HW03p, part II

[Your Name Goes Here]

April 13, 2018

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

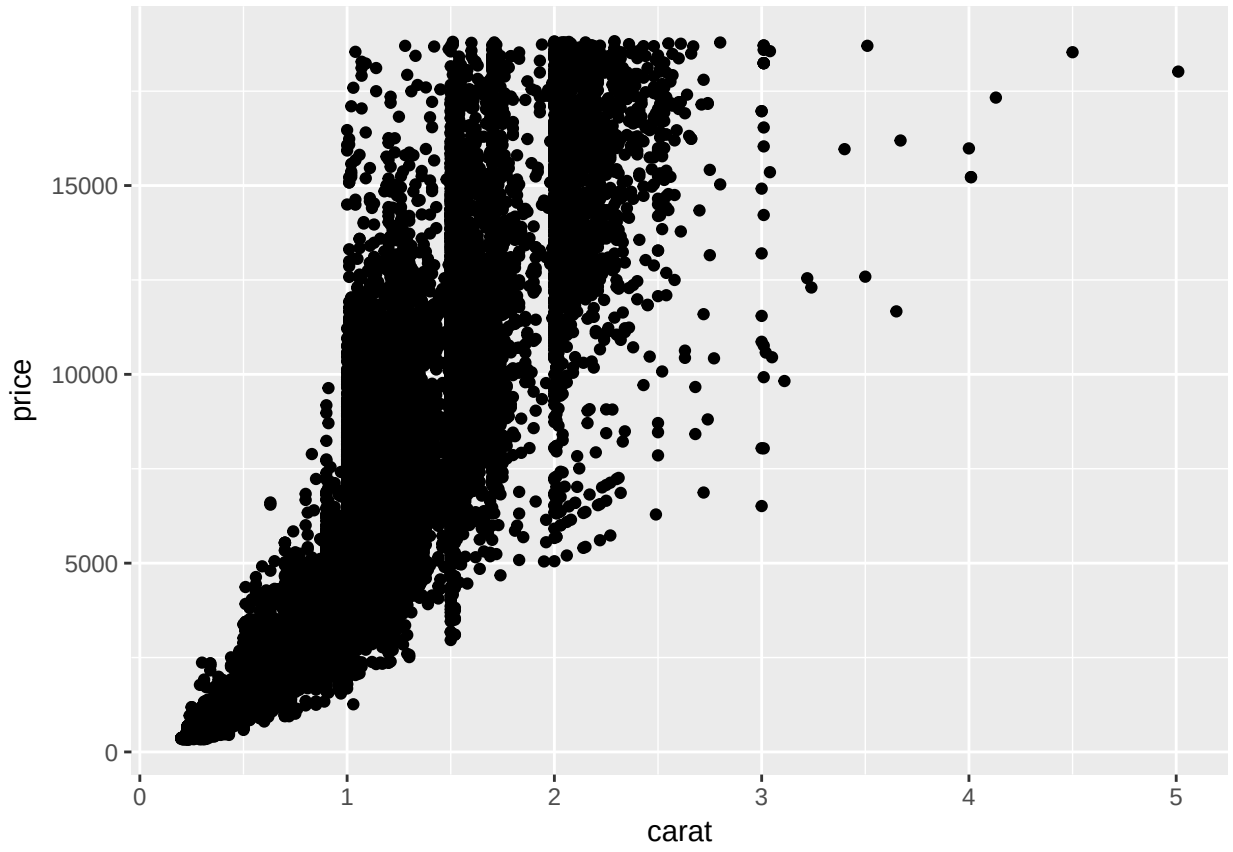
```
library(pacman)
pacman::p_load(ggplot2)
```

```
?diamonds
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 ...
```

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

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



** TO-DO # Price vs. carat doesn't quite appear linear as there appears to be a slight curve upwards in the graph. However, the curve is slight, so a linear model should do fine.

