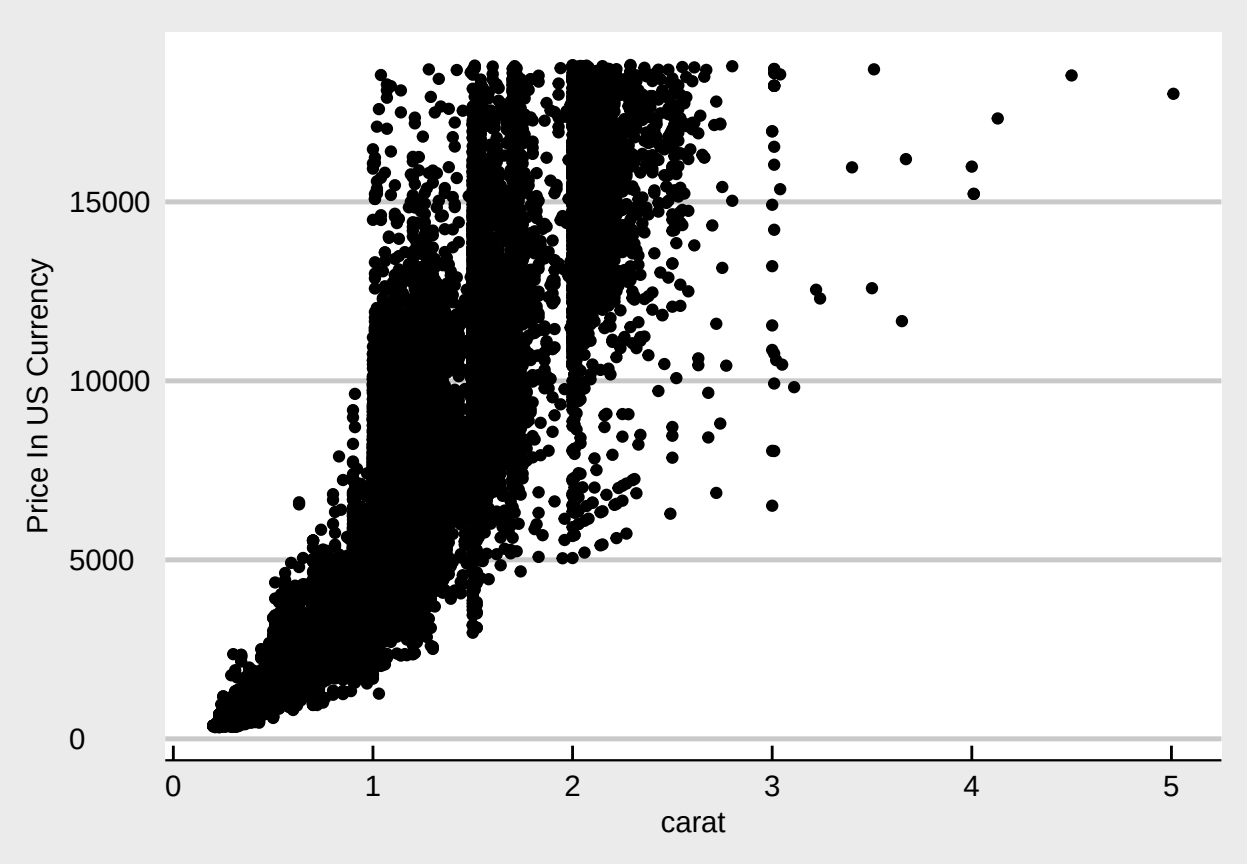

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```

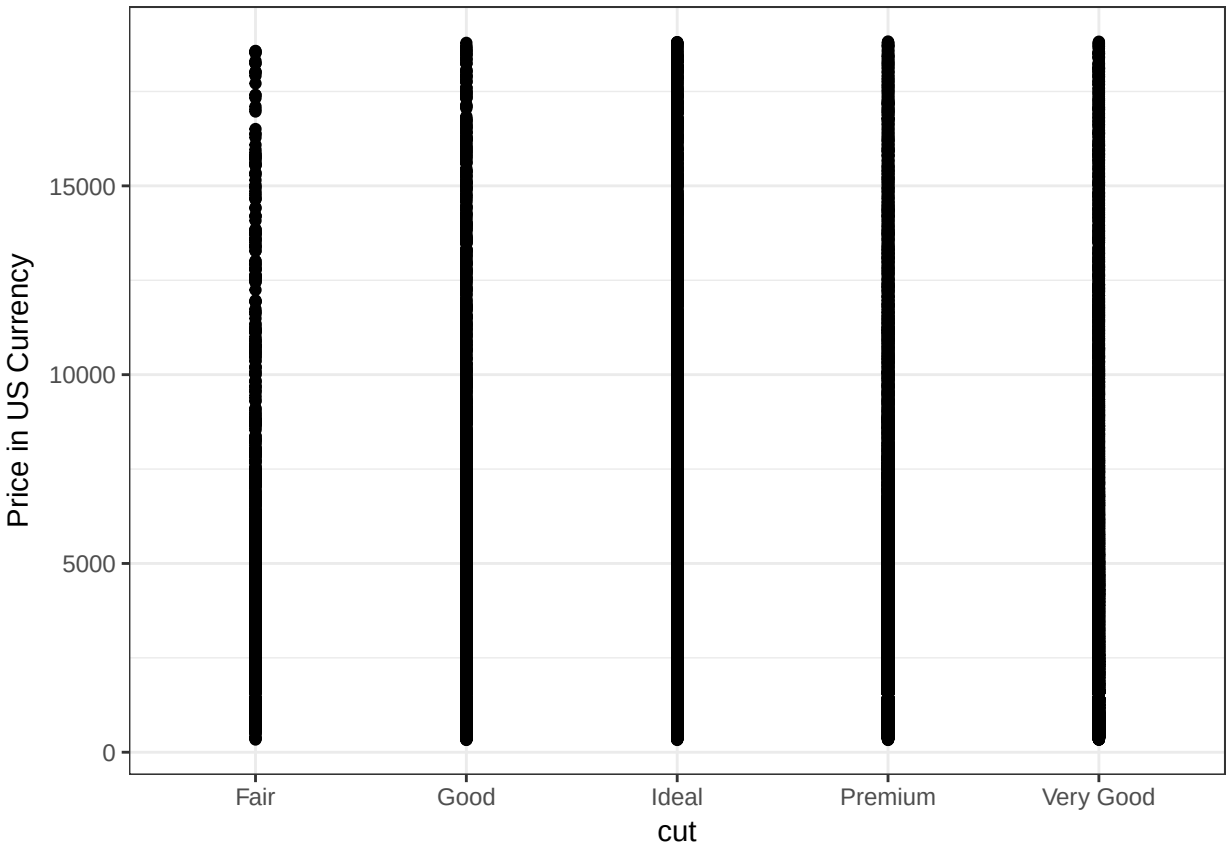


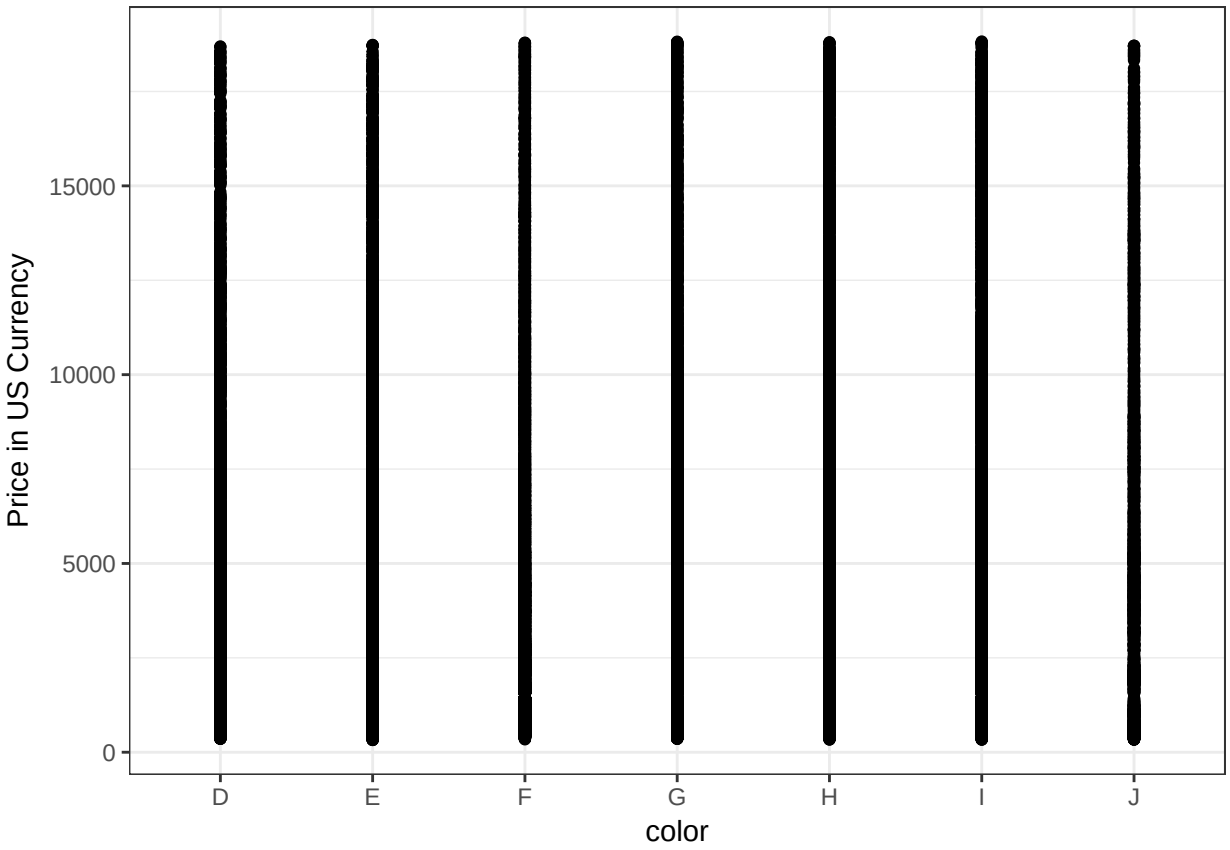
Use `ggplot` to look at the bivariate distributions of the response versus *all* predictors. Make sure you handle categorical predictors differently from continuous predictors. This time employ a for loop when an logic that handles the predictor type.

