Lab05

Rachel Brunner

11:59PM March 12, 2022

We will work with the diamonds dataset from last lecture:

```
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)
```

Table 1: 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:

```
ncol(diamonds)
```

```
## [1] 10
```

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

#TO-DO

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

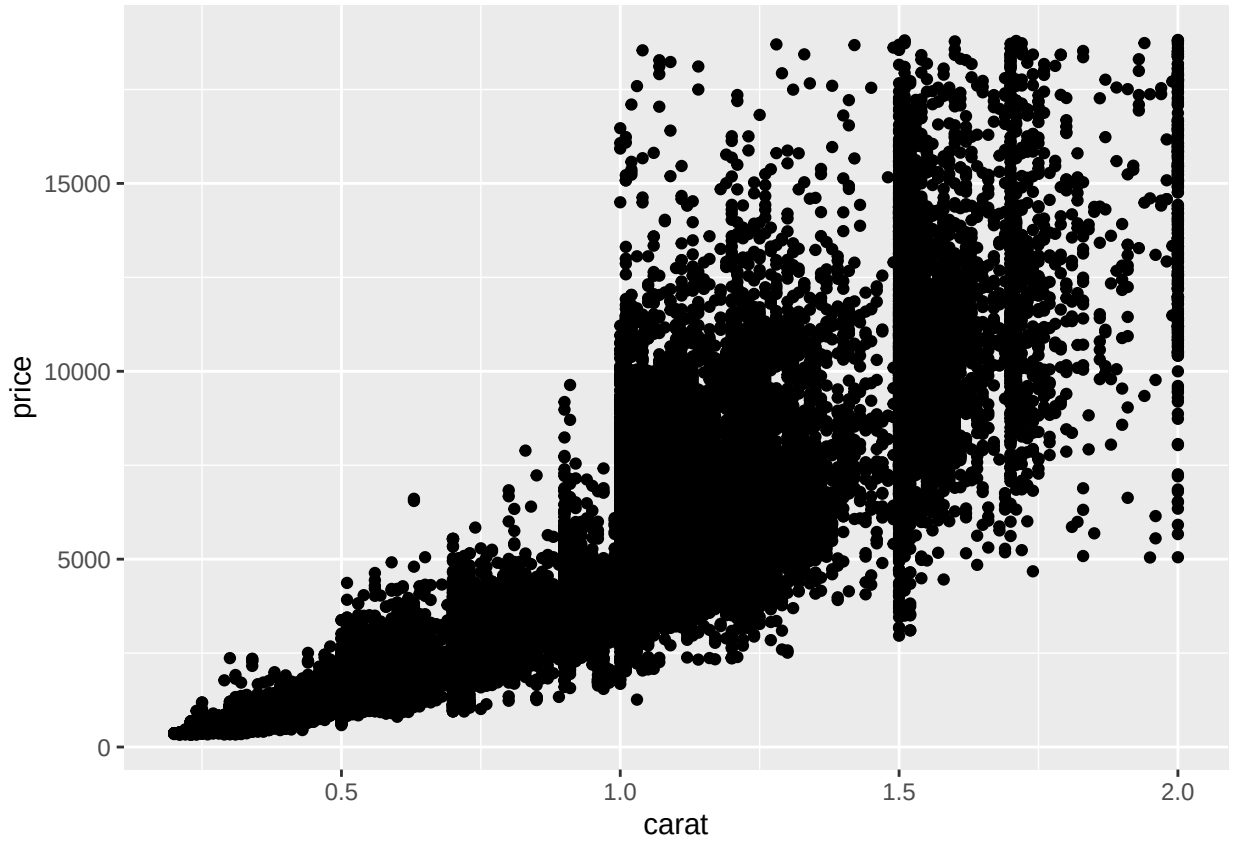
#TO-DO

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

```
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.

