Lab 5

Namisha Singh

April 3, 2022

We will work with the diamonds dataset from last lecture:

```r
pacman::p_load(ggplot2) #this loads the diamonds data set too
?diamonds
diamonds$cut =      factor(diamonds$cut, ordered = FALSE)
diamonds$color =    factor(diamonds$color, ordered = FALSE)
diamonds$clarity =  factor(diamonds$clarity, ordered = FALSE)
skimr::skim(diamonds)
```

Data summary

Name	diamonds
Number of rows	53940
Number of columns	10
_______________________	
Column type frequency:	
factor	3
numeric	7
________________________	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
cut	0	1	FALSE	5	Ide: 21551, Pre: 13791, Ver: 12082, Goo: 4906
color	0	1	FALSE	7	G: 11292, E: 9797, F: 9542, H: 8304
clarity	0	1	FALSE	8	SI1: 13065, VS2: 12258, SI2: 9194, VS1: 8171

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
carat	0	1	0.80	0.47	0.2	0.40	0.70	1.04	5.01	▇▂▁▁▁

depth	0	1	61.75	1.43	43.0	61.00	61.80	62.50	79.00	▁▁▇▁▁
table	0	1	57.46	2.23	43.0	56.00	57.00	59.00	95.00	▁▇▁▁▁
price	0	1	3932.80	3989.44	326.0	950.00	2401.00	5324.25	18823.00	▇▂▁▁▁
x	0	1	5.73	1.12	0.0	4.71	5.70	6.54	10.74	▁▁▇▃▁
y	0	1	5.73	1.14	0.0	4.72	5.71	6.54	58.90	▇▁▁▁▁
z	0	1	3.54	0.71	0.0	2.91	3.53	4.04	31.80	▇▁▁▁▁

Given the information above, what are the number of columns in the raw X matrix?

9

Verify this using code:

```r
ncol(diamonds)

## [1] 10
```

Would it make sense to use polynomial expansions for the variables cut, color and clarity? Why or why not?

No such thing as polynomial expansions for categorical variables.

Would it make sense to use log transformations for the variables cut, color and clarity? Why or why not?

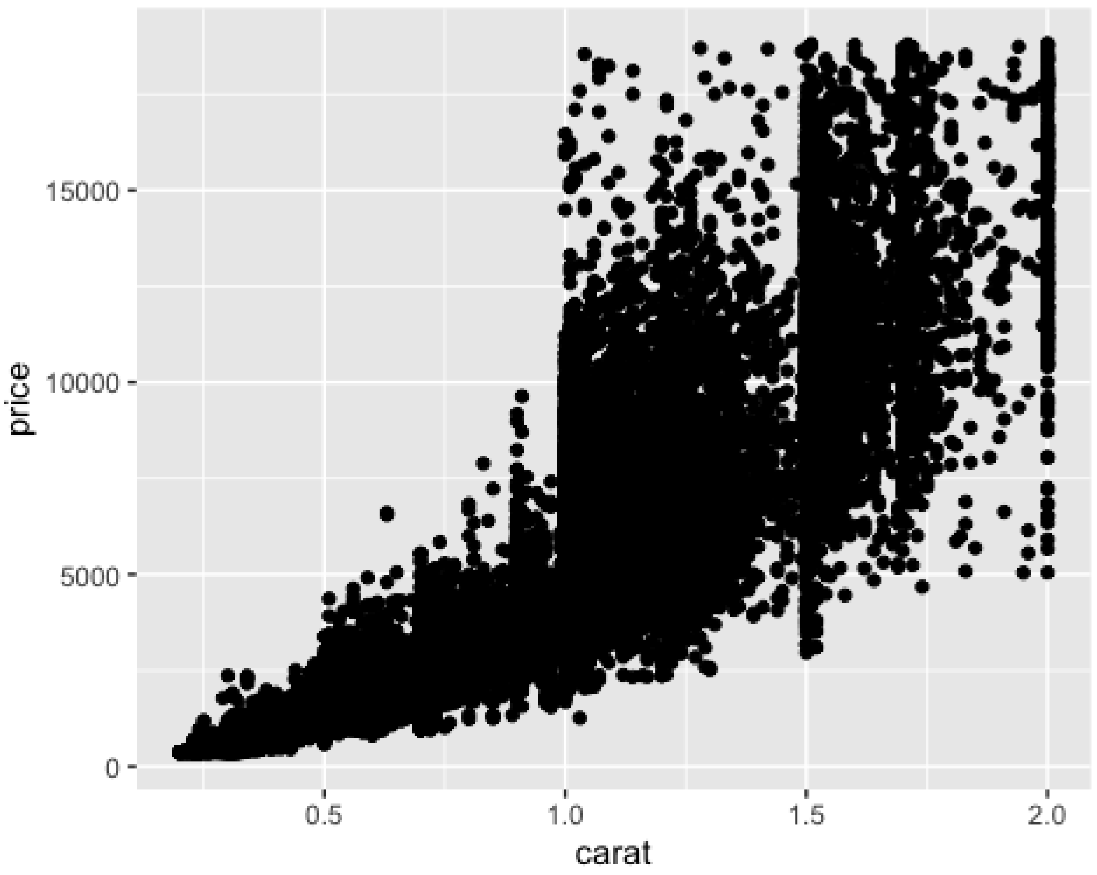
No, it would not make sense to use log transformations for cut, color, and clarity because they are categorical.

