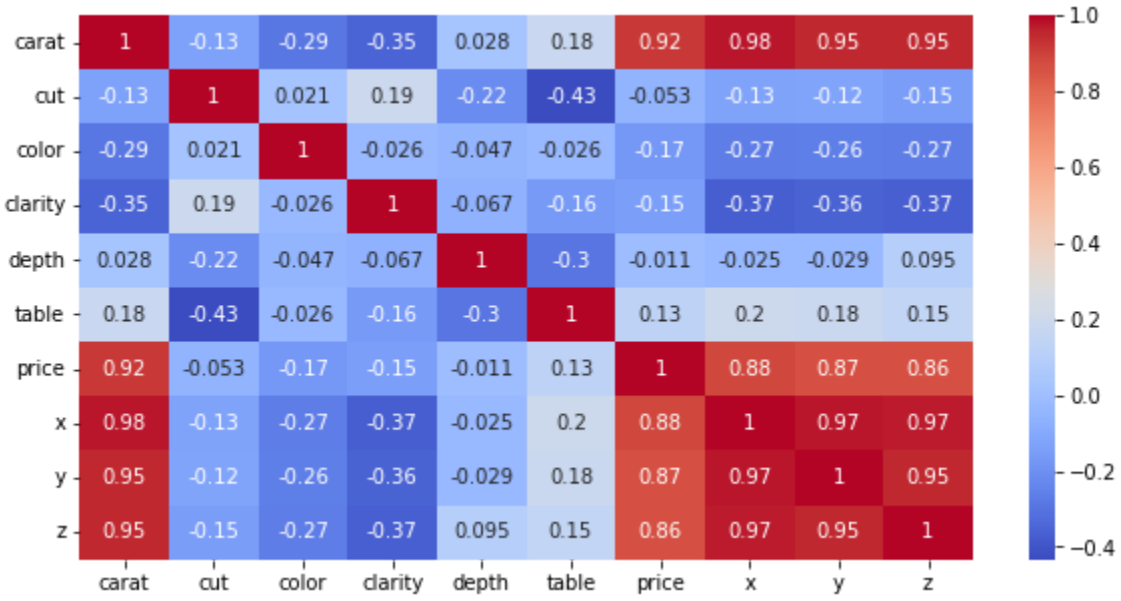
ECE 219:

Project 4: Regression Analysis and Define Your Own Task!

Sarah Wilen swilen1@g.ucla.edu UID: 305070897	Marco Venegas venmarco310@g.ucla.edu UID: 80546121	Henry Peters hpeters@g.ucla.edu UID: 205704526