```
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
```

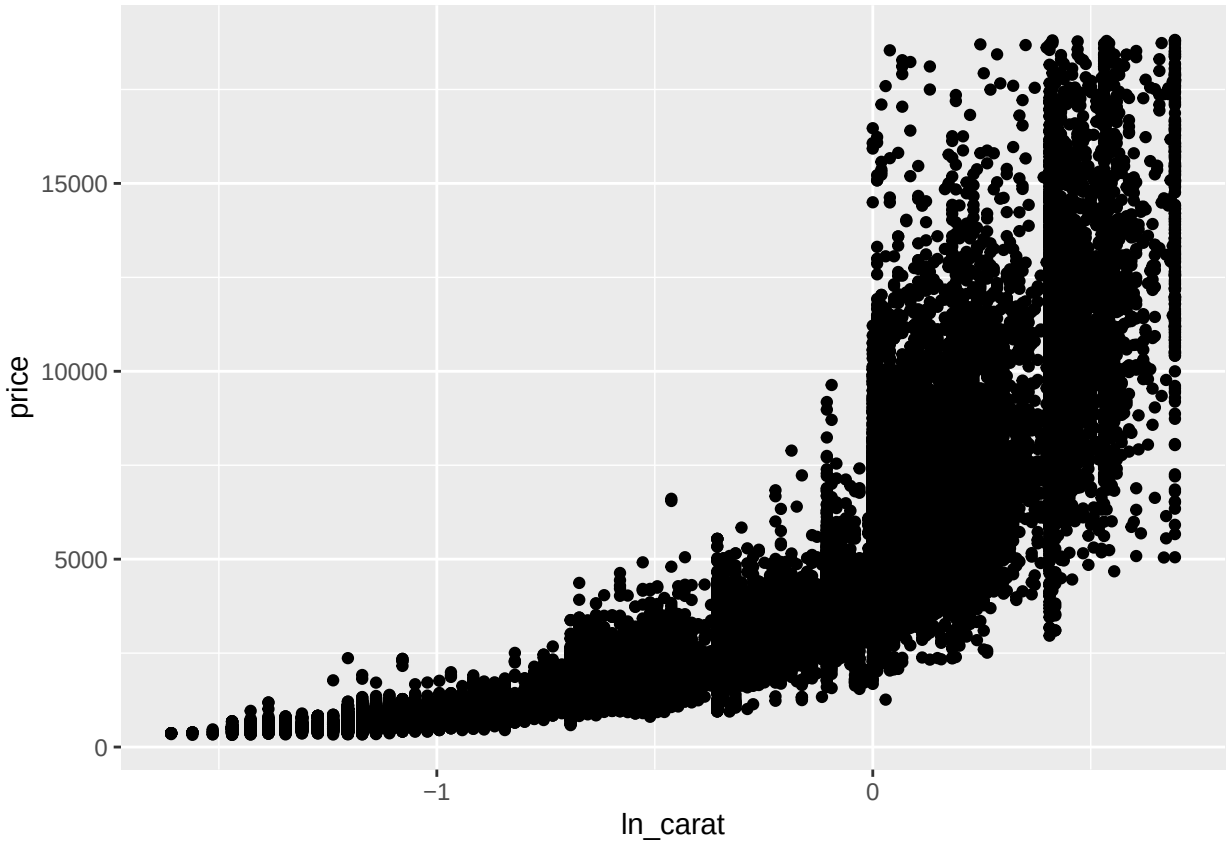
```
## [1] 3395.901
```

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

RMSE will be higher here because clarity is categorical, so it is very hard to model well.

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?

price ~ carat would be a better model, because the error is lower.

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

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

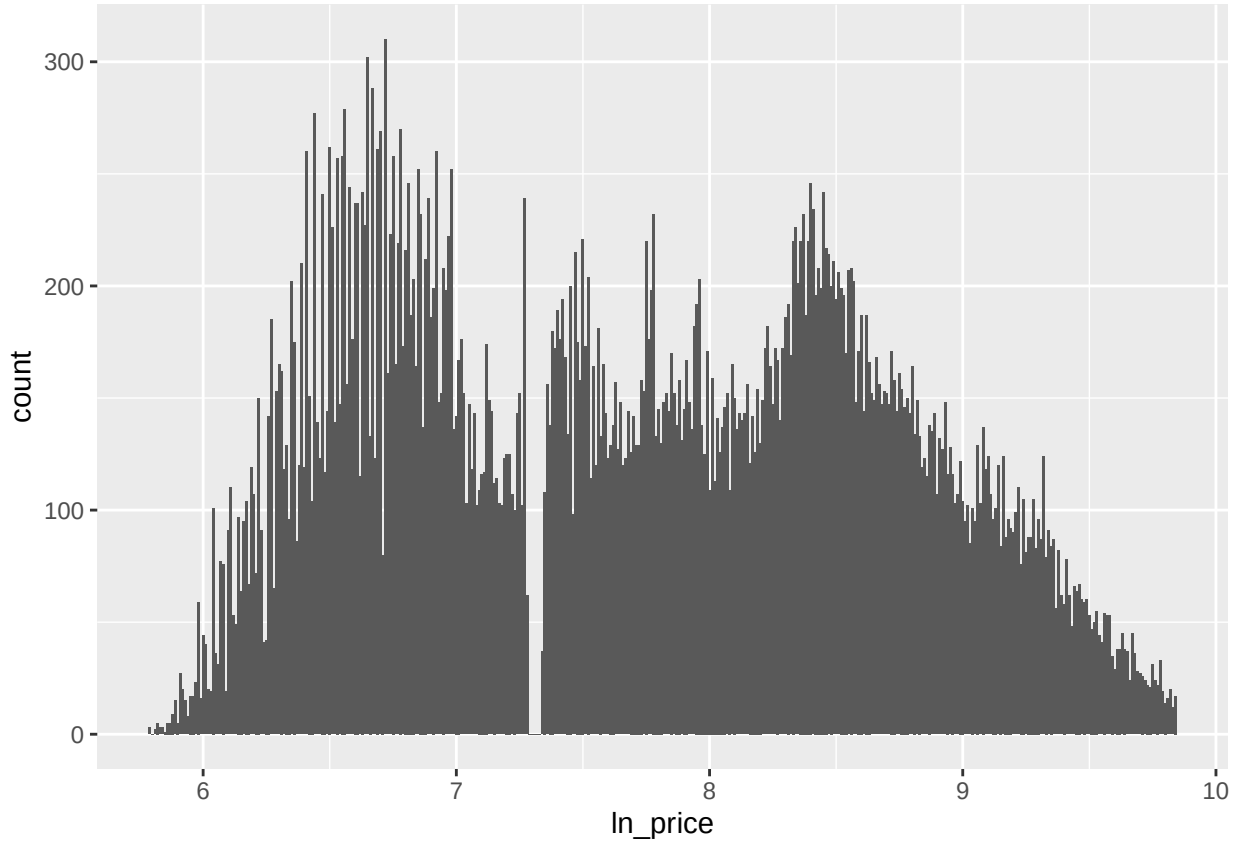
```
## [1] 1451.927
```

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