In order to ensure there is no time trend in the data, randomize the order of the diamond observations in D:.

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

Let's also concentrate only on diamonds with <= 2 carats to avoid the issue we saw with the maximum. So subset the dataset. Create a variable n equal to the number of remaining rows as this will be useful for later. Then plot it.

```r
diamonds = diamonds[diamonds$carat<=2,]
n = nrow(diamonds)
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point()
```



Create a linear model of price ~ carat and gauge its in-sample performance using s_e.

```
mod1 = lm(price ~ carat, diamonds)
summary(mod1)$sigma

## [1] 1451.927
```

Create a model of price ~ clarity and gauge its in-sample performance

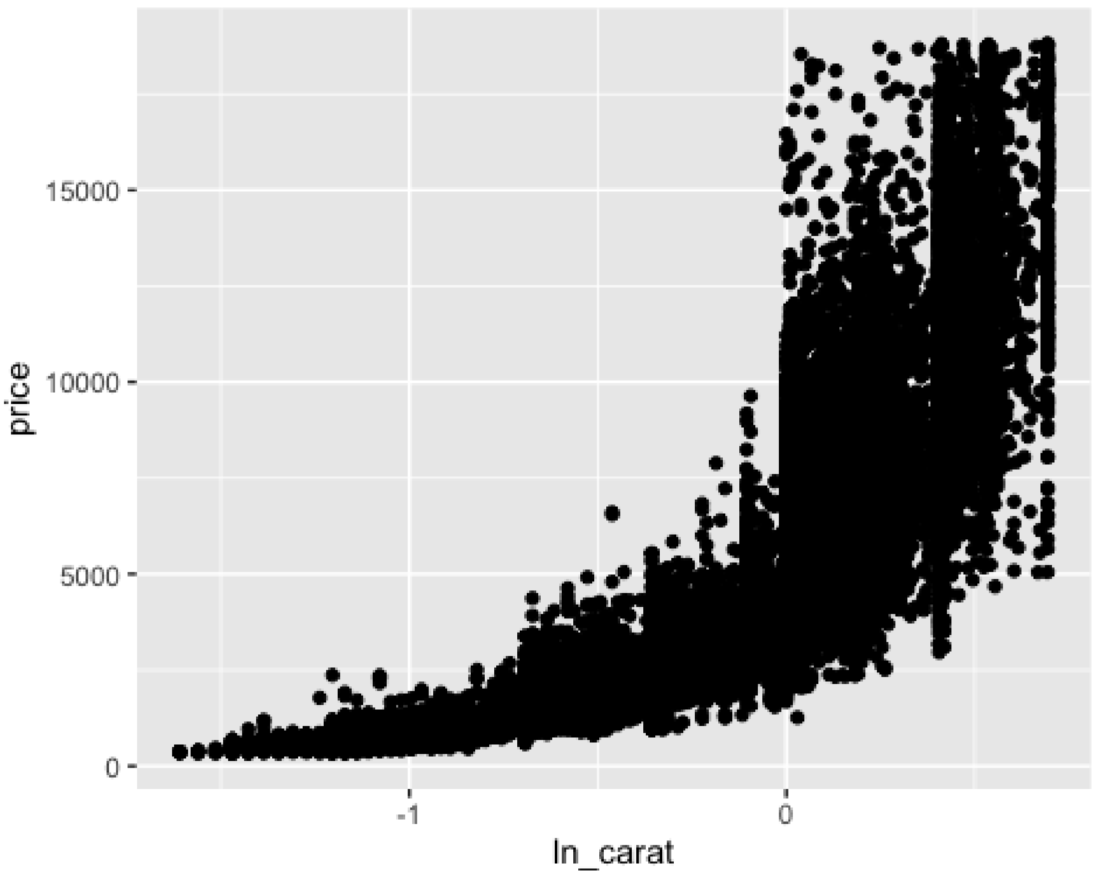
```
mod = lm(price ~ clarity, diamonds)
summary(mod)$sigma

## [1] 3395.901
```

Why is the model price ~ carat substantially more accurate than price ~ clarity?

The model price ~ carat is substantially more accurate because it is continuous and hence can assess more data points whereas clarity is ordinal categorical so it would need to be coded in terms of severity.

Create a new transformed feature ln_carat and plot it vs price.

```
diamonds$ln_carat = log(diamonds$carat)
ggplot(diamonds, aes(x = ln_carat, y = price)) +
  geom_point()
```



Would price ~ ln_carat be a better fitting model than price ~ carat? Why or why not?

No it would not be a better fitting model since the data points from the plot show that it does not need to be transformed with log.