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

```
#TO-DO
square_carat = lm(price ~ poly(carat, 2) + as.character(cut) + as.character(color) + as.character(clari
names(square_carat$coefficients) = c("(Intercept)", "carat", "carat^2", "cut Good", "cut Ideal", "cut P
square_carat
```

```
##
## Call:
## lm(formula = price ~ poly(carat, 2) + as.character(cut) + as.character(color) +
##     as.character(clarity) + depth + table + x + y + z, data = diamonds)
##
## Coefficients:
##   (Intercept)          carat        carat^2       cut Good      cut Ideal
##      21804.31     1536647.48      -76105.46         538.33         807.52
##   cut Premium  cut Very Good        color E        color F        color G
##        747.70         678.32        -209.44        -284.55        -496.85
##       color H        color I        color J     clarity IF    clarity SI1
##       -997.60       -1469.25       -2357.80        5243.52        3565.41
##   clarity SI2    clarity VS1    clarity VS2   clarity VVS1   clarity VVS2
##       2605.54        4475.44        4163.35        4904.23        4843.80
##          depth          table              x              y              z
##       -116.23         -36.37       -2123.01         -23.46         -83.11
```

What is b, R^2 and the RMSE?

```
#TO-DO
list(
     "b = " = coef(square_carat),
     "R^2 = " = summary(square_carat)$r.squared,
     "RMSE = " = summary(square_carat)$sigma
)
```

```
## $`b = `
##   (Intercept)         carat       carat^2      cut Good     cut Ideal
##   21804.30931 1536647.48327  -76105.45945     538.33407     807.51616
##   cut Premium cut Very Good       color E       color F       color G
##     747.69518     678.31993    -209.43992    -284.54706    -496.84716
##       color H       color I       color J    clarity IF   clarity SI1
##    -997.60127   -1469.25151   -2357.79746    5243.52276    3565.41193
##   clarity SI2   clarity VS1   clarity VS2  clarity VVS1  clarity VVS2
##    2605.54013    4475.44424    4163.34947    4904.22750    4843.80493
##         depth         table             x             y             z
##    -116.22729     -36.37384   -2123.00617     -23.46172     -83.11272
##
## $`R^2 = `
## [1] 0.9214777
##
## $`RMSE = `
## [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.

**TO-DO # It's the same as above, but here, the carat^2 coefficient measures how much the price goes up on average for each 1 SQUARE increase in carat. i.e. the average increase in price as carat goes from 1 to 4, or 4 to 9, or 9 to 16.

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

**TO-DO # This is a slight but significant improvement over our model in #4, as R^2 has increased to .9215 from .9198, and RMSE has decreased from 1130 to 1118. It's significant because our p+1 still does not approach n. This model is probably a signficant improvement because the graph of price vs. carat looks slightly parabolic.

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

```
#TO-DO
sqcarat_predict = function(data) {
     predict(square_carat, data)
}
sqcarat_predict(diamonds[1, ])
```

```
##        1
## -1159.29
```

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

```
#TO-DO
color_poly = lm(price ~ poly(color, 2), diamonds)
```

```
## Warning in mean.default(x): argument is not numeric or logical: returning
## NA
```

```
## Warning in Ops.ordered(x, xbar): '-' is not meaningful for ordered factors
```

```
## Error in qr.default(X): NA/NaN/Inf in foreign function call (arg 1)
```

Why did this throw an error?

**TO-DO # It throws an error because color is not a numeric value, therefore it cannot be squared. i.e. what's "Fair"^2?

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.

```
#TO-DO
diamonds_mm = model.matrix(price ~ carat + as.character(cut) + as.character(color) + as.character(clari
colnames(diamonds_mm) = c("(Intercept)", "carat", "cut Good", "cut Ideal", "cut Premium", "cut Very Goo
X = as.matrix(diamonds_mm)
y = diamonds$price
b = solve(t(X) %*% X) %*% t(X) %*% y
b
```

```
##                       [,1]
## (Intercept)    2184.477351
## carat         11256.978308
## cut Good        579.751446
## cut Ideal       832.911845
## cut Premium     762.143950
## cut Very Good   726.782591
## color E        -209.118085
## color F        -272.853832
## color G        -482.038904
## color H        -980.266675
## color I       -1466.244474
## color J       -2369.398063
## clarity IF     5345.102246
## clarity SI1    3665.472080
## clarity SI2    2702.586294
## clarity VS1    4578.397915
## clarity VS2    4267.223565
## clarity VVS1   5007.759045
## clarity VVS2   4950.814072
## depth           -63.806100
## table           -26.474085
## x             -1008.261098
## y                 9.608886
## z               -50.118891
```

What is b, R^2 and the RMSE?

