title: "Lab 5"
author: Asher Katz
output: pdf_document
date: 4/4/2022

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)
```

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

10

Verify this using code:

```{r}

ncol(diamonds)


```

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

It doesn't make sense because these are categorical dummy variables, if you made them ordinal, in theory you could find a ways to do a polynomial expansion.

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

It doesn't make sense because these are categorical dummy variables, if you made them ordinal, in theory you could find a ways to do a log transformation.

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}
```

```
n = nrow(diamonds)
diamonds = diamonds[diamonds$carat <= 2,]
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.

```{r}
mod = lm(price~carat, diamonds)
summary(mod)$sigma
```

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

```{r}
#TO-DO
mod2 = lm(price~clarity, diamonds)
summary(mod2)$sigma
```

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

Because price is much more related to carat then clarity

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

```{r}
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, because the price~carat has a fairly linear trend, no reason to try a log scale.

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

```{r}
summary(mod)$sigma
summary(mod2)$sigma
```

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

```{r}
#TO-DO
diamonds$ln_price = log(diamonds$price)
ggplot(diamonds) + geom_histogram(aes(x = ln_price), binwidth = 0.01)
```

Now plot it vs carat.

```{r}
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, because the linear trend line fell apart a bit when dealing with larger carat. Loging the price gives it a very linear trend

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

```{r}
mod3 = lm(ln_price ~ carat,diamonds)
yhat = exp(mod3$fitted.values)
SSE = sum( (yhat - diamonds$price)^2 )
RMSE = sqrt(SSE/(n-2))
RMSE
summary(mod)$sigma

#needs to be checked, num not right
```

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 valid?

we are only using one feature?! we aren't over fitting with one

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?

Yes, 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}
#TO-DO
mod4 = lm(ln_price~ ln_carat,diamonds)
yhat = exp(mod4$fitted.values)
SSE = sum( (yhat - diamonds$price)^2 )
RMSE = sqrt(SSE/(n-2))
```

RMSE
summary(mod)$sigma
```

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

```{r}
coef(mod4)
```

Interpret b_1, the estimated slope of ln_carat.

the percent change in price is 1.70 for every 1.00 percent change in carat

Interpret b_0, the estimated intercept.

minimum weight, that being zero, which is meaningless

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}
#Model B
#TO-DO
moda = lm(ln_price~ln_carat, diamonds)
modb = lm(ln_price~ ln_carat * clarity, diamonds)
summary(moda)$sigma
summary(modb)$sigma
```

Which model does better? Why?

B does better because clarity helps to explain some of the change in price, which carat cannot explain alone.

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)

```{r}
modc = lm(ln_price ~ln_carat * (clarity + cut + color), diamonds)
```
```

```
summary(modc)$sigma
summary(modb)$sigma
```

Which model does better? Why?

It does better because all of the features explain some of the change in price which carat cannot alone.

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).

```{r}
#Model D
#TO-DO
diamonds = diamonds[diamonds$x >0 & diamonds$z >0, ]

diamonds[is.infinite(diamonds$ln_z), ]
modd = lm(ln_price ~ (ln_carat + ln_x + ln_y + ln_z + ln_depth + ln_table) * (clarity + cut + color), diamonds)
summary(modd)$sigma
summary(modc)$sigma

```

Which model does better? Why?

Simply, model D has more complex features and this will fit the data better then model C

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

```{r}
#TO-DO
modd$rank
ncol( model.matrix( ~ (ln_carat + ln_x + ln_y + ln_z + ln_depth + ln_table) * (clarity + cut + color), diamonds ))
```

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).

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

summary(modd)$sigma
summary(mod_e)$sigma
```

Which model does better? Why?

They're about the same, if everything is linear, the RMSE will be the same if everything is Log'd

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!

```{r}
#Model F
modf = 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(modf)$sigma
summary(mod_e)$sigma
```

Which model does better? Why?

Model F is better because we're just overfitting at this point and F is more overfit

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

sinosoidal functions maybe exponentials, products of more than one feature

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:

```{r}
#TO-DO
k = 10
set.seed(1996)
D = diamonds[ sample(1:nrow(diamonds), 2000), ]
Dtrain = D[ 1 :((1 - 1/k)*2000), ]
Dtest = D[((1 - 1/k)* 2000 + 1) : 2000, ]
```

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!

```{r}
insampleRMSE = list()
oosRMSE = list()

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

```
oosRMSE[['A']] = sd(Dtest$ln_price - predict(moda, Dtest))
oosRMSE[['B']] = sd(Dtest$ln_price - predict(moda, Dtest))
oosRMSE[['C']] = sd(Dtest$ln_price - predict(moda, Dtest))
oosRMSE[['D']] = sd(Dtest$ln_price - predict(moda, Dtest))
oosRMSE[['E']] = sd(Dtest$ln_price - predict(moda, Dtest))
oosRMSE[['F']] = sd(Dtest$ln_price - predict(moda, Dtest))

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

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

That our numbers will vary depending on where our test was sampled from

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.

```{r}
K= 10
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)
```

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

It will cycle the all of the data through the test set, using the remaining data at the time as training data. This will accomplish a much lower variance as we will have the vector of all error metrics from each test set, which will give us an average error

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

```{r}
K = 10
folds_vec = cut(modf$fitted.values, breaks = quantile(modf$fitted.values, seq(0, 1, length.out = K + 1)), include.lowest = TRUE, labels = FALSE)
head(folds_vec, 200)
s_e = sd(modf$residuals)/sqrt(200)
s_s_e = sqrt( (((sum(s_e - mean(s_e)) ))^2)/(k-1) )
s_e
s_s_e
```

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

Yes, it will cycle the all of the data through the test set, using the remaining data at the time as training data. This will accomplish a much lower variance as we will have the vector of all error metrics from each test set, which will give us an average error.

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.

```{r}
#TO-DO
K = 10
folds_vec = cut(modf$fitted.values, breaks = quantile(modf$fitted.values, seq(0, 1, length.out = K + 1)), include.lowest = TRUE, labels = FALSE)
head(folds_vec, 200)
s_e = sd(modf$residuals)/sqrt(length(modf$residuals))
s_e_s_F = sqrt( (((sum(s_e - mean(s_e)) ))^2)/(k-1) )
ggplot(data.frame(s_e_s_F = s_e_s_F)) + geom_histogram(aes(x = s_e_s_F))
```