Verify this by comparing R^2 and RMSE of the two models:

```
mod1 = lm(price~carat,diamonds)
summary(mod1)

##
## Call:
## lm(formula = price ~ carat, data = diamonds)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8100.5  -775.3   -21.9   519.6 12770.1
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2222.72      13.45  -165.2   <2e-16 ***
## carat        7687.13      15.83   485.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

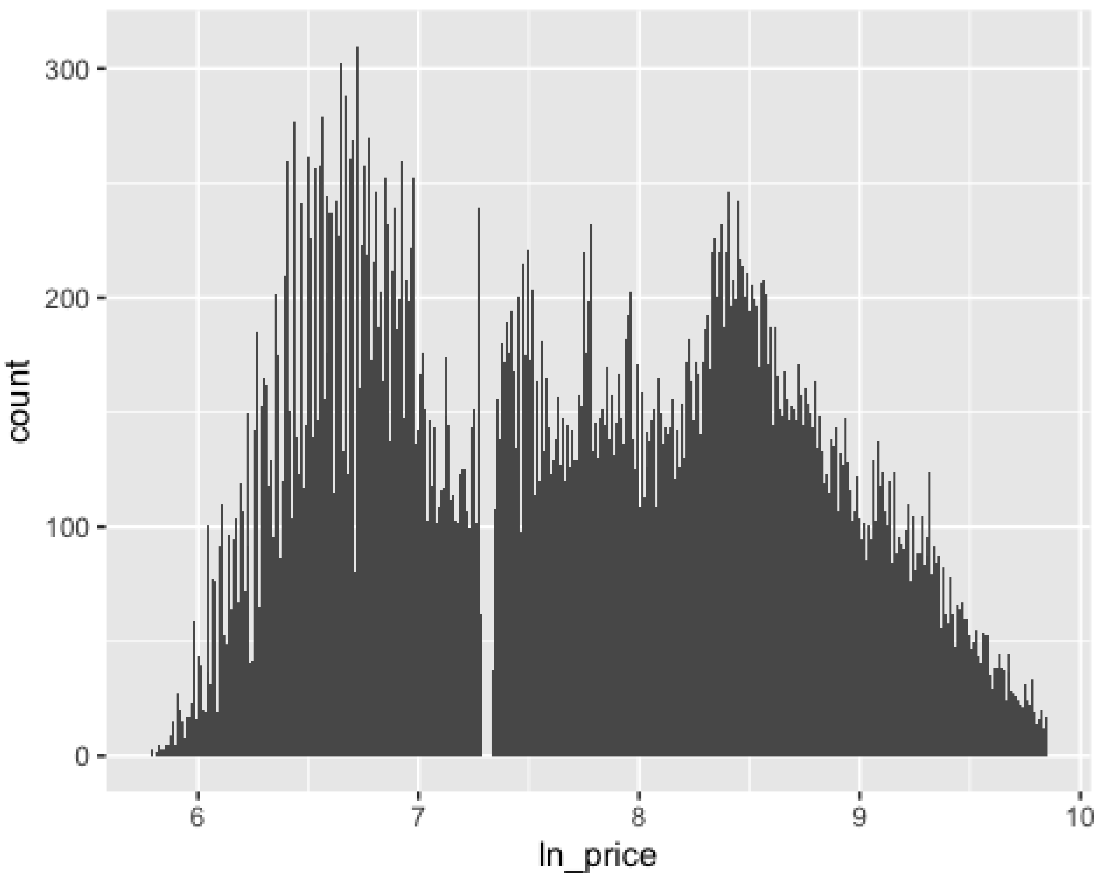
```
##
## Residual standard error: 1452 on 52049 degrees of freedom
## Multiple R-squared:  0.8192, Adjusted R-squared:  0.8192
## F-statistic: 2.359e+05 on 1 and 52049 DF,  p-value: < 2.2e-16
```