```
#TO-DO
# R^2:
SSE_0 = sum((y-mean(y))^2)
yhat = X %*% b
e = y-yhat
SSE = sum((y-yhat)^2)
r_squared = (SSE_0 - SSE) / SSE_0

# RMSE:
p = ncol(diamonds_mm) - 1
n = nrow(diamonds_mm)
```

```
MSE = (1/(n-(p+1))) * SSE
RMSE = sqrt(MSE)

list(
     "b = " = b,
     "R^2 = " = r_squared,
     "RMSE = " = RMSE
)
```

```
## $`b = `
##                       [,1]
## (Intercept)    2184.477351
## carat         11256.978308
## cut Good        579.751446
## cut Ideal       832.911845
## cut Premium     762.143950
## cut Very Good   726.782591
## color E        -209.118085
## color F        -272.853832
## color G        -482.038904
## color H        -980.266675
## color I       -1466.244474
## color J       -2369.398063
## clarity IF     5345.102246
## clarity SI1    3665.472080
## clarity SI2    2702.586294
## clarity VS1    4578.397915
## clarity VS2    4267.223565
## clarity VVS1   5007.759045
## clarity VVS2   4950.814072
## depth           -63.806100
## table           -26.474085
## x             -1008.261098
## y                 9.608886
## z               -50.118891
##
## $`R^2 = `
## [1] 0.9197915
##
## $`RMSE = `
## [1] 1130.094
```

Are they the same as in #4?

**TO-DO # YES!

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.

```
#TO-DO
qrx = qr(X)
Q = qr.Q(qrx)
R = qr.R(qrx)

b = solve(R) %*% t(Q) %*% y
b
```

```
##                       [,1]
## (Intercept)    2184.477350
## carat         11256.978307
## cut Good        579.751446
## cut Ideal       832.911845
## cut Premium     762.143950
## cut Very Good   726.782591
## color E        -209.118085
## color F        -272.853832
## color G        -482.038904
## color H        -980.266675
## color I       -1466.244474
## color J       -2369.398063
## clarity IF     5345.102246
## clarity SI1    3665.472080
## clarity SI2    2702.586294
## clarity VS1    4578.397915
## clarity VS2    4267.223565
## clarity VVS1   5007.759045
## clarity VVS2   4950.814072
## depth           -63.806100
## table           -26.474085
## x             -1008.261098
## y                 9.608886
## z               -50.118891
```

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

```
#TO-DO

yhat = as.numeric(X %*% b)
e = y-yhat

# I think it's just this, but I'm not sure. Beware before running!:
# H = Q %*% t(Q)
# maybe_yhat = H[1:6, ] %*% y

head(yhat)
```

```
## [1] -1346.3643  -664.5954   211.1071  -830.7372 -3459.2242 -1380.4876
```

```
head(e)
```

```
## [1] 1672.3643  990.5954  115.8929 1164.7372 3794.2242 1716.4876
```

```
#H[1:20, 1:6]
#maybe_yhat
```

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

```
#TO-DO
# (a) $\hat{y}$ and $e$ sum to the vector $y$ (the prices in the original dataframe),
```

```r
yhat_plus_e = yhat + e
all.equal(yhat_plus_e, y)
```

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

```r
# (b) $\hat{y}$ and $e$ are orthogonal
yhat %*% e
```

```
##          [,1]
## [1,] 1.236695
```

```r
yhat_dot_e = as.numeric(yhat %*% e)
yhat_dot_e
```

```
## [1] 1.236695
```

```r
all.equal(yhat_dot_e, 0)
```

```
## [1] "Mean relative difference: 1"
```

```r
# I think this comes out to not exactly zero because of numerical error?
################################################################################
#????????????????????????????????????????????????????????????????????????????