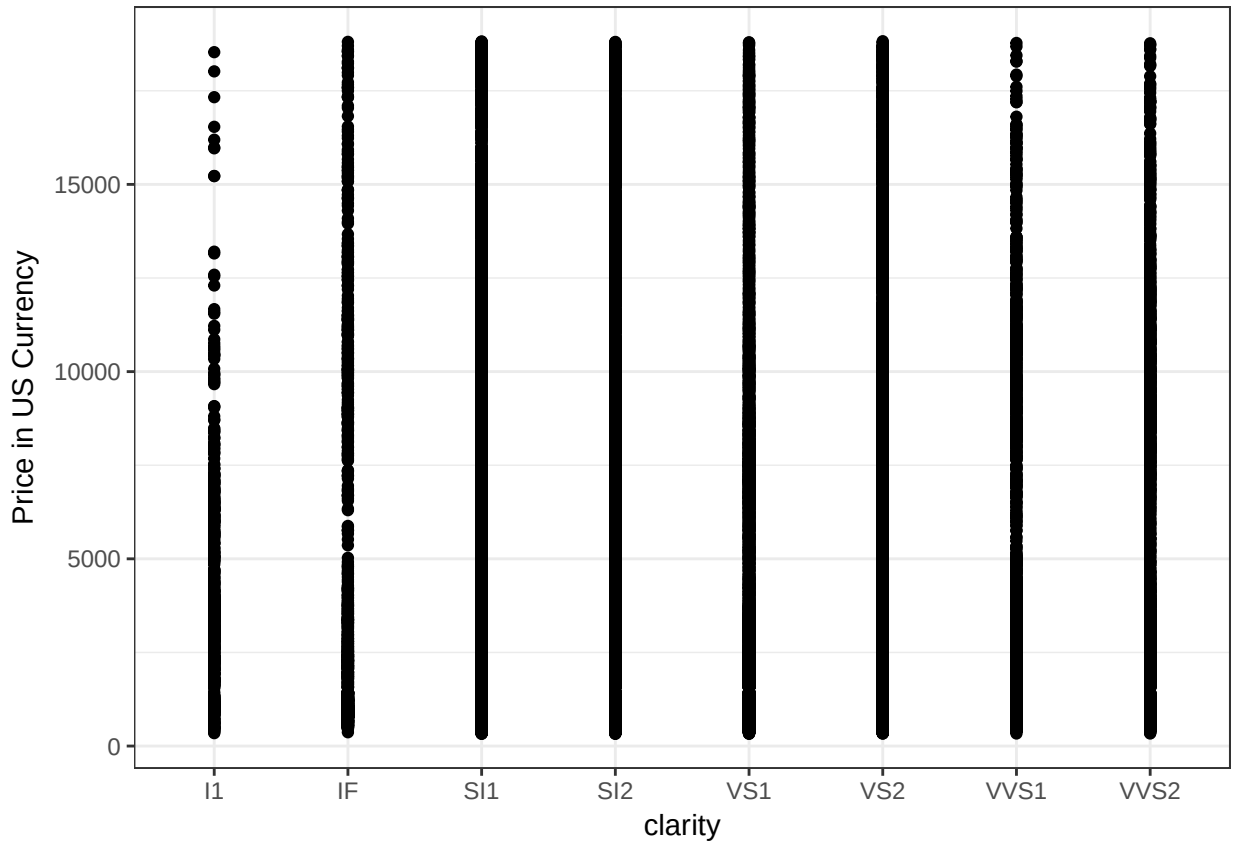
```
for(i in 1:ncol(diamonds)){
  if (is.null(levels(diamonds[[i]]))==TRUE){
    gg_continuous=ggplot(diamonds,aes(x = diamonds[i], y = diamonds[7])) + geom_point()+xlab(colnames(di
    plot(gg_continuous);
  }else{
    gg_discrete = ggplot(diamonds, aes(diamonds[[i]],diamonds[[7]])) + geom_point() + xlab(colnames(diam
    plot(gg_discrete);
  }
}
```