```
mod2 = lm(price~ln_carat,diamonds)
summary(mod2)
```

```
##
## Call:
## lm(formula = price ~ ln_carat, data = diamonds)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4974.2 -1235.6  -325.8   890.9 12519.0
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5817.93      10.25   567.7   <2e-16 ***
## ln_carat     5229.25      14.57   358.9   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1832 on 52049 degrees of freedom
## Multiple R-squared:  0.7122, Adjusted R-squared:  0.7122
## F-statistic: 1.288e+05 on 1 and 52049 DF,  p-value: < 2.2e-16
```

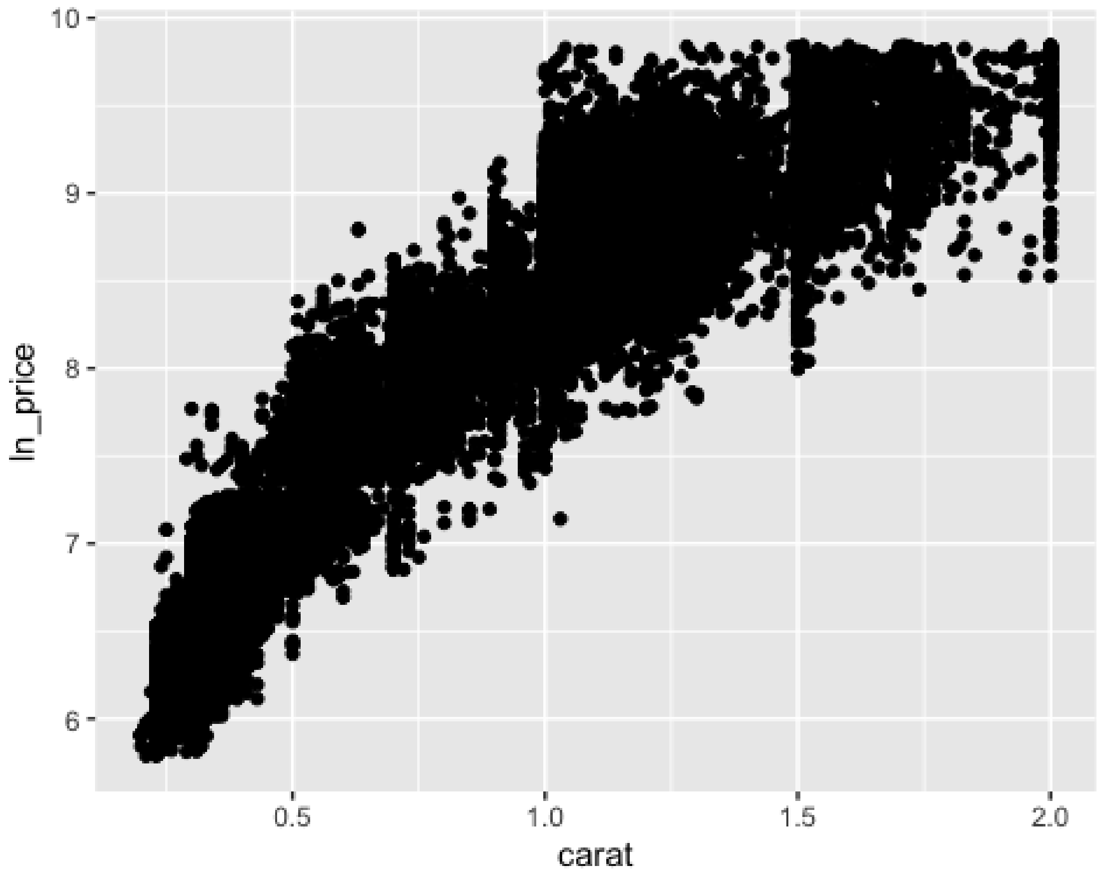
Create a new transformed feature ln_price and plot its estimated density:

```
diamonds$ln_price = log(diamonds$price)
ggplot(diamonds) + geom_histogram(aes(x = ln_price), binwidth = 0.01)
```



Now plot it vs carat.

```
ggplot(diamonds, aes(x = carat, y = ln_price)) +
  geom_point()
```



Would ln_price ~ carat be a better fitting model than price ~ carat? Why or why not?

Yes, ln_price ~ carat would be a better fitting model than price ~ carat because from the graph above, we can see that its distribution is more normal.

Verify this by computing s_e of this new model. Make sure these metrics can be compared apples-to-apples with the previous.

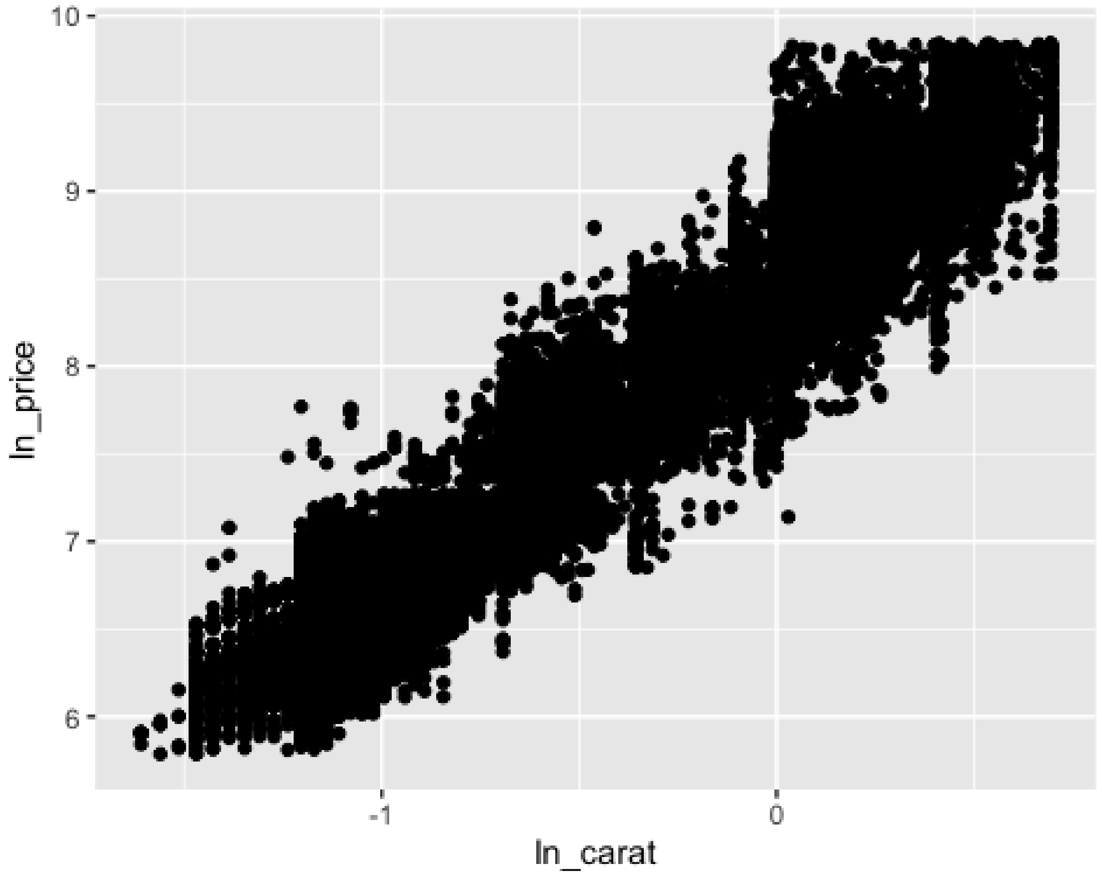
```
mod3 = lm(ln_price ~ carat, diamonds)
y_hat = exp(mod3$fitted.values)
sse = sum((y_hat - diamonds$price)^2)
sqrt(sse/(n-2))