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

# all.equal(H %*% e, 0)

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

# all.equal(H %*% yhat, yhat)

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

# I = diag(53940)
# all.equal((I-H) %*% yhat, 0)

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

# SSE? + SSR = SST
ybar = mean(y)
SSE = sum((y-yhat)^2)
SSM = sum((yhat-ybar)^2)
SST = sum((y-ybar)^2)
all.equal(SSE + SSM, SST)
```

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

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

```r
#TO-DO

poly5_mod = lm(price ~ poly(carat, 5) + as.character(cut) + as.character(color) + as.character(clarity)
poly5_mod
```

```
##
## Call:
```

```
## lm(formula = price ~ poly(carat, 5) + as.character(cut) + as.character(color) +
##     as.character(clarity) + poly(depth, 5) + poly(table, 5) +
##     poly(x, 5) + poly(y, 5) + poly(z, 5), data = diamonds)
##
## Coefficients:
##                (Intercept)             poly(carat, 5)1
##                      722.3                   683720.7
##            poly(carat, 5)2             poly(carat, 5)3
##                   -61002.3                    66981.3
##            poly(carat, 5)4             poly(carat, 5)5
##                    10301.0                    16016.3
##      as.character(cut)Good      as.character(cut)Ideal
##                      237.8                      410.8
##   as.character(cut)Premium  as.character(cut)Very Good
##                      347.8                      301.5
##       as.character(color)E       as.character(color)F
##                     -212.1                     -265.1
##       as.character(color)G       as.character(color)H
##                     -500.8                    -1011.2
##       as.character(color)I       as.character(color)J
##                    -1524.0                    -2421.9
##    as.character(clarity)IF    as.character(clarity)SI1
##                     4792.1                     3176.1
##   as.character(clarity)SI2    as.character(clarity)VS1
##                     2260.0                     4050.5
##   as.character(clarity)VS2   as.character(clarity)VVS1
##                     3742.8                     4454.6
##  as.character(clarity)VVS2             poly(depth, 5)1
##                     4407.6                    -2214.6
##            poly(depth, 5)2             poly(depth, 5)3
##                    -8369.1                      235.7
##            poly(depth, 5)4             poly(depth, 5)5
##                     5519.7                      790.9
##            poly(table, 5)1             poly(table, 5)2
##                    -5336.8                     -856.2
##            poly(table, 5)3             poly(table, 5)4
##                    -1746.4                    -1003.0
##            poly(table, 5)5                 poly(x, 5)1
##                     -317.8                   -166914.0
##                poly(x, 5)2                 poly(x, 5)3
##                  -333184.3                  -122051.0
##                poly(x, 5)4                 poly(x, 5)5
##                   -35049.6                   -57313.4
##                poly(y, 5)1                 poly(y, 5)2
##                   295450.8                    42533.2
##                poly(y, 5)3                 poly(y, 5)4
##                  -340113.5                   389000.8
##                poly(y, 5)5                 poly(z, 5)1
##                   215370.8                   123695.2
##                poly(z, 5)2                 poly(z, 5)3
##                   -22949.8                     -924.2
##                poly(z, 5)4                 poly(z, 5)5
##                    41852.4                    47744.2
```