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```

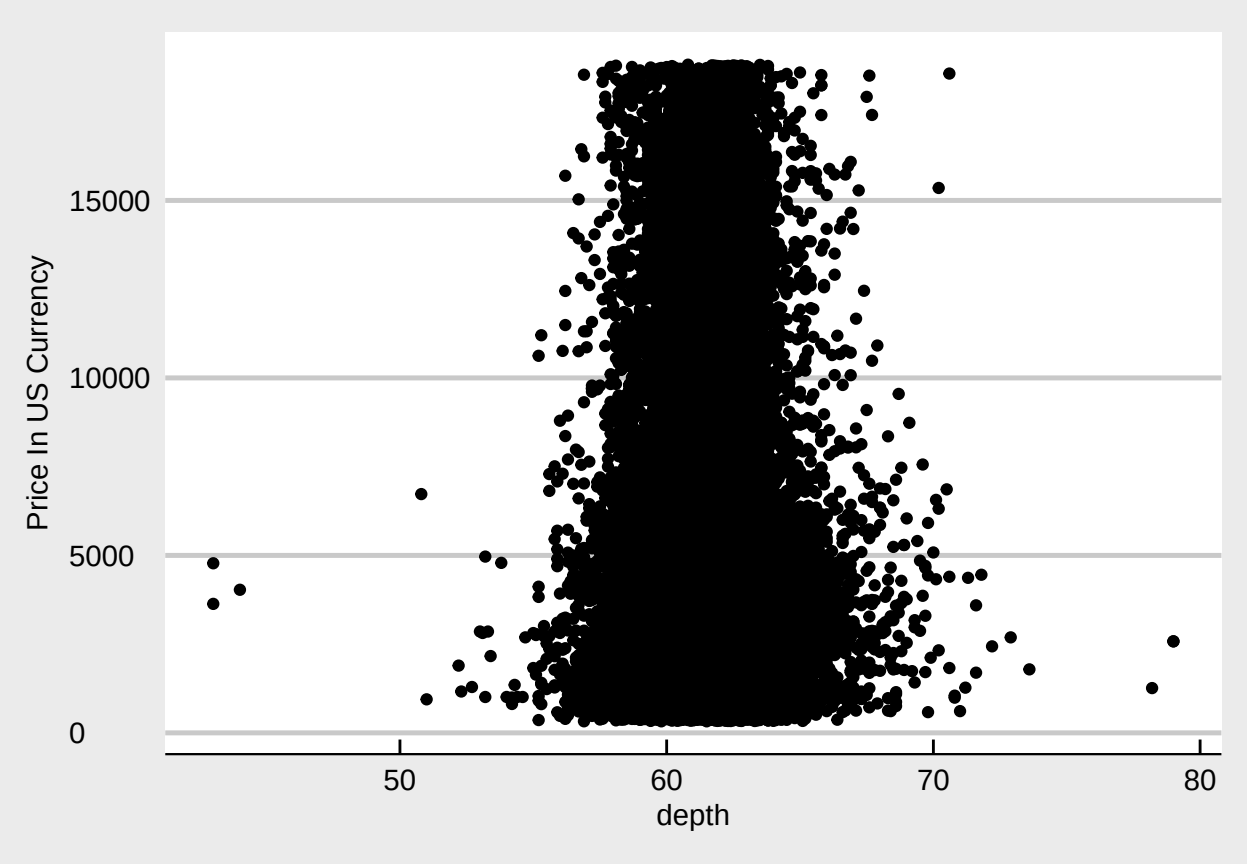

Price In US Currency
15000
10000
5000
0
0
1
2
3
4
5
carat


Price in US Currency
15000
10000
5000
0
Fair
Good
Ideal
Premium
Very Good
cut

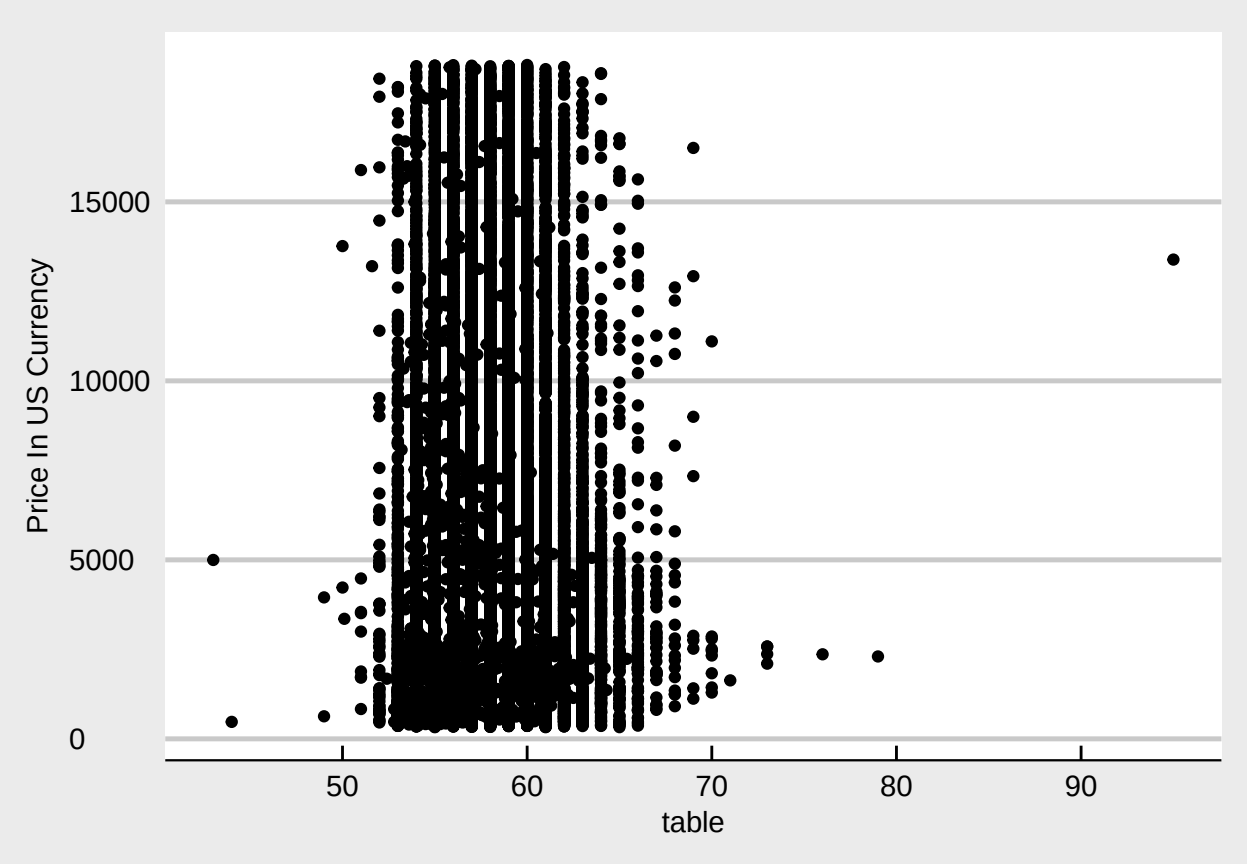

Price in US Currency
15000
10000
5000
0
D
E
F
G
H
I
J
color



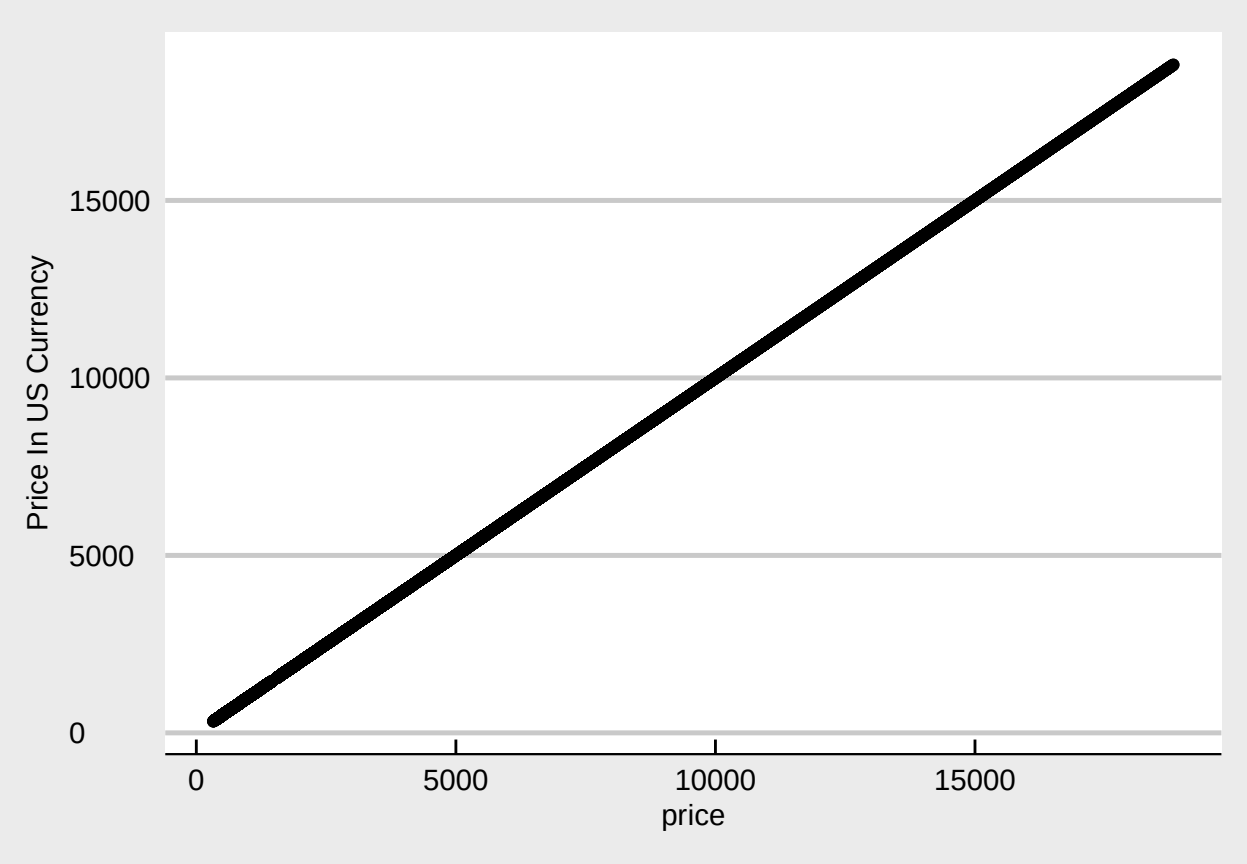

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```



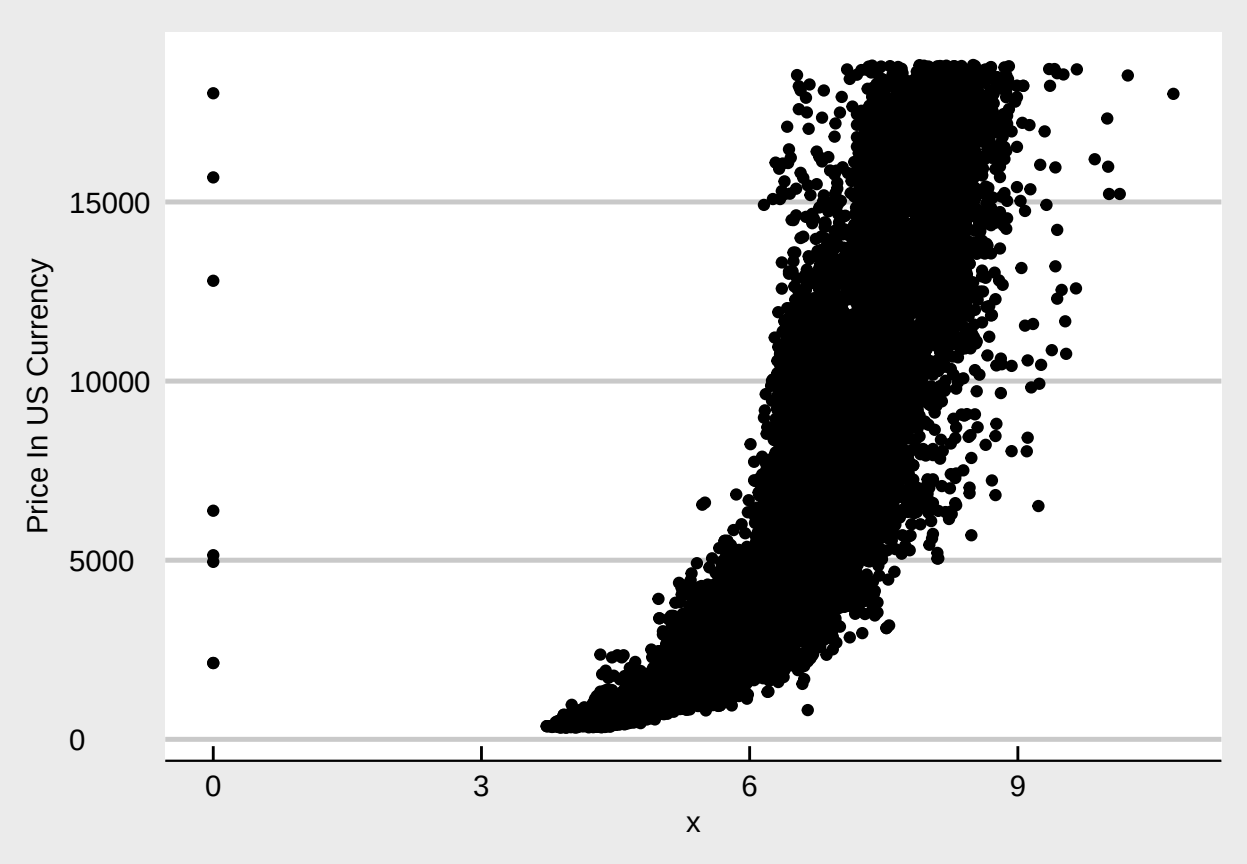