```
## [1] 1832.005
```

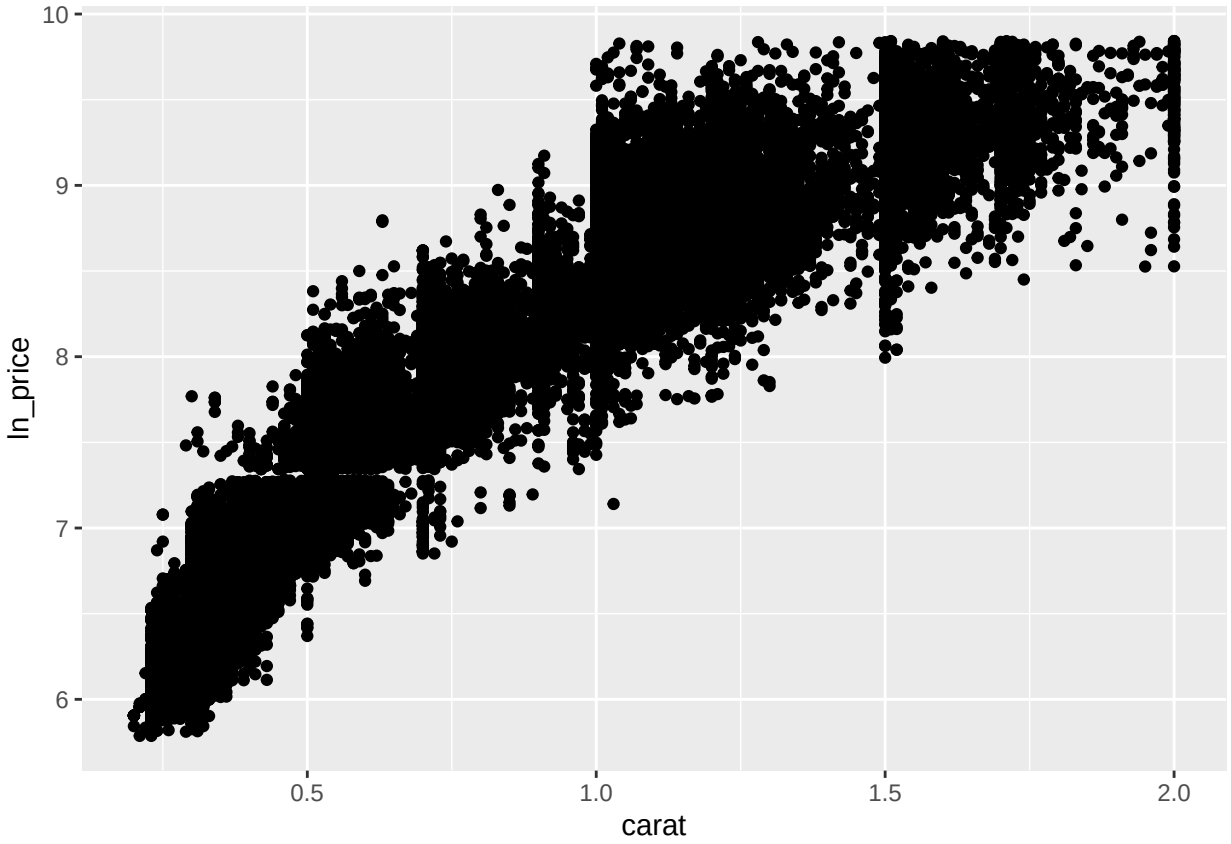
Create a new transformed feature ln_price and plot its estimated density:

```
#diamonds$ln_carat = log(diamonds$carat)
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, becuase it has less error.

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
```

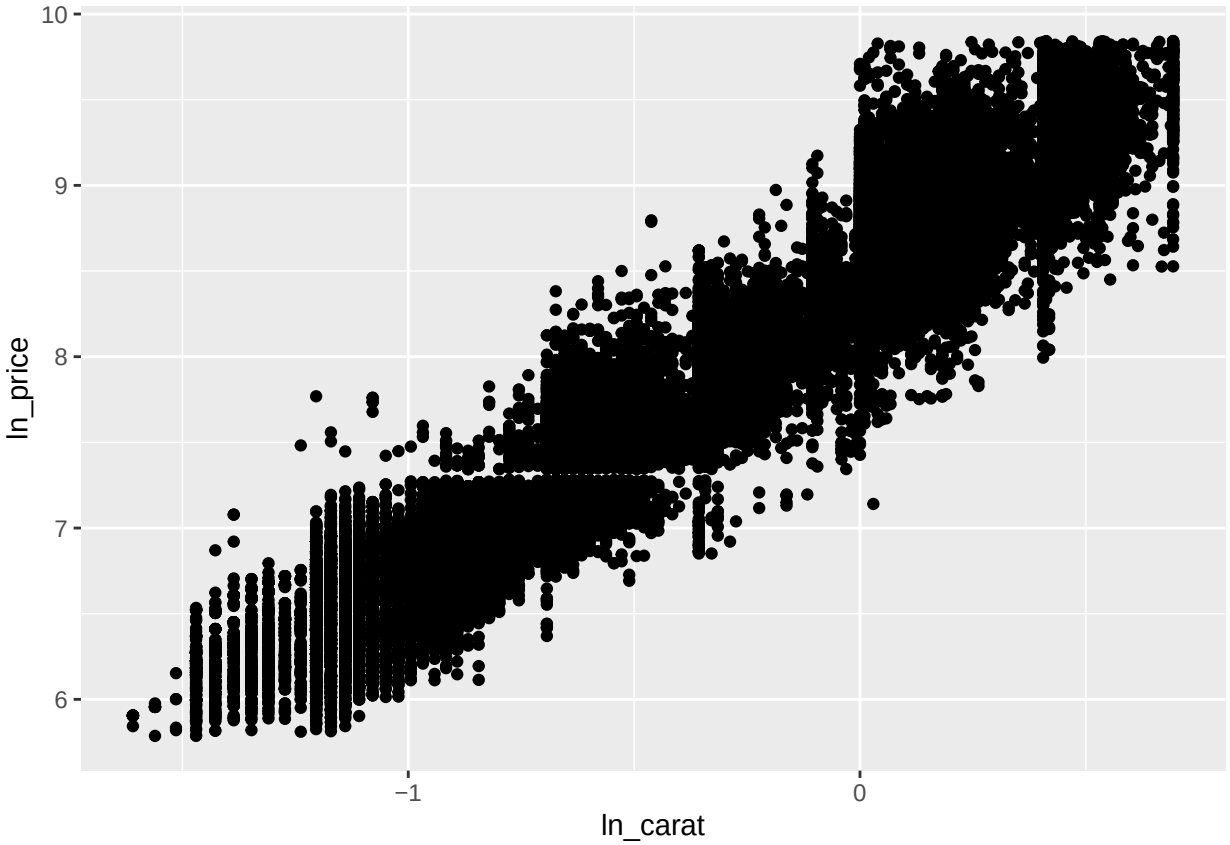
```
summary(mod1)$sigma
```