## [1] 2725.847

summary(mod1)$sigma

## [1] 1451.927
```

We just compared in-sample statistics to draw a conclusion on which model has better performance. But in-sample statistics can lie! Why is what we did likely valid?

We are only using one feature! We aren’t overfitting with one feature and 52K data points.

Plot ln_price vs ln_carat.

```r
ggplot(diamonds, aes(x = ln_carat, y = ln_price)) +
  geom_point()
```



Would ln_price ~ ln_carat be the best fitting model than the previous three we considered? Why or why not? Yeah because it looks like a line.

Verify this by computing s_e of this new model. Make sure these metrics can be compared apples-to-apples with the previous.

```r
mod4 = lm(ln_price~ ln_carat, diamonds)
y_hat = exp(mod4$fitted.values)
sse = sum((y_hat - diamonds$price)^2)
sqrt(sse/(n-2))

## [1] 1396.67

summary(mod1)$sigma

## [1] 1451.927
```

Compute b, the OLS slope coefficients for this new model of ln_price ~ ln_carat.

```r
coef(mod4)
```

```
## (Intercept)    ln_carat
##    8.462579    1.696590
```

```r
#Model A
moda = lm(ln_price ~ ln_carat,diamonds)
summary(moda)$sigma
```

```
## [1] 0.2613383
```

Interpret b_1, the estimated slope of ln_carat.

% change in price is approximately equal to % change in x

Interpret b_0, the estimated intercept.

predicted price of a diamond with no weight. Minimum limit of logical function

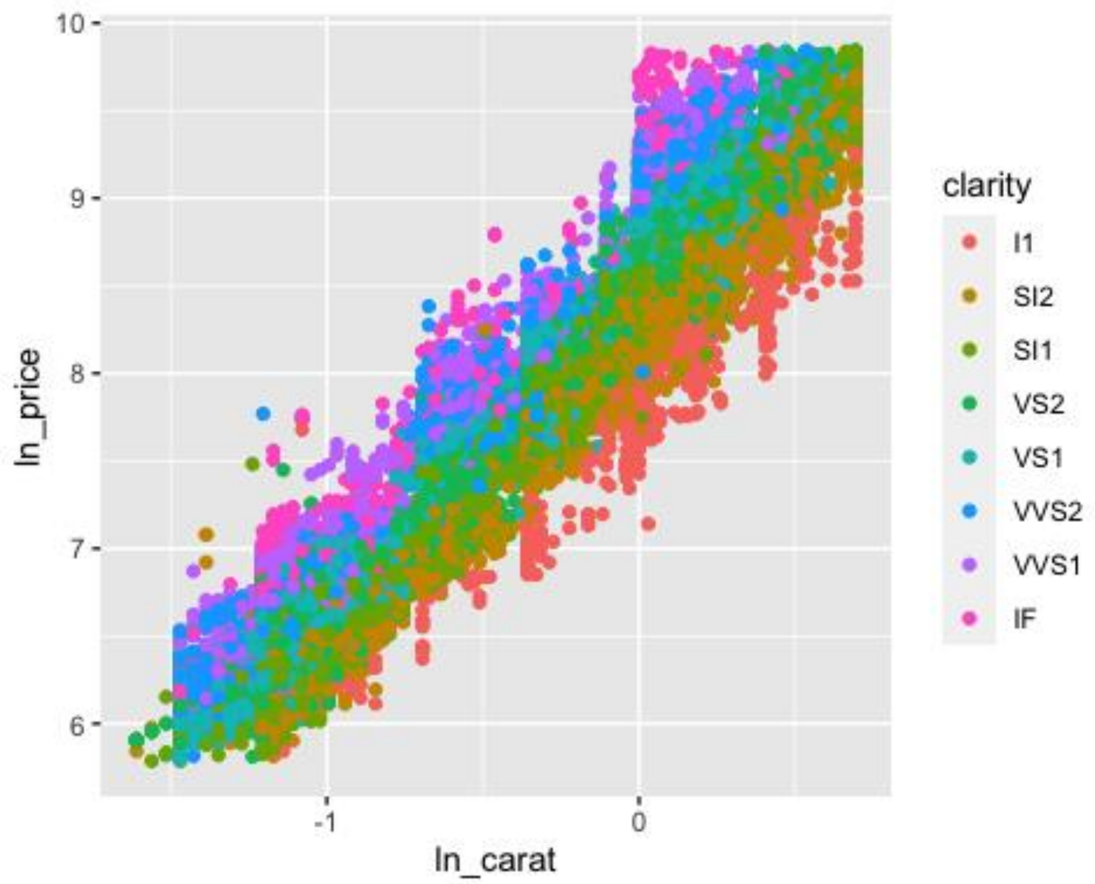
Create other features ln_x, ln_y, ln_z, ln_depth, ln_table.