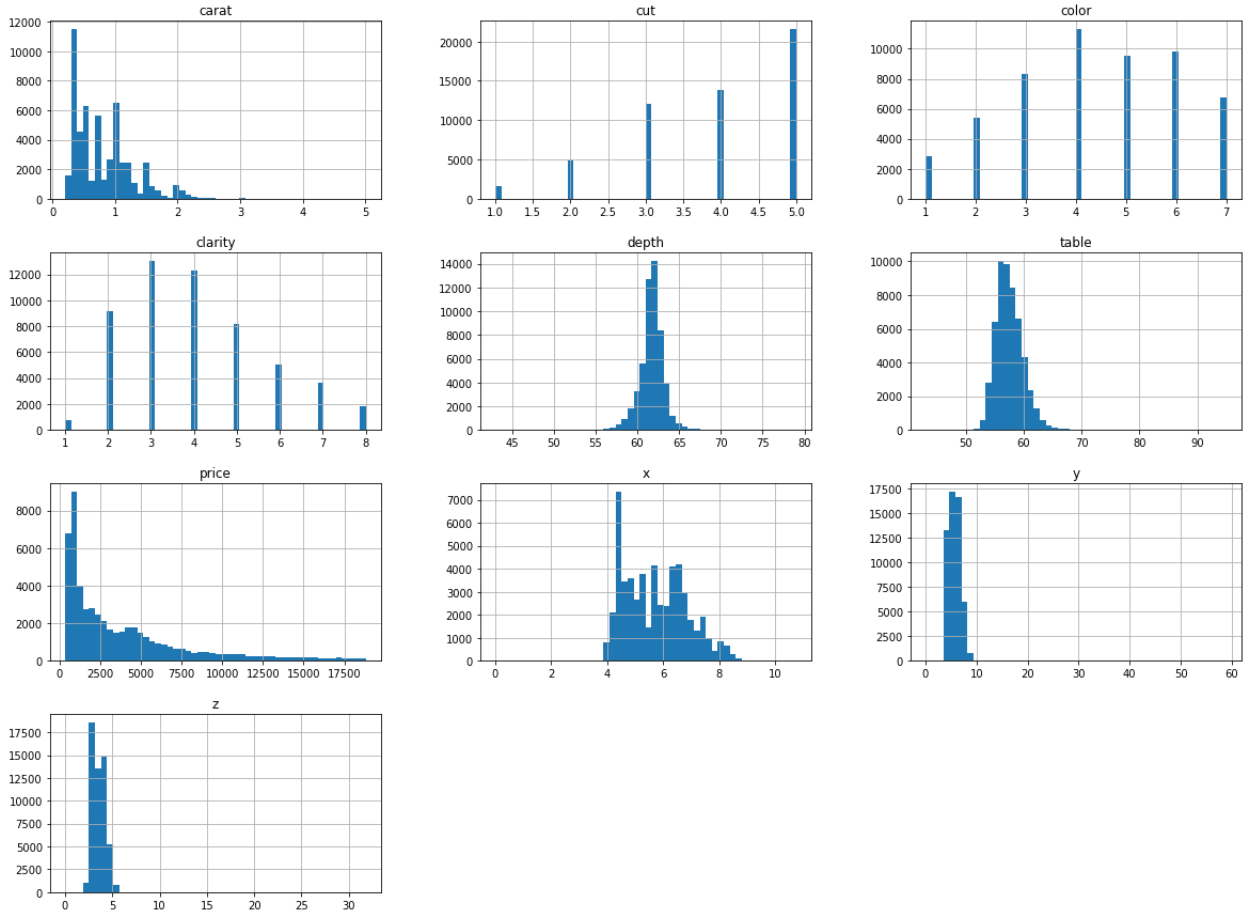
Question 1

Question 1.1



From the Pearson correlation matrix, the features with the highest positive correlation with price in order are carat, x, y, and z. This makes sense because the heavier the diamond (and the larger the diamond), the more expensive it will be. Interestingly, but not so surprisingly, cut, color, clarity, and depth have a slight negative correlation. You would think the better these features, the higher the price. However, I notice from research that depth actually does not positively correlate with the value of the diamond because the most optimal depth is somewhere between 50-60%. Cut, color, and clarity may be slightly negatively correlated because it could be that the larger the carat of the diamond, the less likely it is to find a better color and clarity.

Question 1.2



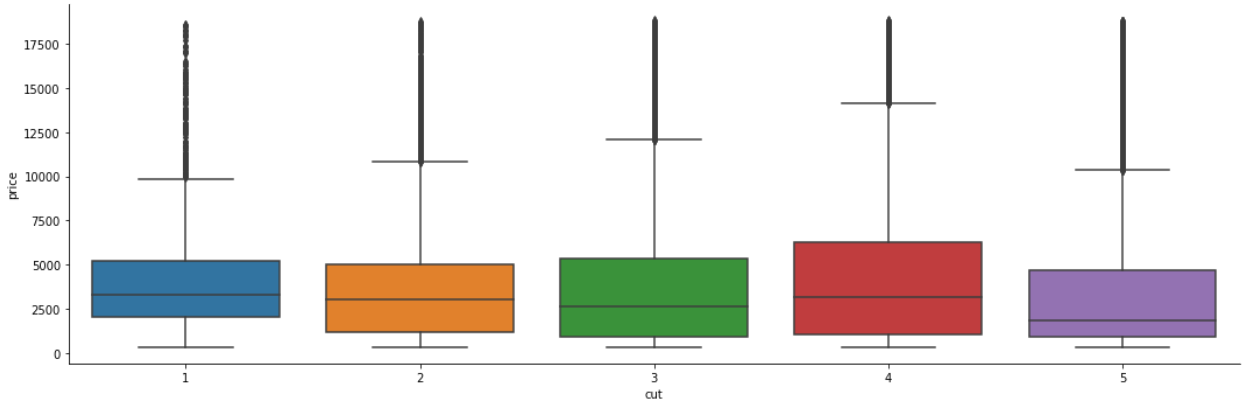
A high skewness means a distribution curve which has a shorter tail on one end and a long tail on the other. We see higher skew resulting in the price with the skew favoring the lower prices. We also see a higher skew in the carat feature as well, which makes sense considering the price skew. A high level of skewness can generate misleading results from our statistical tests. In our case, we have a higher sample of lower priced and lighter weight diamonds. Since our model will be trained on a much larger number of low priced homes, it will be less likely to successfully predict the price for expensive, higher carat diamonds.

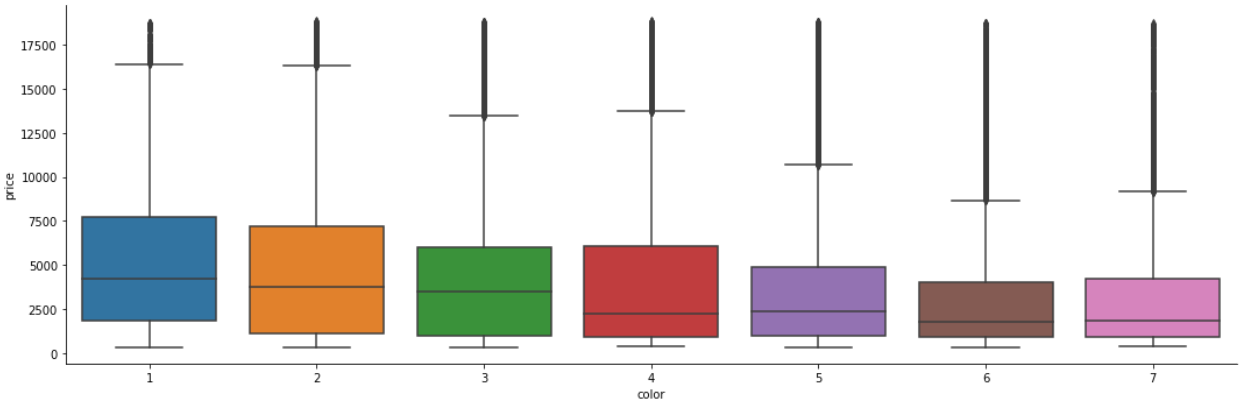
After using pandas' ".skew" method, we noticed the results. As we expect, the size, carat, and price features are quite skewed.