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```



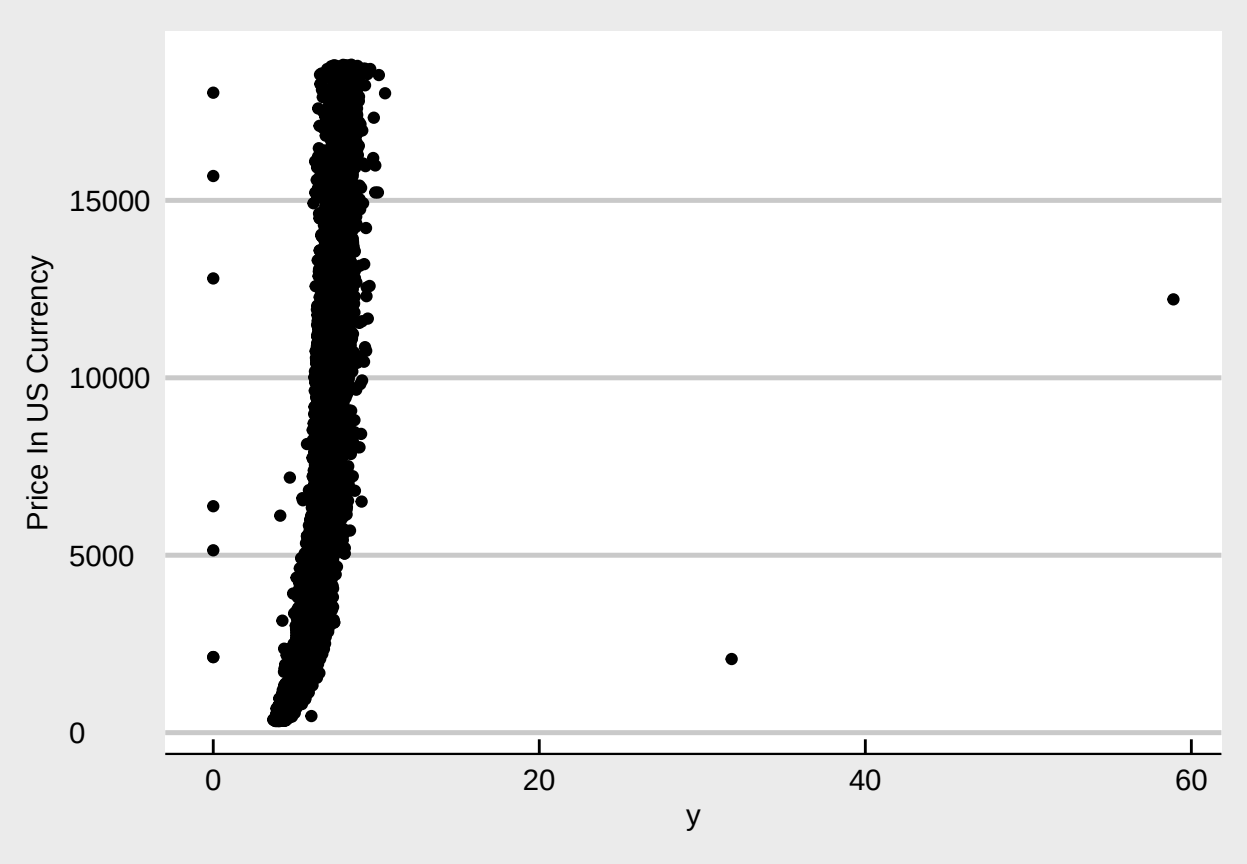

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```



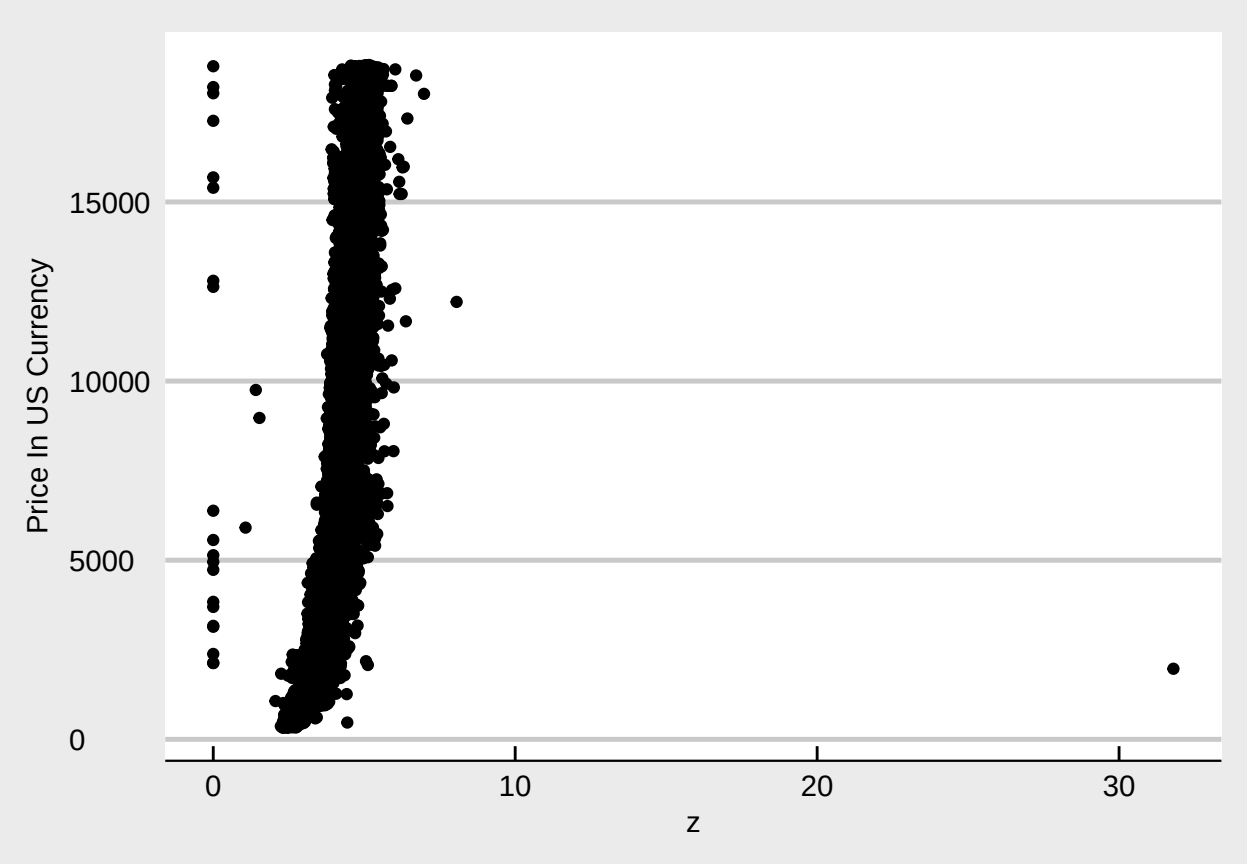

```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```




```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```




```
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
## Don't know how to automatically pick scale for object of type tbl_df/tbl/data.frame. Defaulting to co
```