```r
diamonds$ln_x = log(diamonds$x)
diamonds$ln_y = log(diamonds$y)
diamonds$ln_z = log(diamonds$z)
diamonds$ln_depth = log(diamonds$depth)
diamonds$ln_table = log(diamonds$table)
```

From now on, we will be modeling ln_price (not raw price) as the prediction target.

Create a model (B) of ln_price on ln_carat interacted with clarity and compare its performance with the model (A) ln_price ~ ln_carat.

```r
ggplot(diamonds, aes(x = ln_carat, y = ln_price, color = clarity)) +
geom_point()
```



```
moda = lm(ln_price ~ ln_carat,diamonds)
summary(moda)$sigma

## [1] 0.2613383

modb = lm(ln_price ~ ln_carat * clarity,diamonds)
summary(modb)$sigma

## [1] 0.1865689
```

Which model does better? Why?

Model B did much better

Create a model of (C) ln_price on ln_carat interacted with every categorical feature (clarity, cut and color) and compare its performance with model (B)

```
#Model C
modc = lm(ln_price ~ ln_carat * (clarity + cut + color), diamonds)
summary(modc)$sigma

## [1] 0.1300859
```

Which model does better? Why?

C does better because we're adding more useful dimensions.

Create a model (D) of ln_price on every continuous feature (logs of carat, x, y, z, depth, table) interacted with every categorical feature (clarity, cut and color) and compare its performance with model (C).

```
#Model D
diamonds = diamonds[diamonds$x > 0 & diamonds$ln_z >0,]
diamonds[is.infinite(diamonds$ln_z),]

## # A tibble: 0 x 17
## # … with 17 variables: carat <dbl>, cut <fct>, color <fct>, clarity <fct>,
## #   depth <dbl>, table <dbl>, price <int>, x <dbl>, y <dbl>, z <dbl>,
## #   ln_carat <dbl>, ln_price <dbl>, ln_x <dbl>, ln_y <dbl>, ln_z <dbl>,
## #   ln_depth <dbl>, ln_table <dbl>

modd = lm(ln_price ~ (ln_carat + ln_x  + ln_y + ln_z + ln_depth + ln_table) *
(clarity + cut + color), diamonds)
summary(modd)$sigma

## [1] 0.1269092
```

Which model does better? Why?

Model D because we're adding another orthogonal projection to the column space of x.

What is the p of this model D? Compute with code.

```
modd$rank

## [1] 126

ncol(model.matrix( ~ (ln_carat + ln_x  + ln_y + ln_z + ln_depth + ln_table) *
(clarity + cut + color), diamonds))

## [1] 126
```

Create model (E) which is the same as before except create include the raw features interacted with the categorical features and gauge the performance against (D).

```
#Model E
mod_e = lm(ln_price ~ (carat + x  + y + z + depth + table) * (clarity + cut +
color), diamonds)
summary(mod_e)$sigma

## [1] 0.1260733
```

Which model does better? Why?

Mod E is better because linear slopes are better in this case but the decrease in RMSE may not be significant

Create model (F) which is the same as before except also include also third degree polynomials of the continuous features interacted with the categorical features and gauge performance against (E). By this time you're getting good with R's formula syntax!