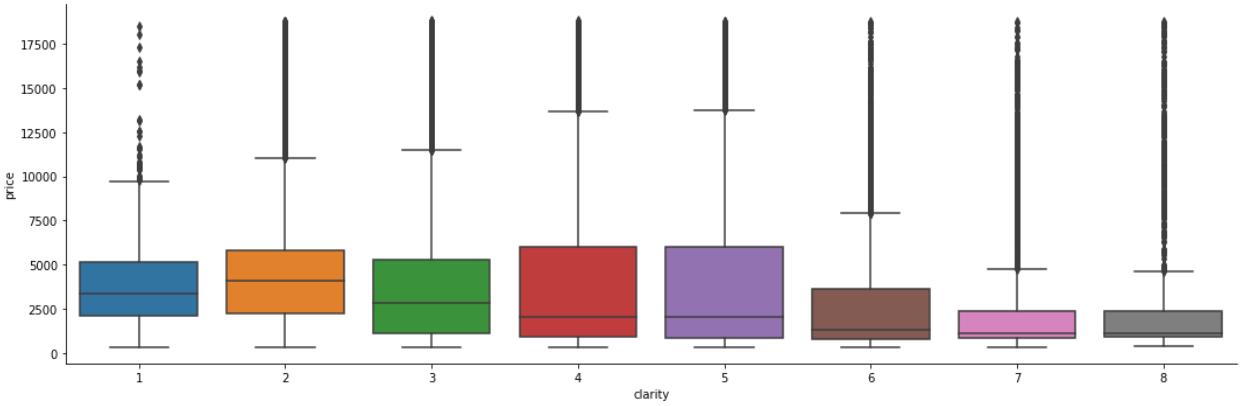
	Skew
y	2.434167
price	1.618388
z	1.522423
carat	1.116646
table	0.796896
clarity	0.551438
x	0.378676
depth	-0.082294
color	-0.189366
cut	-0.717180

We may address the skewed variables by transforming them. We can do log transformation, square root transformation, and box-cox transformation to name a few methods.

Question 1.3

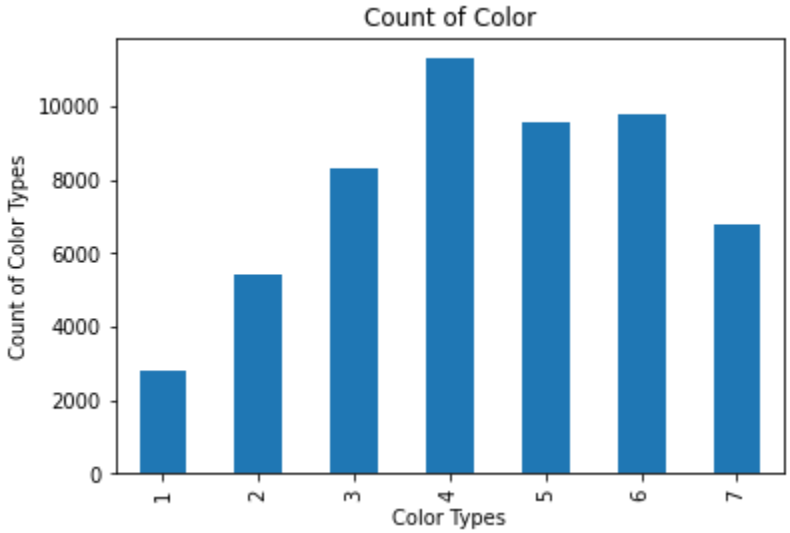




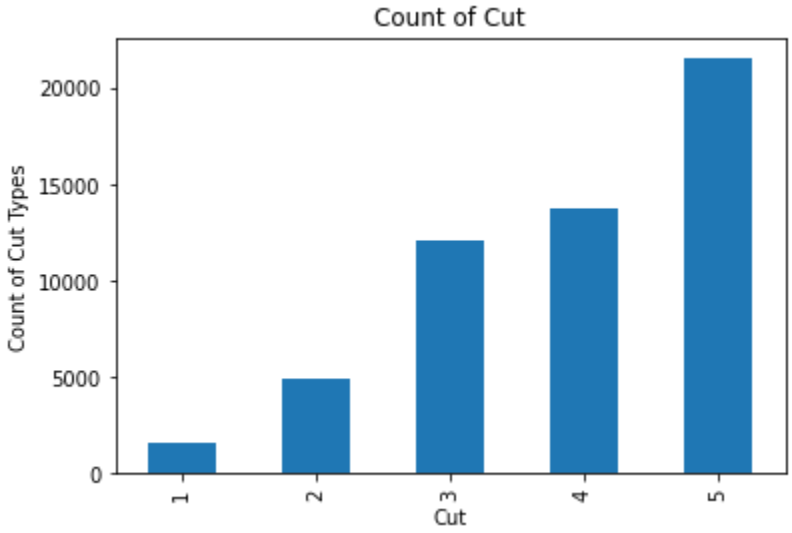


As we noticed from the skew, most of the actual diamonds are within the lower end range of price, therefore, the difference in quality for cut, color, and clarity does not change the price too much. The price remains in the range of around $2,500 - $7,500 for all qualities of these features.

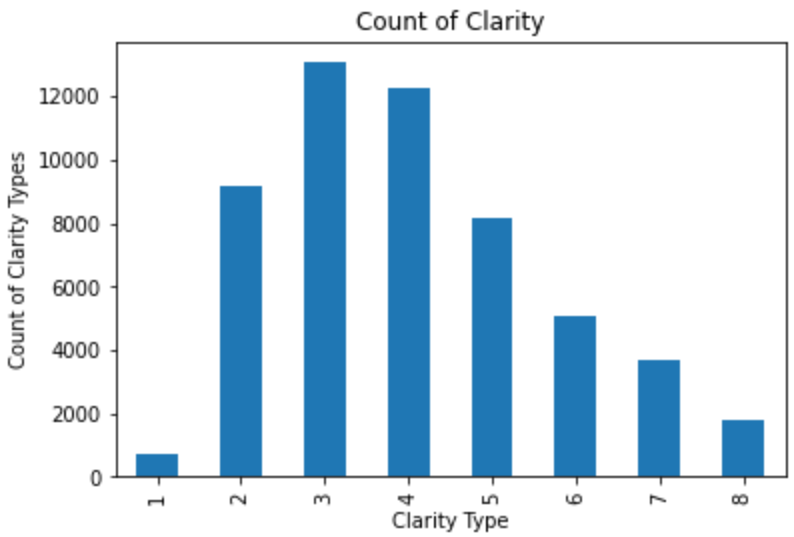
Question 1.4



The majority of diamonds have a color of 4-5 (middle range of good quality color).



We notice that in the diamond set, there are more samples of diamonds with higher quality cuts.