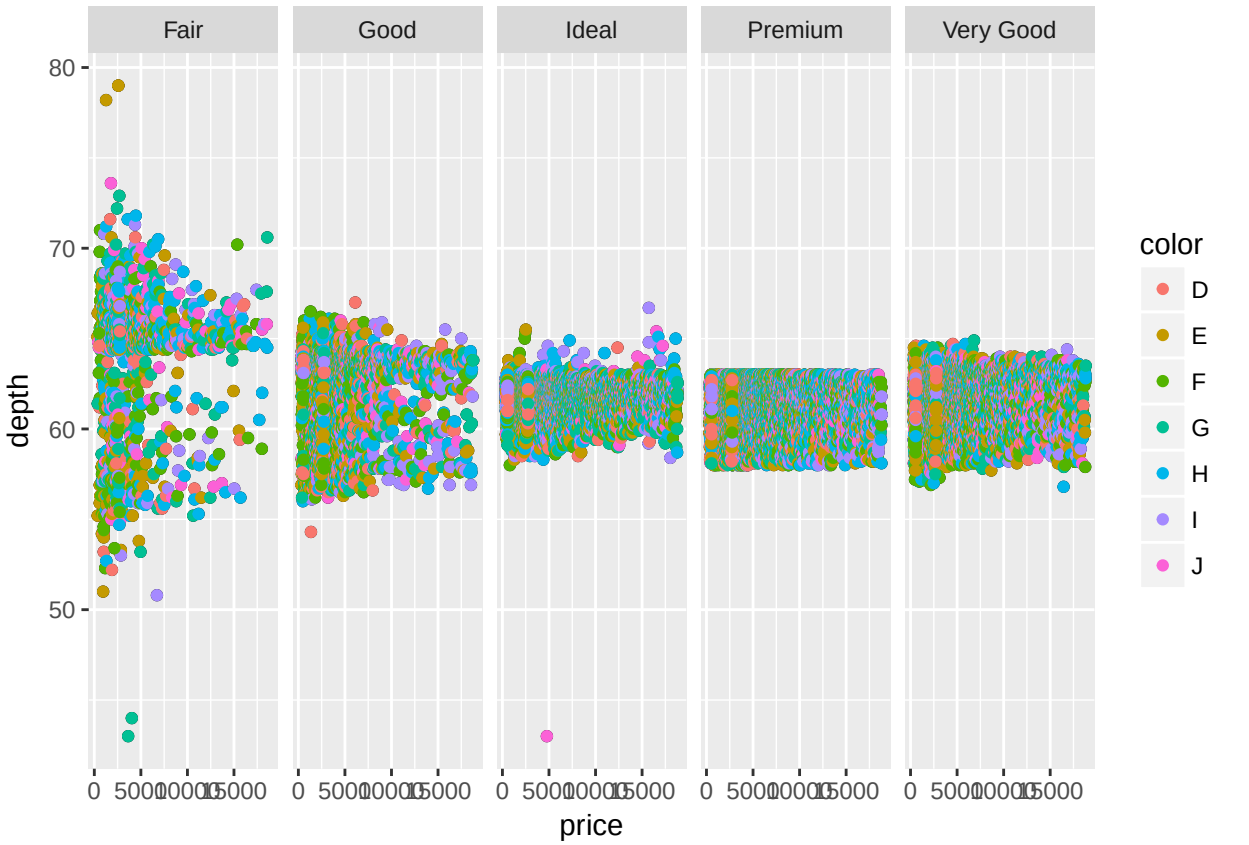


Does depth appear to be mostly independent of price?

For the most part, the depth predictor seems to be independent of price. Any attempt at a function relating price to depth would fail the vertical line test, indicating depth's independence of price.

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

```
depth=diamonds$depth
price = diamonds$price
cut = diamonds$cut
color = diamonds$color
base_plot_3p = ggplot(diamonds, aes(price, depth))
#base_plot_3p + geom_point() +facet_grid(.~cut)
#base_plot_3p + geom_point(aes(col = color))
base_plot_3p + geom_point()+ facet_grid(. ~ cut) + geom_point(aes(col = color))
```




```
#wasn't sure whether you were asking for two separate graphs with cut (using faceting) and color (via c
```

Does diamond color appear to be independent of diamond depth?

Diamond color appears to be independent of diamond depth, since the there is clutter of colors that dominate any part of the graph. Rather it seems to be randomly distributed.

Does diamond cut appear to be independent of diamond depth?

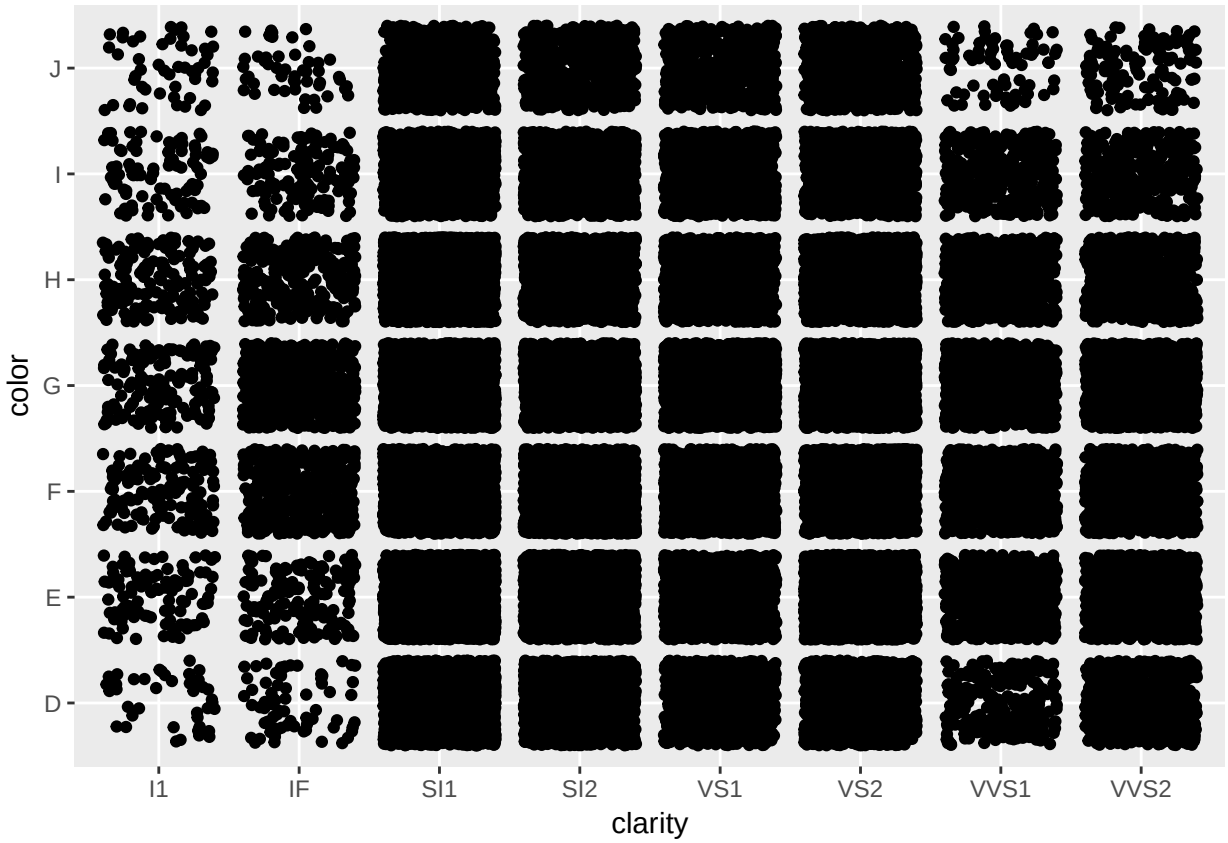
Diamond cut appears to be independent of diamond depth for the most part. Albeit the concentration of each graph gradually increases with every passing cut "level", the diamond depth and diamond cut depict a lack of a function (failure by vertical line test).

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

These plots do not allow me to easily assess the dependence (or lack of a dependence) between diamond price and diamond cut. These plots allow me to interpret relationships between other predictors, but to me diamond cut and diamond price are not so easily readable.

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.

```
clarity = diamonds$clarity
ggplot(diamonds, aes(clarity, color))+geom_jitter()
```



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

For the most part, there appears to be a indendence between diamond color and diamond clarity. The plot above depicts what seems to be a sort of random distribution.

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

```
lsq_price_depth = lm(price~depth, diamonds)
b=coef(lsq_price_depth)
#mp=mean(diamonds$price)
#ggplot(diamonds, aes(depth,price)) + geom_point() + geom_abline(intercept = b[1], slope = b[2], col =
```

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

```
price= diamonds$price
summary(lsq_price_depth)$coefficients
```

```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 5763.66772   740.5563  7.782889 7.214180e-15
## depth        -29.64997    11.9897 -2.472953 1.340325e-02
```

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

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

```
y_hat = b[1]+b[2]*depth
SSE=(price-y_hat)%*%(price-y_hat)
MSE=SSE/(length(price)-2)
RMSE=sqrt(MSE)
RMSE
```

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

```
sd(price-y_hat)
```

```
## [1] 3989.214
```

```
mp
```

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

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

The plot depicts a nearly tacit independence of price on depth, so the linear regression cannot do much better than the null model which is simply the average of the prices plotted in green above (independent of any other variables). Therefore, these metrics are expected since they depict a linear relationship close to the average. This can be seen with such a small R^2 (implying a small difference between the sample variances of the null model and the errors of the model in question), a revelantly small slope (described by the coefficients), and a substantial RMSE (inversely proportional to R^2).

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

```
lsq_price_carat = lm(price ~ carat, diamonds)
b2=coef(lsq_price_carat)
#ggplot(diamonds, aes(carat, price)) + geom_point() + geom_abline( intercept = b2[1], slope = b2[2], co
```

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

```
summary(lsq_price_carat)$coefficients
```

```
##              Estimate Std. Error   t value Pr(>|t|)
## (Intercept) -2256.361   13.05535 -172.8304        0
## carat        7756.426   14.06658  551.4081        0
```

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

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

```
y_hat= b2[1]+b2[2]*carat
```

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

```
SSE_c = (price-y_hat)%*%(price-y_hat)
MSE_c=SSE_c/(length(price)-2)
RMSE_c=sqrt(MSE_c)
RMSE_c
```

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

```
sd(price-y_hat)
```

```
## [1] 3989.214
```