```
## [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?

Because we are only using one feature and n is huge, so it basically impossible to overfit using one feature with about 50,00 data entries. If you throw a few thousand more features this may not be valid. Plot ln_price vs ln_carat.

```
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? Yes, because it is the most linear.

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

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

```
## [1] 1396.67
```

```
summary(moda)$sigma
```

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

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

```
coef(moda)
```

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

```
#Model A
```

Interpret b_1, the estimated slope of ln_carat.

percent change in price is approximately b_1 * percent change in carat.

Interpret b_0, the estimated intercept.

It is the predicted log price of a diamond that weighs nothing. This is obviously meaningless because there is no such diamond.

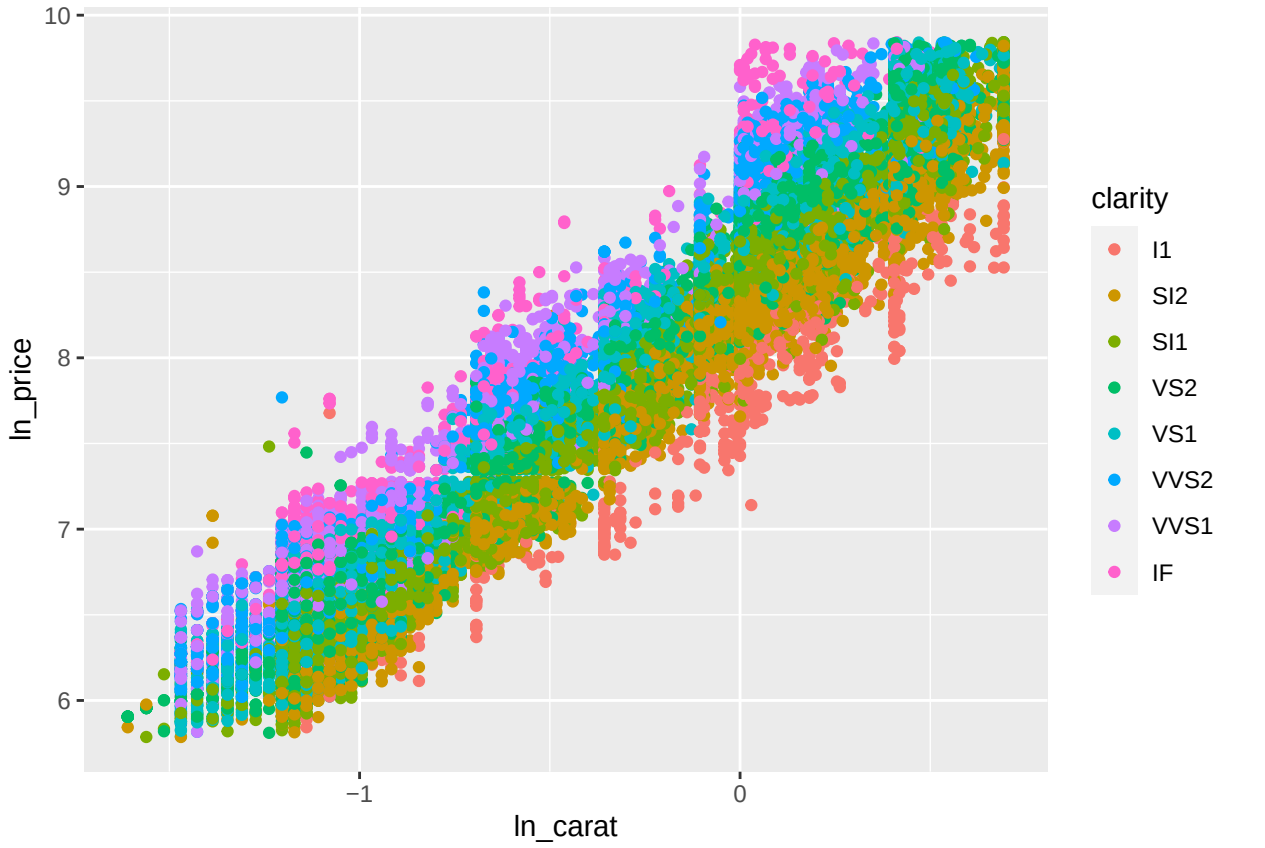
Create other features ln_x, ln_y, ln_z, ln_depth, ln_table.