The majority of the diamonds have a clarity type of 3-4 (middle range of good quality).

Question 2

Question 2.1

We scaled the diamond features using StandardScaler and resulted in a set with a zero mean and unit variance. For example, the original mean for the ‘cut’ feature was 3.9 and after scaling it is basically 0. The variance for the pre-scaled ‘cut’ was 1.25 and after scaling, it became 1.0!

```
diamonds['cut'].mean()
```

```
3.904097144975899
```

```
diamondsScaled['cut'].mean()
```

```
1.4542810164611617e-16
```

```
diamonds['cut'].var()
```

```
1.2467953527310542
```

```
diamondsScaled['cut'].var()
```

```
1.0000185394612433
```

Question 2.2

If we remove features that don't have much dependency on the target price or are highly pairwise correlated, then this may help prevent overfitting to features that don't give much information above a prediction. In fact, for decision trees, they are created by choosing splits on the highest mutual information gain first.

The test RMSE may decrease because we are preventing from overfitting to too many features that don't have much impact on the outcome. In cases of high variance, reducing the number of features may reduce the extent which the model overfits.

In the case of linear regression, RMSE will likely be improved because it will perform better if irrelevant features are removed from the model. In fact, linear regression can be used as a feature selection method.

In the case of neural networks, the more data is better since the neural networks will select the best possible features out of the data on their own, so I don't think removing the features will be necessary to improve the RMSE of the neural network.

While a basic decision tree may be improved by pruning to reduce overfitting of the data, random forests don't require this feature reduction. This is because each individual tree is trained on a random subset of the features. Therefore, when the individual trees are combined into a random forest, it is not likely to overfit. With that said, if we only use a small number of random trees, then we might have overfitting, so removing the features could improve RMSE.

We notice from the mutual info score, the carat, x, y, and z have the most dependencies with price. Therefore, if we wanted to, we could drop the other features that don't provide as much info.

The features 'depth' and 'table' have the lowest mutual info scores with respect to price and after researching, we noticed that the best depth percentage is not 100%, but something around 57.5%-63%, so that the light can enter and reflect from the diamond in an appealing way. Therefore, increasing the depth past this percentage will not result in an increase in price. Additionally, the table percentage ideal is around 54-58%. Anything above that is actually poor.

Feature vs. Price	Mutual Info Score	F Score
Carat	1.65249159	3.04051487e+05
Cut	0.05680802	1.54784468e+02
Color	0.13657897	1.65440124e+03
Clarity	0.21536348	1.18800706e+03

Depth	0.03133825	6.11586346e+00
Table	0.03439326	8.86119363e+02
X	1.41316817	1.93741523e+05
Y	1.421656	1.60915662e+05
Z	1.36041602	1.54923267e+05

Question 3

The out-of-bag error is the average error for each sample of the training observation using predictions from trees in the random forest that do not have these specific training observations in their bootstrapped sample set. Essentially, first the data that is not used to construct a specific tree is found. Then, the R^2 score is calculated using that validation data. Then, the final OOB score is averaged across all the "R^2 scores" for the trees that make up the forest. This helps prevent a deep decision tree from overfitting. In a deep tree, the training error will be low and the testing error will be high. Therefore, the OOB score can be used to validate the model as the model overfits.

R-squared is: $R^2 = 1 - \frac{residual\ sum\ of\ squraes}{total\ sum\ of\ squares}$. Essentially, it tells us how well the data fit the regression model. The best score is 1 and a score below zero means that the residuals are larger than the predictions themselves. The OOB score uses the same mathematical formula as R-squared, but the random forest calculates OOB using only the training data.

Question 4

Question 4.1

Ridge regression and lasso regression are regularization techniques to prevent over-fitting, which may result from linear regression. In ridge regression, the cost function includes a penalty, which is the square of the magnitude of the learned parameters. Therefore, ridge regression tries to shrink the size of the parameters. Due to the squared term, ridge regression can reduce the learned parameters close to zero, but not zero.

In lasso regularization, the magnitude of the learned parameters are taken into account. Therefore, unlike in ridge regression, this regularization can result in some parameters being zero meaning some features

are completely taken out. Therefore, lasso regression can help reduce overfitting as well as do feature selection.

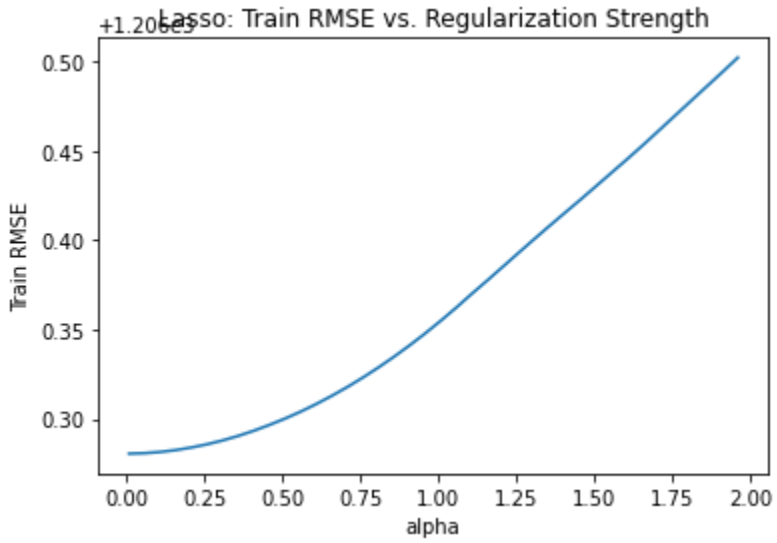
The lower the constraint, the closer the regression will resemble the ordinary linear regression model. In fact, if the constraint is zero, the cost function will be the regular linear regression model. In linear regression, there are no constraints on the learned parameters, so they can be large and may overfit.