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

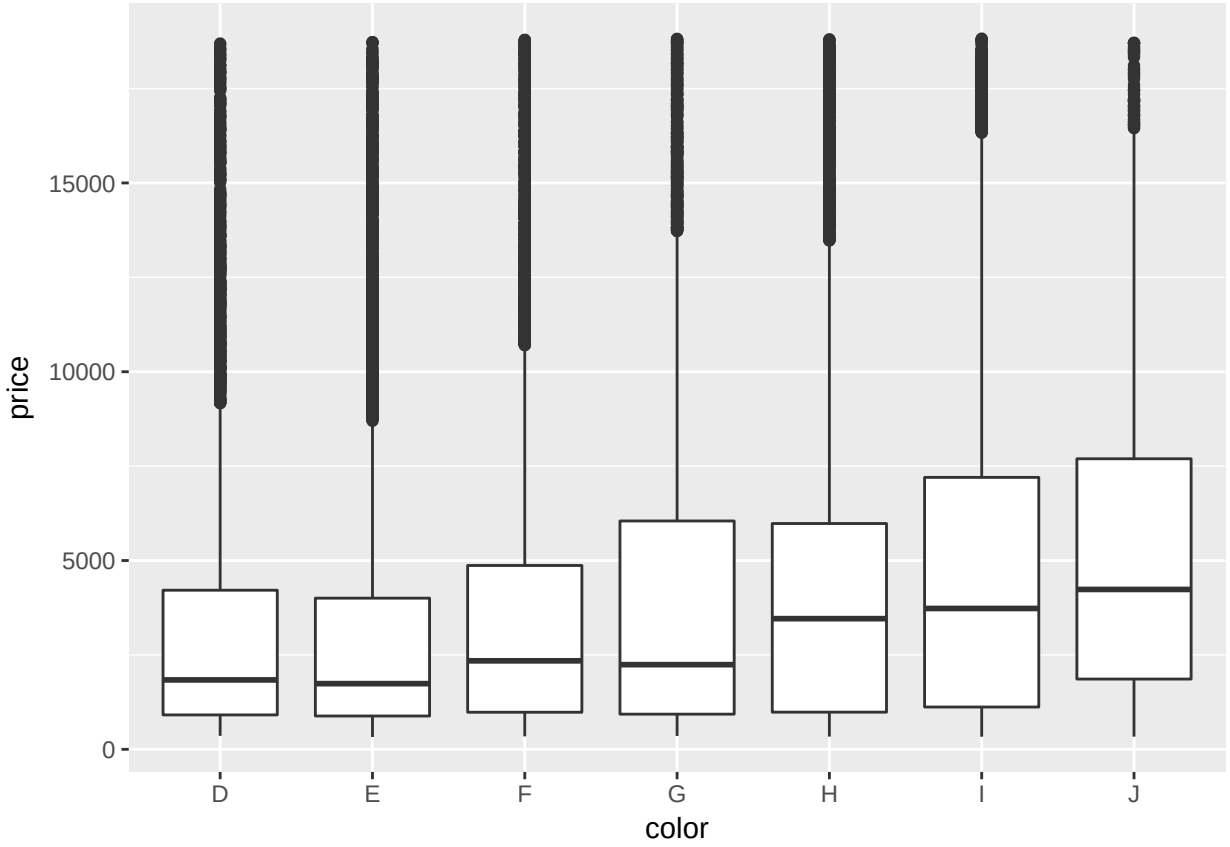
These metrics are very appropiate given that visulalization above, the R^2 is much bigger, and the graph indicates there is a much greater difference between the linear regression and the null model. The R^2 should be close to 1, and it follows the RMSE should be smaller than before, which is shown in the stats above. The coefficents depict a greater slope, as shown in the graph.

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

```
anova_colmod = lm( price ~ color, diamonds)
b3= coef(anova_colmod)
b3
```

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

```
ggplot(diamonds, aes(color, price)) + geom_boxplot()
```



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

```
price=diamonds$price
summary(anova_colmod)$coefficients
```

```
##               Estimate Std. Error   t value      Pr(>|t|)
## (Intercept) 3169.95410   47.70694 66.446391  0.000000e+00
## colorE       -93.20162   62.04724 -1.502107  1.330752e-01
## colorF       554.93230   62.38527  8.895246  6.004834e-19
## colorG       829.18158   60.34470 13.740751  6.836340e-43
## colorH      1316.71510   64.28715 20.481777  7.074714e-93
## colorI      1921.92086   71.55308 26.860072 7.078041e-158
## colorJ      2153.86392   88.13203 24.439060 3.414906e-131
```

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

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

```
X= model.matrix(price ~ color, diamonds)
y_hat3= X%*%b3
SSE_col= (as.vector((price-y_hat3)))%*%(as.vector((price-y_hat3)))
MSE_col=SSE_col/(length(price)-8)
RMSE_col=sqrt(MSE_col)
RMSE_col
```

```
##          [,1]
## [1,] 3926.813
```

```
sd(price-y_hat3)
```

```
## [1] 3926.558
```

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

Given the relation between the RMSE and R^2, it makes sense to have them each at and high and low value respectively. The coefficients of the linear model has been dummified and each display the average of the price of each respective color from as referenced from the categorical variable color D. This can be shown and read off from the box and whisker plot depicted above.

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

Albeit our model included on feature, the feature was a categorical predictor rather than a continuous one. This means the each category was dummified, representing the feature color with 7 dummied down 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.

```
b3
```

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

```
mean(diamonds$price[diamonds$color=="D"])
```

```
## [1] 3169.954
```

```
mean(diamonds$price[diamonds$color=="E"])-mean(diamonds$price[diamonds$color=="D"])
```

```
## [1] -93.20162
```

```
mean(diamonds$price[diamonds$color=="F"])-mean(diamonds$price[diamonds$color=="D"])
```

```
## [1] 554.9323
```

```
mean(diamonds$price[diamonds$color=="G"])-mean(diamonds$price[diamonds$color=="D"])
```

```
## [1] 829.1816
```

```
mean(diamonds$price[diamonds$color=="H"])-mean(diamonds$price[diamonds$color=="D"])
```

```
## [1] 1316.715
```

```
mean(diamonds$price[diamonds$color=="I"])-mean(diamonds$price[diamonds$color=="D"])
```

```
## [1] 1921.921
```

```
mean(diamonds$price[diamonds$color=="J"])-mean(diamonds$price[diamonds$color=="D"])
```

```
## [1] 2153.864
```

```
#they are the same up to a small neglible factor
```

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

```
anova_mod = lm(price ~ 0 +color , diamonds)
bd=coef(anova_mod)
bd
```

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

```
mean(diamonds$price[diamonds$color=="D"])
```

```
## [1] 3169.954
```

```
mean(diamonds$price[diamonds$color=="E"])
```

```
## [1] 3076.752
```

```
mean(diamonds$price[diamonds$color=="F"])
```

```
## [1] 3724.886
```

```
mean(diamonds$price[diamonds$color=="G"])
```

```
## [1] 3999.136
```

```
mean(diamonds$price[diamonds$color=="H"])
```

```
## [1] 4486.669
```

```
mean(diamonds$price[diamonds$color=="I"])
```

```
## [1] 5091.875
```

```
mean(diamonds$price[diamonds$color=="J"])
```

```
## [1] 5323.818
```

```
#correct up to a small neglibile factor
```

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

Extrapolation would look like a new color introduced into the data system that is outside the range of colors we are currently considering. This color would also produce a price. Since the colors are denoted in letters, a new color could be color "k".

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

```
multi_linmod = lm( price ~ . ,diamonds)
b10=coef(multi_linmod)
```

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

```
summary(multi_linmod)$coefficients
```

```
##                  Estimate Std. Error     t value      Pr(>|t|)
## (Intercept)   2184.477350 408.196605   5.3515324  8.756765e-08
## carat        11256.978307  48.627509 231.4940348  0.000000e+00
## cutGood        579.751446  33.592129  17.2585500  1.457949e-66
## cutIdeal       832.911845  33.407474  24.9319012 1.997445e-136
```

```
## cutPremium     762.143950  32.227567  23.6488208 5.151200e-123
## cutVery Good   726.782591  32.240645  22.5424330 5.260968e-112
## colorE        -209.118085  17.893109 -11.6870736  1.620034e-31
## colorF        -272.853832  18.092701 -15.0808792  2.754083e-51
## colorG        -482.038904  17.716125 -27.2090484 6.341673e-162
## colorH        -980.266675  18.835842 -52.0426247  0.000000e+00
## colorI       -1466.244474  21.162346 -69.2855361  0.000000e+00
## colorJ       -2369.398063  26.130883 -90.6742445  0.000000e+00
## clarityIF     5345.102246  51.024042 104.7565425  0.000000e+00
## claritySI1    3665.472080  43.634034  84.0048862  0.000000e+00
## claritySI2    2702.586294  43.818478  61.6768639  0.000000e+00
## clarityVS1    4578.397915  44.546000 102.7791038  0.000000e+00
## clarityVS2    4267.223565  43.853473  97.3063994  0.000000e+00
## clarityVVS1   5007.759045  47.159698 106.1872581  0.000000e+00
## clarityVVS2   4950.814072  45.854755 107.9672993  0.000000e+00
## depth          -63.806100   4.534554 -14.0710870  6.867782e-45
## table          -26.474085   2.911655  -9.0924516  1.000214e-19
## x            -1008.261098  32.897748 -30.6483316 1.601160e-204
## y                9.608886  19.332896   0.4970226  6.191751e-01
## z              -50.118891  33.486301  -1.4966983  1.344776e-01
```

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

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

```
summary(multi_linmod)$sigma
```

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

```
Xmm=model.matrix(price ~ . ,diamonds)
y_hat4=as.vector(Xmm %*% b10)
SSE_all= (price-y_hat4)%*%(price-y_hat4)
MSE_all=SSE_all/(length(price)-(ncol(Xmm)))
RMSE_all=sqrt(MSE_all)
RMSE_all
```

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

```
#95% predictive error: 2*RMSE
2*summary(multi_linmod)$sigma
```

```
## [1] 2260.189
```

```
#95% interval would be
4*summary(multi_linmod)$sigma
```

```
## [1] 4520.378
```

```
#long
```

Interpret all entries in the vector b.