```
#Model F
mod_f = lm(ln_price ~ (poly(carat,3) + poly(x,3)  + poly(y,3) + poly(z,3) +
poly(depth,3) + poly(table,3)) * (clarity + cut + color), diamonds)
summary(mod_f)$sigma

## [1] 0.1082411
```

Which model does better? Why?

Model F does better as it's another projection.

Can you think of any other way to expand the candidate set curlyH? Discuss.

You can expand the set curlH even further by adding even more interactions.

We should probably assess oos performance now. Sample 2,000 diamonds and use these to create a training set of 1,800 random diamonds and a test set of 200 random diamonds. Define K and do this splitting:

```
K = 10
set.seed(1984)
nsamp = 2000
D = diamonds[sample(1:nrow(diamonds), nsamp),]
Dtrain = D[1: ((1-1/K)*nsamp),]
Dtest = D[((1-1/K)*nsamp +1): nsamp,]
```

Compute in and out of sample performance for models A-F. Use s_e as the metric (standard error of the residuals). Create a list with keys A, B, ..., F to store these metrics. Remember the performances here will be worse than before since before you're using nearly 52,000 diamonds to build a model and now it's only 1,800!

```
insampleRMSE <- list()
oosRMSE <- list()

#copy mods
moda = lm(ln_price ~ ln_carat,Dtrain)
summary(moda)$sigma

## [1] 0.261077

modb = lm(ln_price ~ ln_carat * clarity, Dtrain)
summary(modb)$sigma

## [1] 0.1872862

modc = lm(ln_price ~ ln_carat * (clarity + cut + color), Dtrain)
summary(modc)$sigma
```

```
## [1] 0.1275609
```

```
modd = lm(ln_price ~ (ln_carat + ln_x  + ln_y + ln_z + ln_depth + ln_table) *
(clarity + cut + color), Dtrain)
summary(modd)$sigma
```

```
## [1] 0.1246257
```

```
mod_e = lm(ln_price ~ (carat + x  + y + z + depth + table) * (clarity + cut +
color), Dtrain)
summary(mod_e)$sigma
```

```
## [1] 0.1237445
```

```
mod_f = lm(ln_price ~ (poly(carat,3) + poly(x,3)  + poly(y,3) + poly(z,3) +
poly(depth,3) + poly(table,3)) * (clarity + cut + color), Dtrain)
summary(mod_f)$sigma
```

```
## [1] 0.1003676
```

```
insampleRMSE[['A']] <- summary(moda)$sigma

oosRMSE[['A']] <- sd(Dtest$ln_price - predict(moda,Dtest))
oosRMSE[['B']] <- sd(Dtest$ln_price -predict(modb, Dtest))
oosRMSE[['C']] <- sd(Dtest$ln_price -predict(modc, Dtest))
oosRMSE[['D']] <- sd(Dtest$ln_price -predict(modd, Dtest))
oosRMSE[['E']] <- sd(Dtest$ln_price -predict(mod_e, Dtest))
oosRMSE[['F']] <- sd(Dtest$ln_price -predict(mod_f, Dtest))

oosRMSE
```

```
## $A
## [1] 0.2618867
##
## $B
## [1] 0.1948186
##
## $C
## [1] 0.1306982
##
## $D
## [1] 0.1312725
##
## $E
## [1] 0.1327393
##
## $F
## [1] 0.1459769
```

```
insampleRMSE
```

```
## $A
## [1] 0.261077
```

You computed oos metrics only on n_* = 200 diamonds. What problem(s) do you expect in these oos metrics?

Since we only had n_* = 200 when n = 53940, we probably would expect there to be an innacurate number as we are choosing from a much smaller sample of the total n that we have and K is large.

To do the K-fold cross validation we need to get the splits right and crossing is hard. I've developed code for this already. Run this code.

```
temp = rnorm(n)
folds_vec = cut(temp, breaks = quantile(temp, seq(0, 1, length.out = K + 1)),
include.lowest = TRUE, labels = FALSE)
head(folds_vec, 200)
```

```
##   [1]  8  9  3  2  5 10  7  4  1  2  9 10  2 10  5  2  9  8  2  2  2  6  9
2  5
##  [26]  3  3  1  1  3  3 10  5 10  5  5  9  1  5  6  6  3  1  9  1  3  2  3
10  1
##  [51]  6  2  7  2  4  1  8  3 10  5 10  9  2  3 10  1  4  5 10  3  9  3  1
8  3
##  [76]  9 10  4  7  7  1  1  7  6  3  9  9 10 10  4  6  6  3 10  2  3  3  9
3  1
## [101]  1  9  6  1  8  6  8  7 10  6  3  1  7  4  3  6  3  5  8  5  9  1  5
8  6
## [126]  8  1  1 10  2 10  9  2  2  1  5  3  5  8 10  8 10  4 10  4  6  1 10
5  4
## [151]  6  2  9  8  3  7  8 10  1  5  8  2  8  3 10  4  4 10  8  7  2  3  1
7  8
## [176]  5  2  7  5  7  6  4  2  7  9  4  1  8  4  3  7  6  9  3  6  9  9  6
3  4
```

Comment on what it does and how to use it to do a K-fold CV:

The code above creates temp which has random variables with a normal distribution. Essentially, it tells you which index is inside of which fold's test set.

It divides the dataset into k groups or "folds" that are equal size.This can be used to do K-fold cross validation because we can then choose one of the folds to be teh holdout set and then repeat this process k times while using a completely different set to be in D_test. Finally, we can calculate the overall D_test RMSE, which would be the average of the k test's RMSE's.

Do the K-fold cross validation for model F and compute the overall s_e and s_s_e.