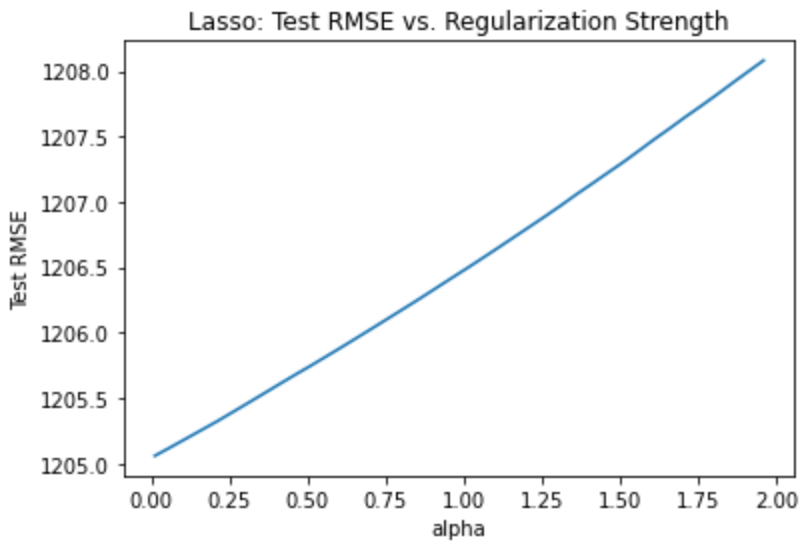
(a) Ordinary least squares

Avg Train OLS RMSE: 1206.2805967946167
Avg Test OLS RMSE: 1205.045050850989

(b) Lasso regularization

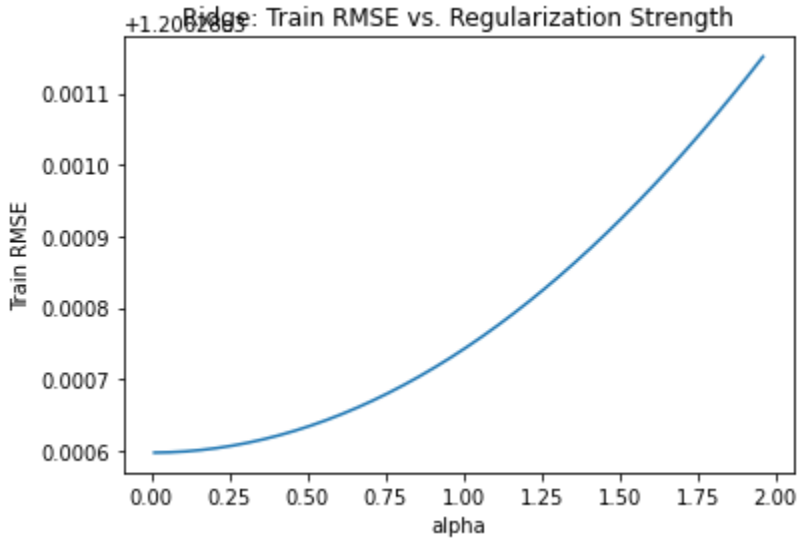
We determined the best alpha (regularization strength) by using a conservative grid search of values alpha=[0.01, 2] step size of 0.05.

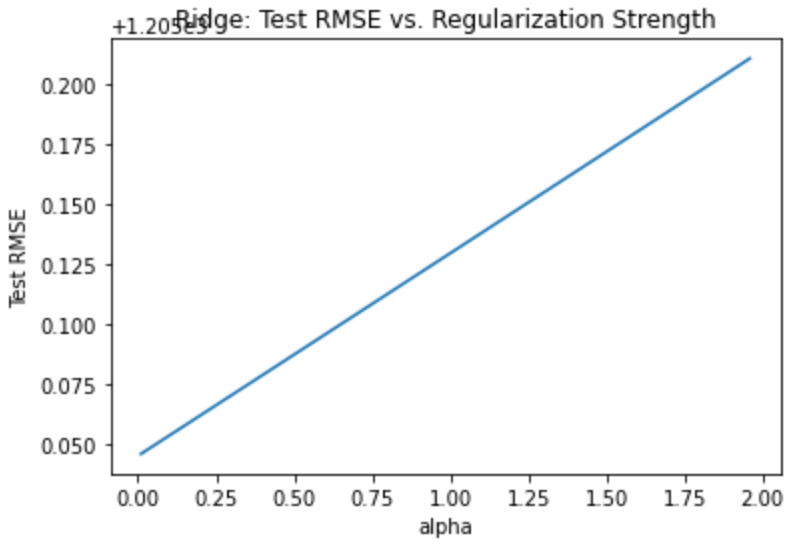




For alpha = 0.05:
Avg Train Lasso RMSE: 1206.2807904295664
Avg Test Lasso RMSE: 1205.1091902930993

(c) Ridge regression





Alpha=0.05

Avg Train Ridge RMSE: 1206.280597157293
Avg Test Ridge RMSE: 1205.0492630689291

Question 4.2

After testing many different constraints for the ridge and lasso regressions, we notice that the lowest train and test RMSE still result from the ordinary linear regression without parameter penalties. You'll see this from the previous graphs of the RMSE vs. regularization strength.

Question 4.3

As you can see from the previous questions, when "grid-searching" through different parameter constraints, the lowest test and train RMSE still resulted when the alpha was 0 reducing the cost function of the ridge regression to that of an ordinary linear regression without regularization. Regularization is used to avoid overfitting, but it looks like overfitting is not much of a problem in our case because the regularization did not improve the test RMSE much or at all.

We are predicting that a more complex model will predict even better since we don't suffer from overfitting, so the neural network and random forests that follow should perform quite well.

Between the standardized and the unstandardized features, the standardized features perform a bit better than the unstandardized.