Each entry chere represents the slope of the multivariate linear regression as counted from the intercept. The continuous predictors produced one dimensions that corresponds to that predictor, yet the categorical predictors were dummified and added a dimension for every level they had in their category. This was shown in the previous part of the question. Each of the entries for b that correspond to categorical predictors are the really just the mean of the prices that fall in the respective level of the categorical predictor, scaled to the intercept. Adding a unit of a 1 to anyone of these variables would change the prediction by slope shown

plus the intercept. For example, increasing carat by 1 unit, would influence our prediction to be increase that respective dimension by the sumation of the estimate for carot and the intercept as shown above.

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

As expected, the coefficients, entries of b, increased in amount because of the categorical variables being dummified. Each of these variables are referenced from the categorical variable color D. It makes sense the R^2 and the RMSE are the values they are (respectively big and small) because we have more information available, yet not enough information to get close to our n, i.e., the sample size. In such a way, we prevent overfitting yet increase the number of predictors to accurately describe price (up to a certain degree of uncertainty).

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

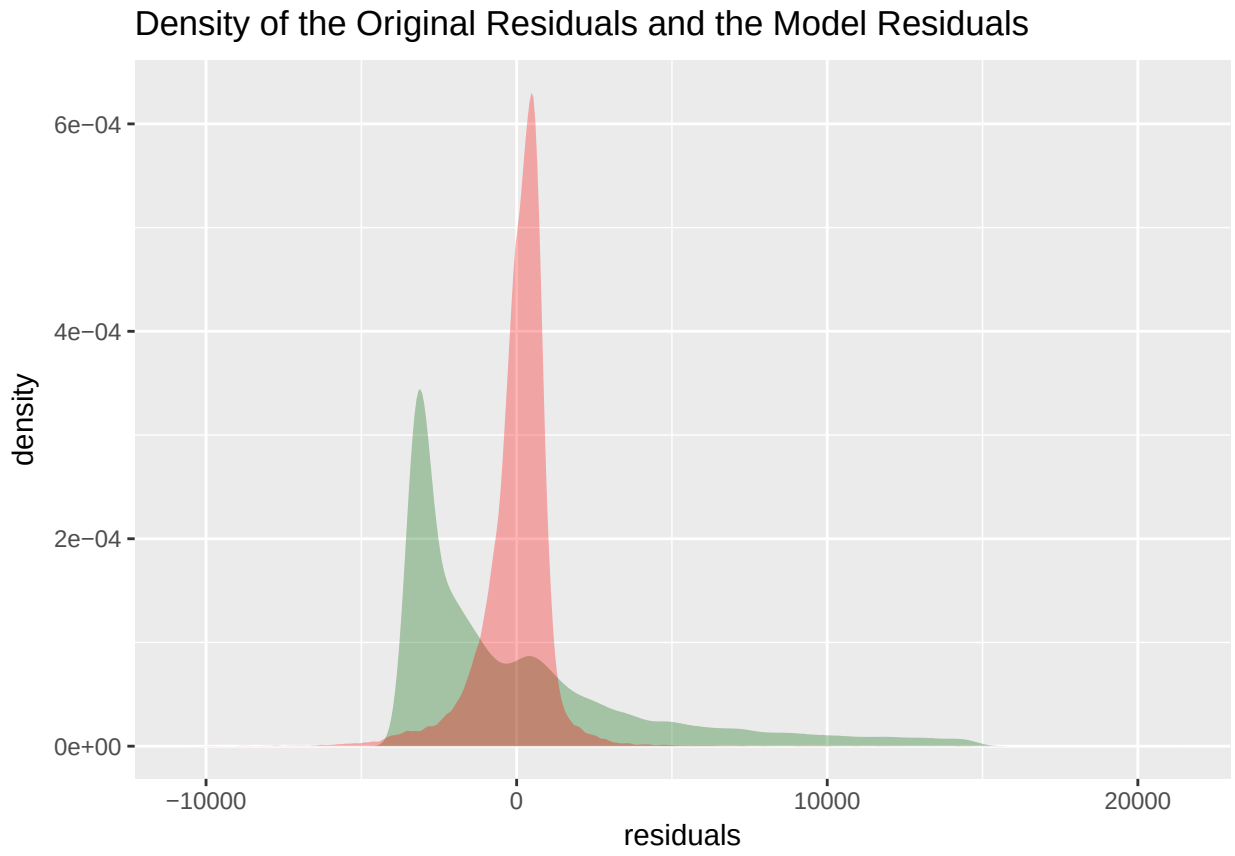
R^2 is high because this time we linearly regressed to all the predictors available. There are more predictors to explain the price and reduce our error i.e., p increased. Practically speaking, knowing more description of the diamonds, such as the carot, the color, cut, dimensions, etc. helps a diamond expert to assess more about the diamond and the price.

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

I do not think I overfit because the number of predictors used in this data frame is much less than my sample size (reducing estimation error).

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

```
orig_residuals =diamonds$price-mean(diamonds$price)
mod_residuals = y_hat4-diamonds$price
ggplot(data.frame(orig_residuals,mod_residuals)) + stat_density(aes(x=orig_residuals), fill="darkgreen"
```



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

Price vs carat appears to be close to linear, being closer to quadratic or cubic.

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

```
degree_2_poly= lm(price ~poly(carat, 2, raw=TRUE), diamonds)
#plot_function_d2= function(x,b){
#  b[1]+b[2]*x+b[3]*x^2
#}
b_2= coef(degree_2_poly)
#ggplot(diamonds, aes(carat,price)) + geom_point() + stat_function(fun = plot_function_d2, args = list(
```

What is b, R^2 and the RMSE?

```
summary(degree_2_poly)$coefficients
```

```
##                               Estimate Std. Error   t value      Pr(>|t|)
## (Intercept)                 -1832.5774   21.68730 -84.50001  0.000000e+00
## poly(carat, 2, raw = TRUE)1  6677.0273   46.40405 143.88889  0.000000e+00
## poly(carat, 2, raw = TRUE)2   507.9133   20.81960  24.39592 9.695712e-131
```

```
summary(degree_2_poly)$r.squared
```

```
## [1] 0.8509749
```

```
summary(degree_2_poly)$sigma
```

```
## [1] 1540.103
```

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.

Here, each different degree term acts as a distinct "nonsense" predictor. This affects R^2 but not substantially much. Each element of b is referenced from the intercept, which is similar to the linear regression except now there is a higher order term. Essentially, the squared term poly(carat,2,raw=TRUE)2 is treated as an additional predictor, one that is linearly independent and referenced fromt the intercept.

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

As shown in the visualization above, there is an improvement in this particular data frame; the quadratic relationship fits the data better than the linear regression but not by wide a margin. This quadratic fit can also introduce a bit of overfitting, but not a signficant amount.

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

```
g_predict=function(x_star){
  b_2=as.vector(summary(lm(price~Xmm,diamonds))$coefficients)
  x_star%*%b_2
}
```

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

```
degree_2poly= 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?

This threw an error because the predictor that is attempting to explain diamond price is a categorical one, applying a quadratic regression to a categorical predictor cannot be interpreable when considering what a squared categorical term looks like.

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.

```
y = as.vector(diamonds$price)
X7=model.matrix(price~., diamonds)
XtX=t(X7)%*%X7
XtXinv=solve(t(X7)%*%X7)
b_7=XtXinv%*%t(X7)%*%y
```

What is b, R^2 and the RMSE?

```
b_7
```

```
##                      [,1]
## (Intercept)   2184.477351
## carat        11256.978308
## cutGood        579.751446
## cutIdeal       832.911845
## cutPremium     762.143950
## cutVery Good   726.782591
## colorE        -209.118085
## colorF        -272.853832
## colorG        -482.038904
## colorH        -980.266675
## colorI       -1466.244474
## colorJ       -2369.398063
```

```
## clarityIF     5345.102246
## claritySI1    3665.472080
## claritySI2    2702.586294
## clarityVS1    4578.397915
## clarityVS2    4267.223565
## clarityVVS1   5007.759045
## clarityVVS2   4950.814072
## depth          -63.806100
## table          -26.474085
## x            -1008.261098
## y                9.608886
## z              -50.118891
```

```
y_hat7=X7%*%b_7
e=as.vector(y-y_hat7)
Rsq= (var(y)-var(e))/var(y)
Rsq
```

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

```
SSE7=e%*%e
MSE7=SSE7/(length(y)-ncol(X7))
RMSE7=sqrt(MSE7)
RMSE7
```

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

Are they the same as in #4?

They are practically the same as in #4, differing by very small orders of magnitude.

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.