```
model_formulas = list(
  A = ln_price ~ ln_carat,
  B = ln_price ~ ln_carat * clarity,
  C = ln_price ~ ln_carat * (clarity + cut + color),
  D = ln_price ~ (ln_carat + ln_x  + ln_y + ln_z + ln_depth + ln_table) *
(clarity + cut + color),
  E = ln_price ~ (carat + x  + y + z + depth + table) * (clarity + cut +
color)
# , F = ln_price ~ (poly(carat,3) + poly(x,3)  + poly(y,3) + poly(z,3) +
poly(depth,3) + poly(table,3)) * (clarity + cut + color)
)

#Model A
mod = lm(model_formulas[["A"]], diamonds)
summary(mod)$sigma

## [1] 0.2612715

mod$rank #i.e. degrees of freedom  = # vectors in colsp[X] to project onto

## [1] 2

#Model B
mod = lm(model_formulas[["B"]], diamonds)
summary(mod)$sigma

## [1] 0.1865674

mod$rank #i.e. degrees of freedom  = # vectors in colsp[X] to project onto

## [1] 16

#Model C
mod = lm(model_formulas[["C"]], diamonds)
summary(mod)$sigma

## [1] 0.1300683

mod$rank #i.e. degrees of freedom  = # vectors in colsp[X] to project onto

## [1] 36

#Model D
mod = lm(model_formulas[["D"]], diamonds)
summary(mod)$sigma

## [1] 0.1269092

mod$rank #i.e. degrees of freedom  = # vectors in colsp[X] to project onto

## [1] 126

#Model E
mod = lm(model_formulas[["E"]], diamonds)
summary(mod)$sigma

## [1] 0.1260733
```

```
mod$rank

## [1] 126

#Model F
#mod = lm(model_formulas[["F"]], diamonds)
#summary(mod)$sigma
#mod$rank




oos_se = list()
oos_s_se = list()
for (model_idx in LETTERS[1 : 5]){
  e_vec_k = list()
  for (k in 1 : K){
    test_indicies_k = which(folds_vec == k)
    train_indicies_k = which(folds_vec != k)
    mod = lm(model_formulas[[model_idx]], diamonds[train_indicies_k, ])
    e_vec_k[[k]] = sd(diamonds$price[test_indicies_k] - predict(mod,
diamonds[test_indicies_k, ]))
  }
  oos_se[[model_idx]] = mean(unlist(e_vec_k))
  oos_s_se[[model_idx]] = sd(unlist(e_vec_k))
}
res = rbind(unlist(oos_se), unlist(oos_s_se))
rownames(res) = c("avg", "sd")
res

##      A  B  C  D  E
## avg NA NA NA NA NA
## sd  NA NA NA NA NA
```

Does K-fold CV help reduce variance in the oos s_e? Discuss.

Yes, K-fold CV does help reduce variance in the oos s_e because when you take k folds, you are looking at different samples of the data which helps ensure more data points are validated, hence reducing variation in results. It is essentially an average so its standard error is approximately s_e / sqrt(K). It's only approximate because it is the formula for the std err or independent samples. K oos samples are not exactly independent.

Imagine using the entire rest of the dataset besides the 2,000 training observations divvied up into slices of 200. Measure the oos error for each slice on Model F in a vector s_e_s_F and compute the s_s_e_F and also plot it.

```
n_step = 1 / K * nsamp
oos_se = list()
ses = list()
n_test = 1 / K * nsamp
n_train = nsamp - n_test
all_models_train = list()
```

```r
test_indicies = sample(1 : n, n_test)
train_indicies = sample(setdiff(1 : n, test_indicies), n_train)
all_other_indicies = setdiff(1 : n, c(test_indicies, train_indicies))
starting_ks = seq(from = 1, to = (length(all_other_indicies) - n_step), by =
n_step)
#for (model_idx in LETTERS[1 : 5]){
#  se_k = list()
#  for (k in 1 : length(starting_ks)){
#    diamonds_k = diamonds[all_other_indicies[starting_ks[k] :
(starting_ks[k] + n_step - 1)], ]
#    se_k[[k]] = sd(diamonds_k$price - predict(all_models_train[[model_idx]],
diamonds_k))
#  }
#  ses[[model_idx]] = se_k
#  oos_se[[model_idx]] = unlist(se_k)
#}
# pacman::p_load(reshape2)
# ggplot(reshape2::melt(oos_se)) + geom_boxplot(aes(x = L1, y = value)) +
xlab("model")
# ggplot(reshape2::melt(oos_se)) + geom_boxplot(aes(x = L1, y = value)) +
xlab("model") + ylim(0, 5000)
# ggplot(data.frame(s_e_s_F = s_e_s_F)) + geom_histogram(aes(x = s_e_s_F))
```