```
#summary(poly5_mod)
```

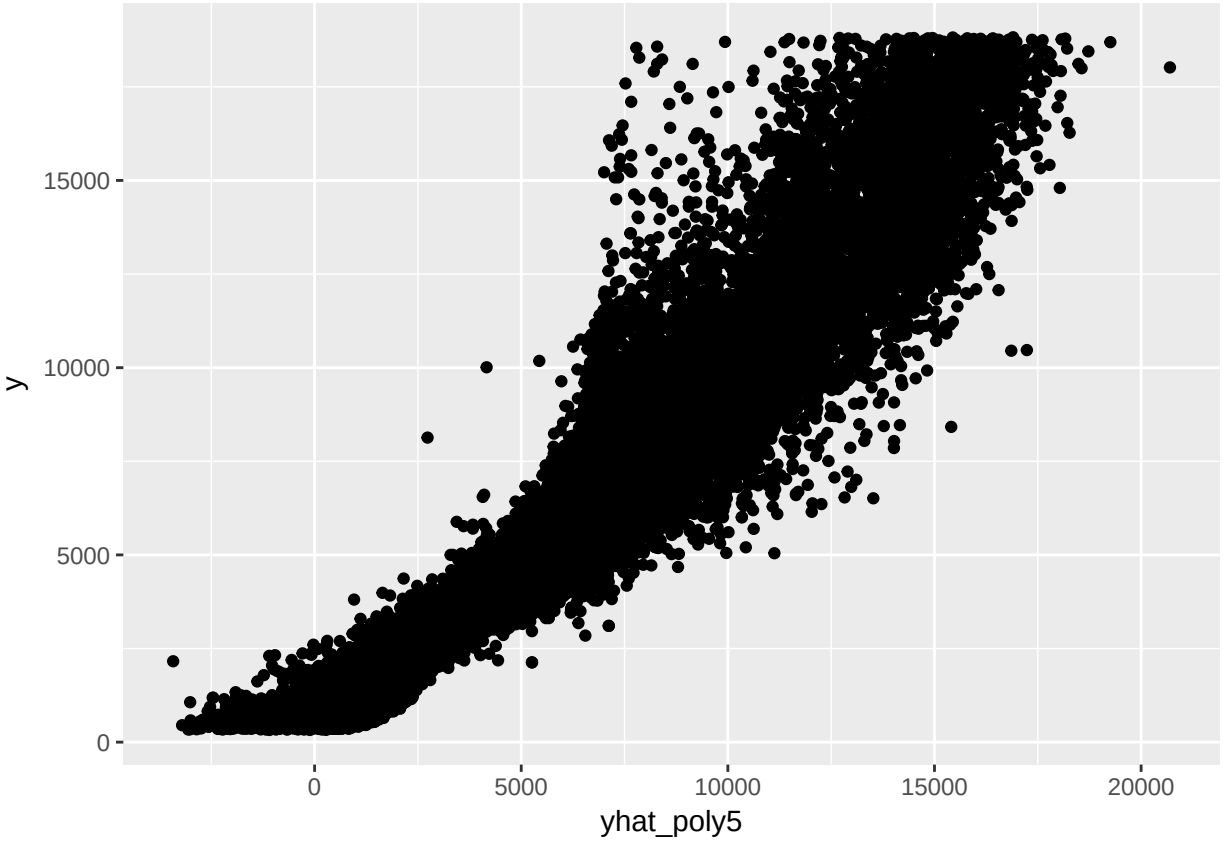
Report R^2, RMSE, the standard error of the residuals (s_e) but you do not need to report b.

```
#TO-DO
n = nrow(diamonds)
p = length(poly5_mod$coefficients)
list(
     "R^2 = " = summary(poly5_mod)$r.squared,
     "RMSE = " = summary(poly5_mod)$sigma,
     "s_e = " = sqrt((1/(n-(p+1)))*sum((summary(poly5_mod)$residuals)^2))
)
```

```
## $`R^2 = `
## [1] 0.9334179
##
## $`RMSE = `
## [1] 1029.864
##
## $`s_e = `
## [1] 1029.874
```

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

```
#TO-DO
yhat_poly5 = predict(poly5_mod, diamonds)
dummy_frame = data.frame(y = y, yhat_poly5 = yhat_poly5)
ggplot(dummy_frame, aes(x = yhat_poly5, y = y)) +
          geom_point()
```



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

```
#TO-DO
residuals_poly5 = summary(poly5_mod)$residuals
big_error_list = residuals_poly5[ residuals_poly5 > 1000 ]
length(big_error_list)
```

```
## [1] 6040
```

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.

**TO-DO # Using the Empirical Rule, our RMSE gives us a 95% confidence window of about $4000 (±$2000) around our price estimate. If you're looking to make a bid on a diamond or wtv people do with this data, you probably want more confidence that you're getting a good deal, which means a smaller CI.

What is the degrees of freedom in this model?