```
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.

```
#Model B
pacman:: p_load(ggplot2)
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?

#TO-DO

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?

Model C does better becuase you have increased the complexity of H, added more relavent new 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$x > 0 & diamonds$z > 0, ]
modd = lm(ln_price ~ (ln_carat + ln_x + ln_y + ln_z + ln_depth + ln_table) * (clarity + cut + color), di
summary(modd)$sigma
```

```
## [1] 0.1269092
```

Which model does better? Why?

Model D does better because you have increased your dimensionality a lot by interacting a bunch of features. Added a nonorthogonal vector to the dimension of X.

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

```
modd$rank
```

```
## [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?

They are very similar, so it seems like the nonlinearities do not affect it as much. But E is actually a drop better, because it has lower RMSE, but just by a tiny drop.

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
modf = lm (ln_price ~ (poly(carat, 3) + poly(x, 3) + poly(y, 3) + poly(z, 3) + poly(depth, 3) + poly(tab
cut + color), diamonds)
summary(modf)$sigma
```

```
## [1] 0.1082411
```

Which model does better? Why?

Model F is a lot better because you added more dimensions, but it is really overfit.

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

You can expand the set by adding any combination of logs and interactions. We can also square certain features, or express them as polynomials of any degree. The more we do this, though, the more the model will be overfit.

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()
moda = lm(ln_price ~ ln_carat, Dtrain)
insampleRMSE[['A']]  = summary(moda)$sigma
modb = lm(ln_price ~ ln_carat * clarity, Dtrain)
insampleRMSE[['B']]  = summary(modb)$sigma
modc = lm(ln_price ~ ln_carat * (clarity + cut + color), Dtrain)
insampleRMSE[['C']] = summary(modc)$sigma
modd = lm(ln_price ~ (ln_carat + ln_x + ln_y + ln_z + ln_depth + ln_table) * (clarity + cut + color), D
insampleRMSE[['D']] = summary(modd)$sigma
mod_e = lm(ln_price ~ (carat + x + y + z + depth + table) * (clarity + cut + color), Dtrain)
insampleRMSE[['E']] = summary(mod_e)$sigma
modf = lm (ln_price ~ (poly(carat, 3) + poly(x, 3) + poly(y, 3) + poly(z, 3) + poly(depth, 3) + poly(ta
cut + color), Dtrain)
insampleRMSE[['F']] = summary(modf)$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(modf, Dtest))
```

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

```
cbind(
  unlist(insampleRMSE),
  unlist(oosRMSE)
)
```

```
##        [,1]      [,2]
## A 0.2531668 0.2834927
## B 0.1837146 0.1863434
## C 0.1285177 0.1359552
```

```
## D 0.1248787 0.1367282
## E 0.1236078 0.1417794
## F 0.1033050 0.1800725
```

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

They will probably be highly variant since the model is being tested on so few diamonds.

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, la
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:

In a K-fold CV, the data is split into training and testing data K times. Doing this random split K times allows the data to be split randomly and the model to be tested on different data each time. This reduces the variance in the oos error metrics.

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

```
modf = lm (ln_price ~ (poly(carat, 3) + poly(x, 3) + poly(y, 3) + poly(z, 3) + poly(depth, 3) + poly(ta
cut + color), diamonds)
summary(modf)$sigma
```

```
## [1] 0.1082411
```

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

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

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

I think the K-fold CV should help reduce the vaiance of the oos s_e because the testing data is changing each tme.

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.

```
modf = lm (ln_price ~ (poly(carat, 3) + poly(x, 3) + poly(y, 3) + poly(z, 3) + poly(depth, 3) + poly(tab
cut + color), diamonds)
```