```
indices = sample(1 : nrow(X7), 2000)
X7q = X7[indices, ]
yq = y[indices]
rm(indices)
qrX7=qr(X7q)
Q=qr.Q(qrX7)
R=qr.R(qrX7)
yhat_via_Q = Q %*% t(Q) %*% yq
b7=solve(R)%*%t(Q)%*%yq
b7
```

```
##                       [,1]
## (Intercept)  -34698.77982
## carat         12041.00792
## cutGood         452.36444
## cutIdeal        584.25439
## cutPremium      685.71371
## cutVery Good    394.66749
## colorE         -308.40488
## colorF         -447.36565
## colorG         -551.57353
## colorH        -1078.19161
## colorI        -1784.12336
```

```
## colorJ        -2531.86051
## clarityIF      5817.74240
## claritySI1     3927.48654
## claritySI2     2992.13833
## clarityVS1     4875.68317
## clarityVS2     4520.12466
## clarityVVS1    5275.98365
## clarityVVS2    5166.89585
## depth           560.89562
## table           -37.83466
## x               178.13925
## y              5021.77719
## z            -10529.60772
```

```
b10
```

```
##  (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
```

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

```
yhat_via_Q = as.vector(Q %*% t(Q) %*% yq)
e=as.vector(yq-yhat_via_Q)
H=Q%*%t(Q)

head(yhat_via_Q)
```

```
## [1] 3740.347 1183.350 8167.528 6790.448 3006.501 1218.818
```

```
head(e)
```

```
## [1] -752.34670 -392.35022  827.47212 1838.55178 -839.50091  -52.81817
```

```
#head(H)
```

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

```
#(a)
head(yhat_via_Q+e)==head(yq)
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE
```

```
#(b)
yhat_via_Q%*%e
```

```
##              [,1]
## [1,] 0.0002864525
```

```
#(c)
head(H%*%e)
```

```
##                [,1]
## [1,]  3.197442e-14
## [2,] -4.356204e-11
## [3,]  3.370015e-11
## [4,]  1.867662e-11
## [5,] -1.175637e-11
## [6,] -3.213430e-11
```

```
#(d)
head(yhat_via_Q)
```

```
## [1] 3740.347 1183.350 8167.528 6790.448 3006.501 1218.818
```

```
head(H%*%yhat_via_Q)
```

```
##          [,1]
## [1,] 3740.347
## [2,] 1183.350
## [3,] 8167.528
## [4,] 6790.448
## [5,] 3006.501
## [6,] 1218.818
```

```
#(e) the error vector lives in the orthogonal complement of the column space of X
head((diag(ncol(H))-H)%*%yhat_via_Q)
```

```
##                [,1]
## [1,]  1.845635e-12
## [2,] -4.343903e-11
## [3,]  5.168488e-11
## [4,]  5.908163e-12
## [5,] -4.947154e-12
## [6,] -2.955858e-11
```

```
#(f)
e%*%e+(yhat_via_Q-rep(mean(yq),length(yq)))%*%(yhat_via_Q-rep(mean(yq),length(yq)))
```

```
##             [,1]
## [1,] 32799399801
```

```
(yq-rep(mean(yq),length(yq)))%*%(yq-rep(mean(yq),length(yq)))
```

```
##             [,1]
## [1,] 32799399801
```

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

```
mod8=lm(price ~.*.+ poly(carat,5) + poly(depth,5) + poly(table,5) + poly(price,5)+poly(x,5)+poly(y,5)+p
```

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

```
summary(mod8)$r.squared
```

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

```
summary(mod8)$sigma
```

```
## [1] 1.401415e-10
```

```
yhat8=predict(mod8,diamonds)
```

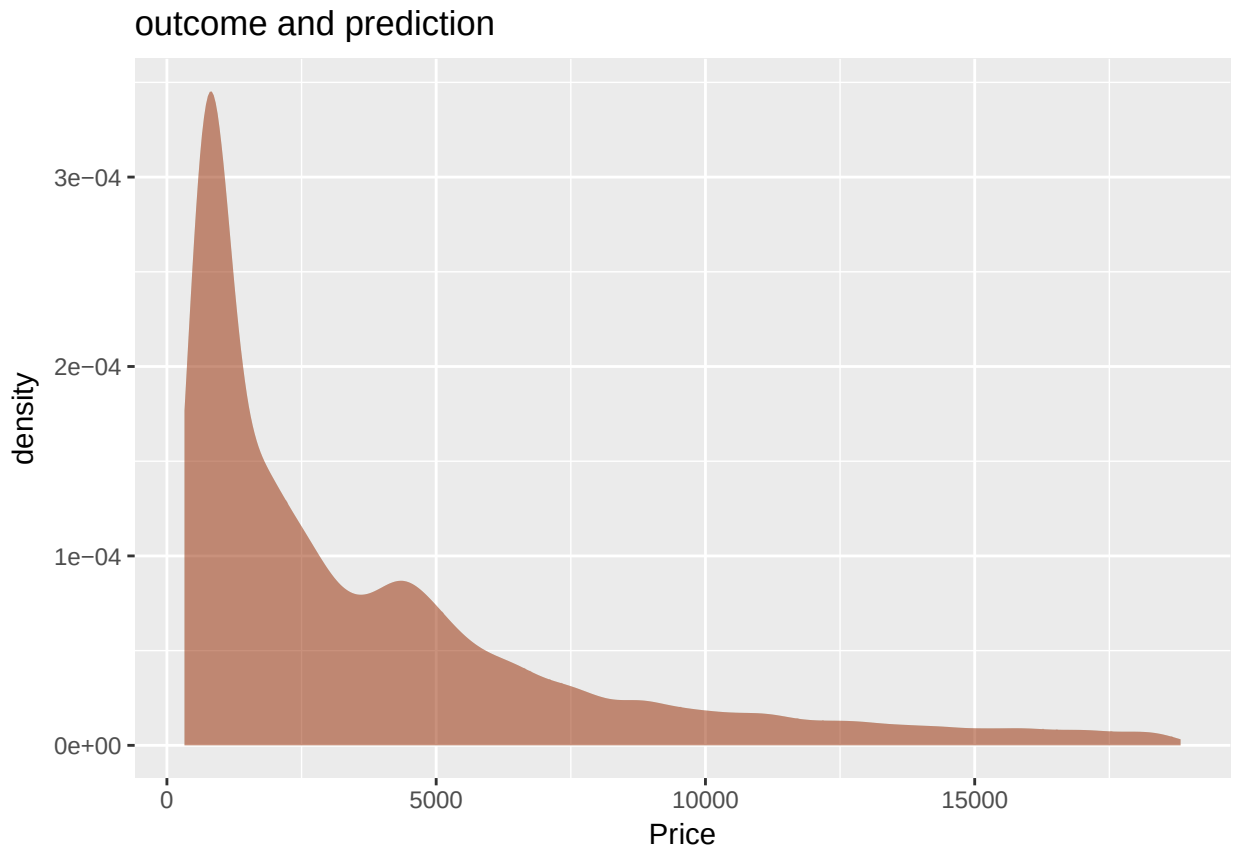
```
## Warning in predict.lm(mod8, diamonds): prediction from a rank-deficient fit
## may be misleading
```

```
e=as.vector(diamonds$price-yhat8)
sd(e)
```

```
## [1] 1.351493e-10
```

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

```
y8=diamonds$price
ggplot(data.frame(yhat8, y8)) + stat_density(aes(x=yhat8), fill="darkgreen", alpha=0.3) +  stat_density
```



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

```
sum(e>=1000)
```

```
## [1] 0
```

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.

The RMSE is very small, which is impressive yet this is due to the interactions and the non-linear regression we fitted the data, increasing the degrees of freedom and overfitting to this data frame. As someone attempting to purchase diamonds, I

What is the degrees of freedom in this model?

```
dof=length(summary(mod8)$coefficient)
dof
```

```
## [1] 1056
```

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

I do not think so, because there is too much overfitting due the the high order of polynomial regression (5th order).

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

I think g is not close to f, because it is not close to h so it must be farther away from h.

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

I can remodel my regression to include more interactions of a higher order, and reduce the regression of the polynomial order to second order while introducing a regression of a third polynomial order.

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.

We can expand the range of some of the predictor such as bigger diamonds and or heavier weight, and incorporate that into our data frame. We can also introduce additional predictors such as fluorescence.

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 * 1 / K)
train_indices=setdiff(1 : n, test_indices)
diamonds_test=diamonds[test_indices,]
diamonds_train=diamonds[train_indices,]
rm(test_indices)
rm(train_indices)
mod9=lm(price ~.*.+ poly(carat,5) + poly(depth,5) + poly(table,5) + poly(price,5)+poly(x,5)+poly(y,5)+p
yhat_test=predict(mod9,diamonds_test)
```

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

```
y_test=diamonds_test$price
sd(y_test-yhat_test)
```

```
## [1] 2.040003e-10
```

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?

There are more magnitudes of error in this validation, which implies overfitting has occured in the model from #8.

Extra-credit: validate the model via cross validation.

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

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

** TO-DO

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:

```
k=5
s_e_s= rep(0,ncol(X))
for(i in 1:ncol(X)){
test_indices = sample(1:n, size= n*1/k)
train_indices = setdiff(1:n, test_indices)
X_test = X[test_indices,i]
X_train = X[train_indices,i]
y_train = y[train_indices]
mod = lm(y_train ~ X_train)
y_hat_oos = predict(mod, X_test)
y_test = y[test_indices]
rm(test_indices)
rm(train_indices)
s_e_s[i]=sd(y_hat_oos-y_test)
}
```

```
## Error in eval(predvars, data, env): numeric 'envir' arg not of length one
```

```
names(s_e_s) = paste("mod", 1 : ncol(X), sep = "")
length(y_train)
```

```
## [1] 80
```

```
length(X_test)
```

```
## [1] 20
```

```
s_e_s
```

```
## mod1 mod2 mod3
##    0    0    0
```

```r
names(which.min(s_e_s))
```

```
## [1] "mod1"
```