```
#TO-DO
length(poly5_mod$coefficients)
```

```
## [1] 48
```

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

**TO-DO # Yes, I do think it's close. With 50,000 observations, there will be little estimation error in the 48 parameters.

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

**TO-DO # Yeah, it's probably pretty close because it has an R^2 value of .93. That means the best possible model using these features could only, at best, explain 7% more variance, and some of that 7% is probably just due to ignorance (i.e. the whims of diamond pricers).

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

** TO-DO # I could add interaction terms.

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.

** TO-DO # The fluctuation of the local or international markets for diamonds would probably explain some more of the variance. Also other trends in diamonds might account for some more of the variance. Also data on sales at the individual diamond sellers probably affects prices.

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

```
#TO-DO
n = nrow(diamonds)
k = 5

y = diamonds$price
test_indices = sample(1:n, n*(1/k))
train_indices = setdiff(1:n, test_indices)
diamonds_train = diamonds[train_indices, ]
diamonds_test = diamonds[test_indices, ]

test_poly5_mod = lm(price ~ poly(carat, 5) + as.character(cut) + as.character(color) + as.character(cla
yhat_test = predict(test_poly5_mod, diamonds_test)
sd(yhat_test - y[test_indices])
```

```
## [1] 1109.681
```

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?

** TO-DO # The OOS Se of the residuals is usually about the same as the in-sample estimate- a little more than 1000, but occasionally balloons up to several million.

Extra-credit: validate the model via cross validation.

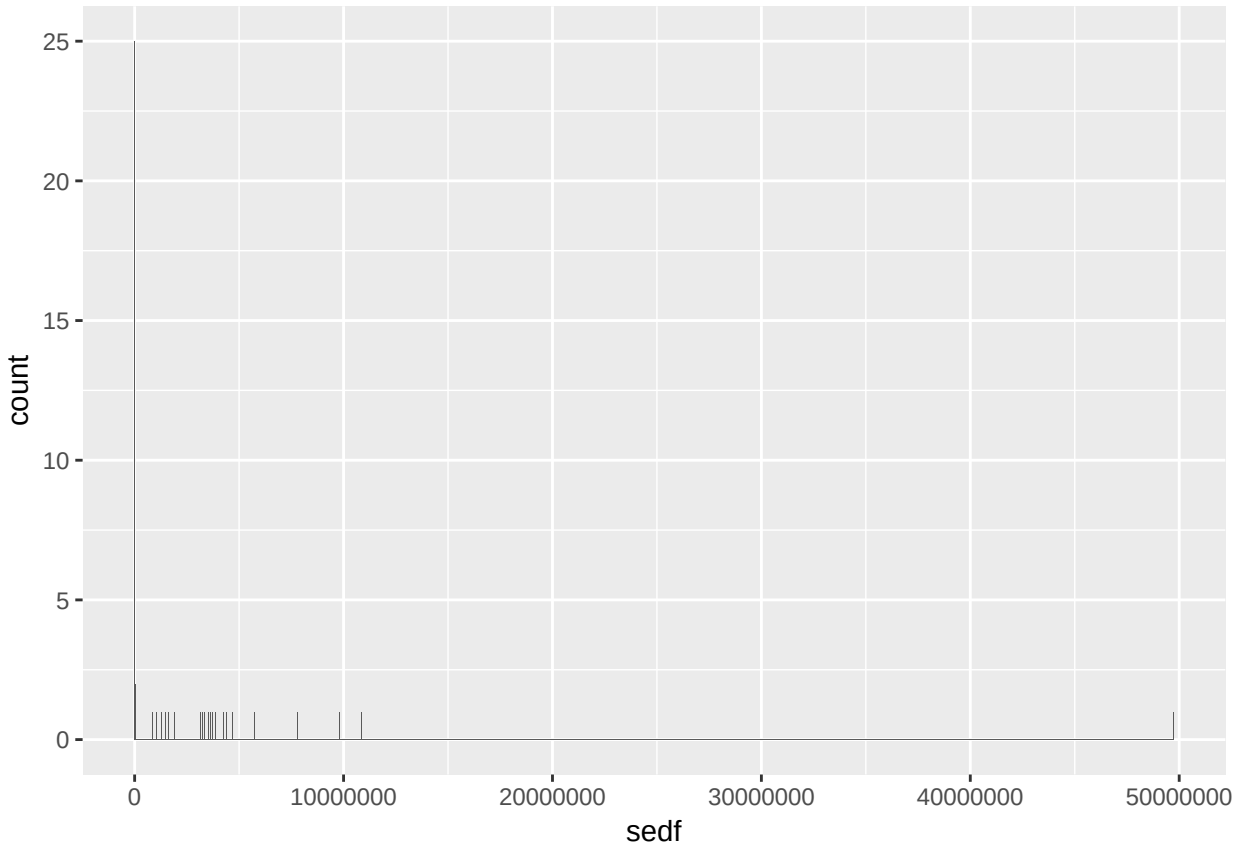
```
#TO-DO if you want extra credit