Avg Train Ridge RMSE (standardized): 1206.280597157293
Avg Test Ridge RMSE (standardized): 1205.0492630689291

Avg Train Ridge RMSE (not standardized): 1245.350237508
Avg Test Ridge RMSE (not standardized): 1236.1340570451

Question 4.4

The P-value for the learned parameters of the linear regression model tells us whether the feature is statistically significant to the target price. The lower the p-value, the more it means our result of parameters are significant, i.e. the results were not random and the specific feature and target price have correlation. A higher p-value means that the results are that percent random and are less significant.

We can infer the most significant features by looking for the lowest p-values. We see that the x8 and x9 features are not very significant due to the higher p-value. The other features have a lower p-value, so these are significant features.

	coef	std err	t	P>|t|	[0.025	0.975]
const	2783.6213	428.811	6.491	0.000	1943.149	3624.093
x1	1.074e+04	51.837	207.261	0.000	1.06e+04	1.08e+04
x2	120.7396	5.715	21.128	0.000	109.539	131.941
x3	322.6998	3.259	99.003	0.000	316.311	329.088
x4	501.8531	3.523	142.449	0.000	494.948	508.758
x5	-79.7979	4.794	-16.645	0.000	-89.194	-70.402
x6	-26.7637	2.948	-9.079	0.000	-32.541	-20.986
x7	-877.6292	35.226	-24.914	0.000	-946.673	-808.586
x8	43.7466	20.751	2.108	0.035	3.075	84.418
x9	-29.3043	36.017	-0.814	0.416	-99.899	41.290

Question 5

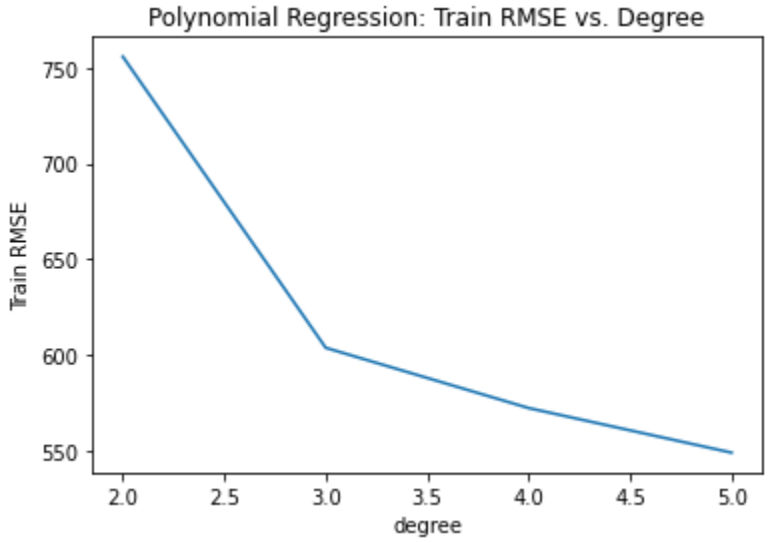
Question 5.1

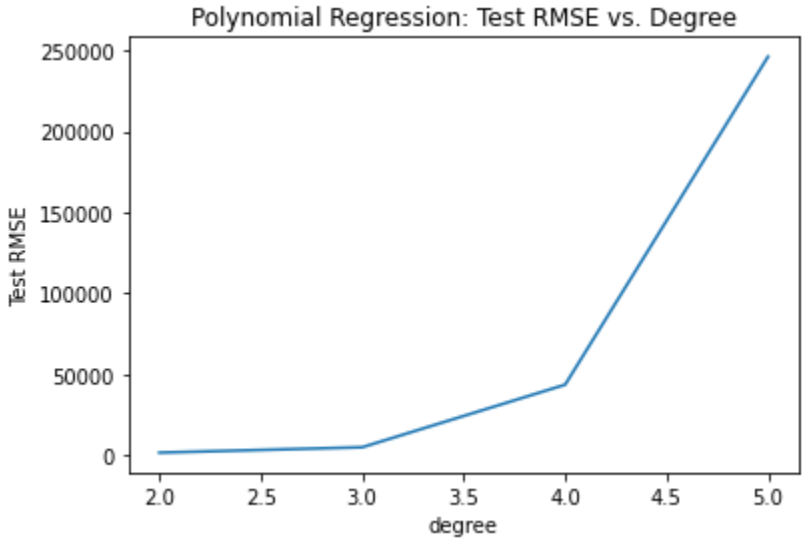
Salient features are those that distinguish one target price from another. In terms of mutual information score, these are the carat, x, y, and z.

Question 5.2

The optimal degree is two and was found by testing every degree between 2 and 5. We determined that degree 2 provides the best combination of training and testing error.

As the degree increased, the training RMSE decreased, the testing RMSE increased. This makes a lot of intuitive sense because as the model becomes more complex (i.e. the degree of the polynomial increases), the model will increasingly overfit. It may cause high testing error versus the training error because with the high degrees of polynomial, the model will be allowed to fit to noise in the training data due to the high number of inflection points. Therefore, the training error will become lower, but since the model overfits to the training error, the testing error will be higher.





---Degree: 2
Avg Train Ridge RMSE (Degree: 2): 755.6745258515463
Avg Test Ridge RMSE (Degree: 2): 1767.506593568652
---Degree: 3
Avg Train Ridge RMSE (Degree: 3): 603.9229569866088
Avg Test Ridge RMSE (Degree: 3): 5098.175122481951
---Degree: 4
Avg Train Ridge RMSE (Degree: 4): 572.5232402494901
Avg Test Ridge RMSE (Degree: 4): 43699.08000837405
---Degree: 5
Avg Train Ridge RMSE (Degree: 5): 549.2852625129897
Avg Test Ridge RMSE (Degree: 5): 246182.02574823238

Question 6

Question 6.1

We introduced a conservative grid with the following parameters:

- Hidden_layer_sizes: (1,1), (1,10), (5, 2), (5, 5), (5, 10), (10, 2), (10, 5), (10, 10), (50, 1), (50, 2) (50, 5), (50, 10)
- Alpha: 0.00005, 0.1, 0.5
- Activation function: logistic, relu

Best model: activation = ‘relu’, alpha = 0.1, hidden_layer_size = (50, 5)
Avg Train MLP NN RMSE: 699.2527456711227
Avg Test MLP NN RMSE: 703.512713848753

Example of top scores in GridSearch:

mean_test_score	mean_train_score	params
701.3139487516557	741.4119396171134	{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (50, 5)}
726.0543270441373	728.6958017642879	{'activation': 'relu', 'alpha': 0.5, 'hidden_layer_sizes': (50, 5)}
751.7172138477366	724.5640205395682	{'activation': 'relu', 'alpha': 0.5, 'hidden_layer_sizes': (50, 10)}
779.3170152016356	809.6554896161203	{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (10, 5)}
802.7930947281063	830.0461700796095	{'activation': 'relu', 'alpha': 0.5, 'hidden_layer_sizes': (5, 10)}

Question 6.2

The neural network performed much better than the linear regression did, which was expected. The linear regression performs well on linear dependencies, but since our data is clearly nonlinear, the neural network performed better. The non-linear activation function allows for the non-linear relationship. However, we should note that a linear regression model has less parameters to estimate than a neural network. Therefore, a neural network requires a larger dataset in order to generalize and prevent overfitting, so it's good to note that when using the neural network. However, we think our dataset for the diamonds is sufficiently large to take advantage of the neural network. It's also good to note that it is easier to explain the linear regression model than it is to interpret the neural network.

Question 6.3

After cross validating the logistic and ReLU activation functions, I chose 'relu' because it consistently provided better train and test RMSE scores.

Question 6.4

Increasing the depth of the neural network may cause overfitting as a more complex model might. This occurs when the neural network has more processing capacity than the limited information contained in the training set provides. Additionally, a very deep neural network will increase computation time. Moreover, there can be an issue called vanishing gradients. As gradients are backpropogated through a neural network that is very deep, they can become small and approach zero, therefore, slowing down or presenting convergence during training. If the depth is very large, then strong regularization is needed.

Question 7

Question 7.1

The max number of features hyperparameter is the number of random subsets of features to consider when splitting a node. During the construction of the simple decision trees, it will consider the number of max features for node splitting. If we increase max_features, the performance may improve because at each node there are more options to consider. This could be considered a regularization parameter because if you consider all features when splitting, it could result in overfitting.

The number of trees hyperparameter is how many trees are in the forest and the number of simple trees that will be voting. The more trees there are, the better our predictive performance will be. This is because we will have more simple models of decision trees with predictions combined to create the final prediction. Changing the amount of trees will result in regularization because the more trees in the forest will help prevent high variance since the outcome is averaged over all trees.

The depth of the tree is how many layers/splits occurred in the tree. The deeper the tree, the accuracy may increase as performance increase, however, it is more likely to overfit the data. However, the shallower the tree, the more likely it is to underfit. With this said, random forests can counter the increase in overfitting from increased depth by averaging over multiple trees. There is a good balance of depth, which we demonstrate after grid searching. The depth of the tree can be considered a regularization parameter because the deeper the tree, the more likely to overfit and the shallower the tree, the more likely to underfit.

In order to tune these hyperparmarter, we should optimize the out-of-bag error to achieve a good balance between bias and variance.

GridSearch Parameters:

- Max number of features: 2, 5, 10
- Number of trees: 5, 25, 50
- Depth of each tree: 4, 5, 10

Example of a few rows from GridSearch:

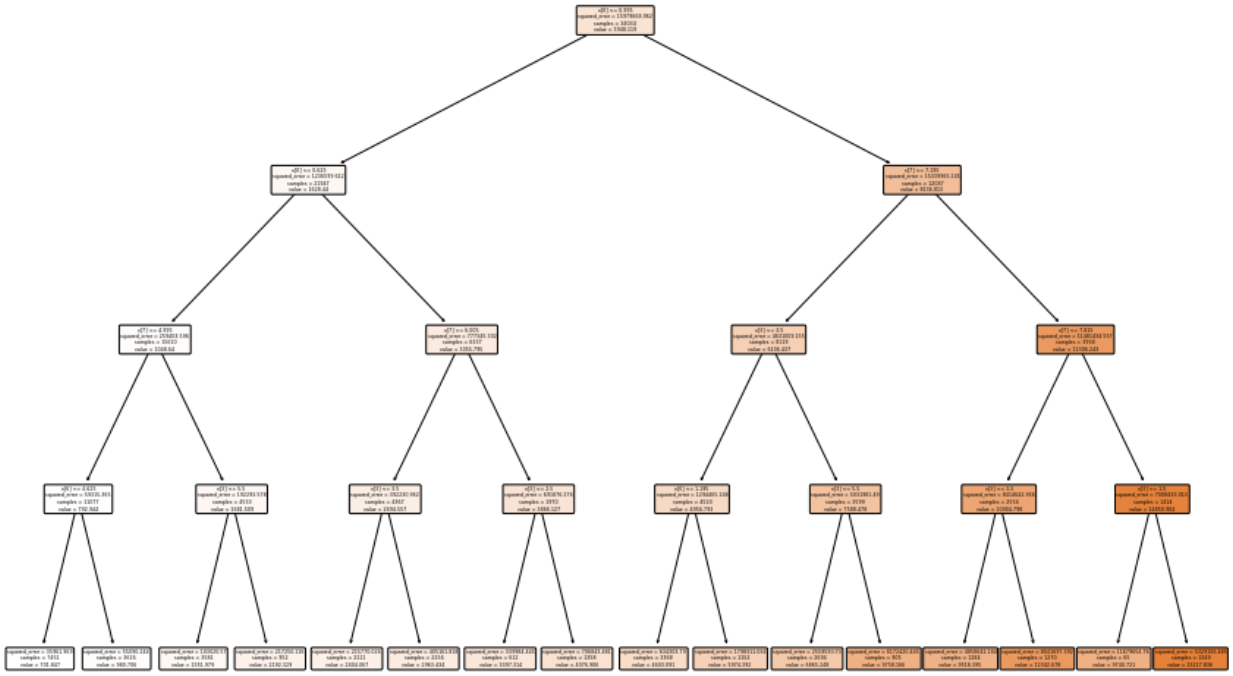
mean_test_score	mean_train_score	mean_oob_score	params
800.3511552202342	478.60981322437885	0.9802231423571544	N_estimators: 50, depth: 10, Max_Features: 10
782.0011715695767	492.6966558473573	0.9798156397564176	N_estimators: 50, depth: 10, Max_Features: 5
792.6949593091811	495.8790293584408	0.9789823196821926	N_estimators: 25, depth: 10, Max_Features: 5
808.2484861409224	480.95201319060834	0.9798020281337052	N_estimators: 25, depth: 10, Max_Features: 10
830.31185358524	518.6346501977685	0.7724561402900709	N_estimators: 5, depth: 10, Max_Features: 5
830.7443941182557	496.3628030373522	0.7772128745251333	N_estimators: 5, depth: 10, Max_Features: 10
1388.4751858070385	1223.3953682134595	0.6961052959090706	N_estimators: 5, depth: 4, Max_Features: 2