n = nrow(diamonds)
k = 5
y = diamonds$price
std_errors = numeric(0)

for (i in 1:50) {
     test_indices = sample(1:n, n*(1/k))
     train_indices = setdiff(1:n, test_indices)
     diamonds_train = diamonds[train_indices, ]
     diamonds_test = diamonds[test_indices, ]

     test_poly5_mod = lm(price ~ poly(carat, 5) + as.character(cut) +
          as.character(color) + as.character(clarity) +
          poly(depth, 5) + poly(table, 5) + poly(x, 5) +
          poly(y, 5) + poly(z, 5), diamonds_train)
     yhat_test = predict(test_poly5_mod, diamonds_test)
     std_errors[i] = sd(yhat_test - y[test_indices])
}
options(scipen=999)
sedf = as.data.frame(std_errors)
sedf
```

```
##      std_errors
## 1   1899246.722
## 2      1012.871
## 3      1081.983
## 4      1046.892
## 5      1029.445
## 6      1069.829
## 7  49719962.082
## 8   3517818.724
## 9      1034.073
## 10     1073.820
## 11  4677201.691
## 12  1052762.316
## 13     1044.902
## 14     1157.769
## 15     1033.339
## 16     1047.423
## 17     1038.020
## 18  1621341.446
## 19  3263636.268
## 20     1131.226
## 21     1047.104
```

```
## 22  9788950.066
## 23     1020.998
## 24     1007.731
## 25     1107.235
## 26  3634131.863
## 27  4257813.439
## 28  7810783.804
## 29 10839187.735
## 30  1274313.269
## 31  3740322.154
## 32  3871344.731
## 33  5745036.815
## 34     1076.570
## 35  3151686.939
## 36  4389376.628
## 37   837896.927
## 38     1069.201
## 39     1054.555
## 40  1483374.065
## 41     1167.104
## 42  3327462.651
## 43    42037.117
## 44     1032.095
## 45    46415.735
## 46    48782.153
## 47    34360.321
## 48     1021.044
## 49     1070.278
## 50     1042.521
```

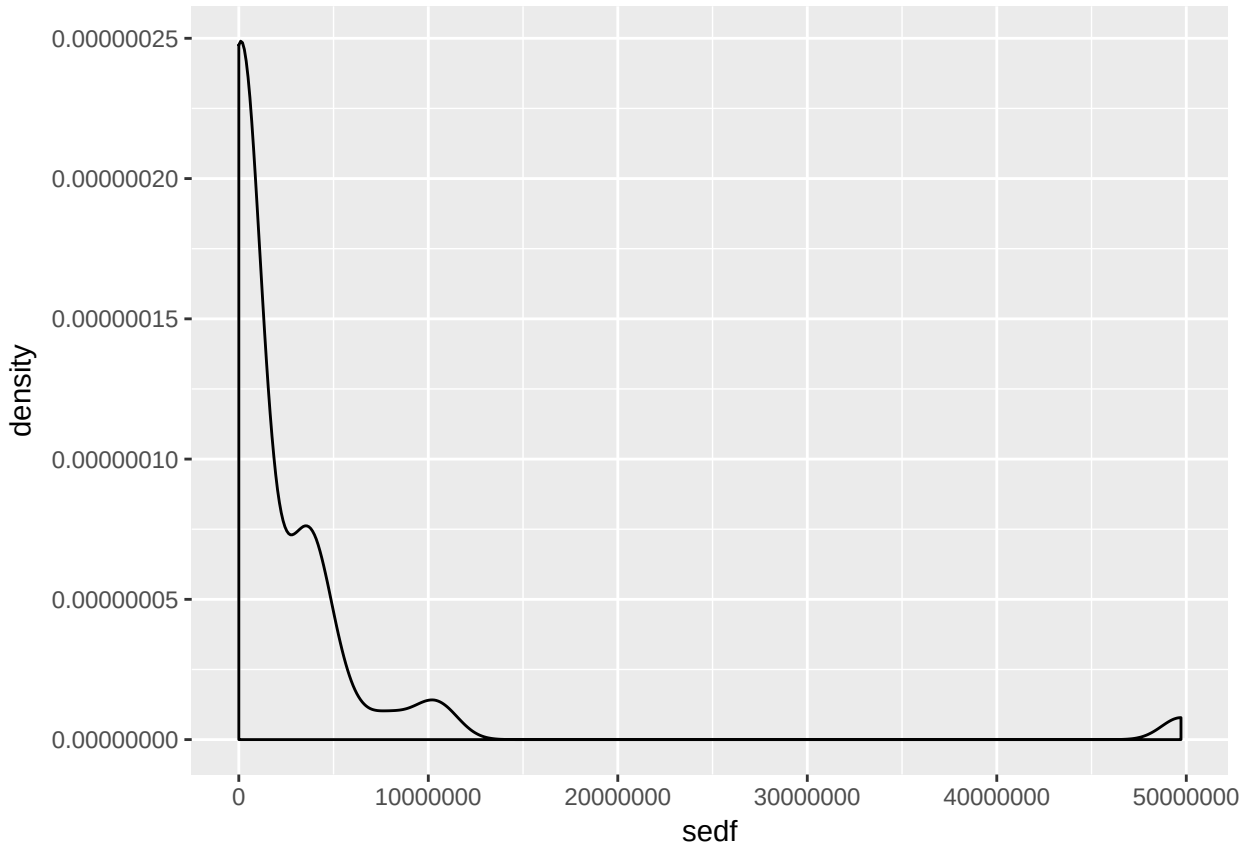
```
#ggplot(std_errors, aes(std_errors)) + geom_histogram()
ggplot(sedf, aes(sedf)) + geom_histogram(binwidth = 10000)
```

```
## Don't know how to automatically pick scale for object of type data.frame. Defaulting to continuous.
```




```
ggplot(sedf, aes(sedf)) + geom_density()
```

```
## Don't know how to automatically pick scale for object of type data.frame. Defaulting to continuous.
```



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

** TO-DO # Yes, depending on which sample happened to go into the test set, the single validation results varied wildly. Most were around 1000, the in-sample s_e, but then there'd be a few that were in the millions- very different.

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:

```
#TO-DO
n = nrow(X)
k = 5

test_indices = sample(1:n, n*(1/k))
train_indices = setdiff(1:n, test_indices)
X_train = as.matrix(X[train_indices, ])
y_train = y[train_indices]
X_test = as.data.frame(X[test_indices, ])
y_test = y[test_indices]

mod = lm(y_train ~ X_train)
y_hat_oos = predict(mod, X_test)
```

```
## Warning: 'newdata' had 20 rows but variables found have 80 rows
```

```
options(scipen=999)
sd(y_hat_oos - y_test)
```

```
## [1] 2.573319
```