Question 7.2

Decision trees are highly non-linear, but they are seen as piecewise linear models. Decision trees divide the data into regions based on questions thresholded on features. From these questions, the decision tree can divide up the feature space into vertical and horizontal lines to create a complex non-linear decision boundary. The random forest is essentially just having many of these decision trees trained on bootstrapped samples of the data. Then, all these simple models of decision trees will take a vote on the final prediction. Therefore, the random forest is also non-linear.

Question 7.3

- Number of trees: 50
- Max_depth: 4
- Max_featuers: 10



The feature selected for branching at the root node is the carat. This makes a lot of sense considering this is the important feature we determined earlier. Since this is the root node for branching, we can tell that it provides a lot of information about splitting the population compared to other features.

Question 7.4

The out of bag score for our best random forest is 0.980. This tells us that 98% of the predicted rows from the out-of-bag sample are correct. This shows that our random forest performed quite well.

The out-of-bag error is the average error for each sample of the training observation using predictions from trees in the random forest that do not have these specific training observations in their bootstrapped sample set. Essentially, first the data that is not used to construct a specific tree is found. Then, the R^2 score is calculated using that validation data. Then, the final OOB score is averaged across all the "R^2 scores" for the trees that make up the forest. This helps prevent a deep decision tree from overfitting. In a deep tree, the training error will be low and the testing error will be high. Therefore, the OOB score can be used to validate the model as the model overfits.

The R2 is 1 minus the ratio of the residual sum of squares and the total sum of squares of the prediction. An R2 of 1 indicates that the model explains all of the variability perfectly. A score below zero means that the residuals are larger than the predictions themselves.

mean_test_score	mean_train_score	mean_oob_score	params
800.3511552202342	478.60981322437885	0.9802231423571544	N_estimators: 50, depth: 10, Max_Features: 10
782.0011715695767	492.6966558473573	0.9798156397564176	N_estimators: 50, depth: 10, Max_Features: 5
792.6949593091811	495.8790293584408	0.9789823196821926	N_estimators: 25, depth: 10, Max_Features: 5
808.2484861409224	480.95201319060834	0.9798020281337052	N_estimators: 25, depth: 10, Max_Features: 10
830.31185358524	518.6346501977685	0.7724561402900709	N_estimators: 5, depth: 10, Max_Features: 5
830.7443941182557	496.3628030373522	0.7772128745251333	N_estimators: 5, depth: 10, Max_Features: 10
1388.4751858070385	1223.3953682134595	0.6961052959090706	N_estimators: 5, depth: 4, Max_Features: 2

Question 8

Question 8.1

The important parameters include: tree depth, learning rate, and L2 regularization. Learning rate affects the overall time of training. Additionally, it may help with regularization. For example, if the training does not converge, we should increase the learning rate. However, if we increase the learning rate too much, then overfitting may occur.

Search space:

- Tree depth: [4, 6, 10]
- Learning rate: [0.03, 0.1]
- L2 regularization: [5, 10]

Question 8.2

val. score: 0.9823007909612922
test score: 0.9825277008586653
best params: OrderedDict([('depth', 6), ('l2_leaf_reg', 7), ('learning_rate', 0.08348635236883853)])

Avg Train CatBoost RMSE: 437.0101442593449
Avg Test CatBoost RMSE: 555.1115449018358

Question 8.3

The tree depth will help with performance because the deeper the tree, the accuracy may increase as performance increase, however, it is more likely to overfit the data. However, the shallower the tree, the more likely it is to underfit. Additionally, the deeper the tree, the longer the overall training time.

The learning rate helps with performance because if the training does not converge, then we should increase the learning rate. The learning rate can also help with regularization. For example, if overfitting is detected, then we should decrease the learning rate. The learning rate will definitely affect the fitting efficiency (speed of efficiency). The larger the learning rate, the larger the gradient step. The smaller the learning rate, the more iterations are required for training and will result in longer overall training time.

The L2 regularization parameter is similar to how you would think in a linear regression setting. L2 regularization helps us minimize loss and complexity by including the regularization term that measures model complexity. L2 regularization will force the weights to decay to zero (but never actually getting to zero), so it is useful to compress the model and prevent overfitting.

Question 9

Question 9.1

#nfl
Tweets Per hour

1. 694.6666666666666,
2. 730.5833333333334,
3. 700.7083333333334,
4. 724.2083333333334,
5. 314.8333333333333,
6. 274.4166666666667,
7. 237.54166666666666,
8. 179.29166666666666,
9. 158.36,
10. 149.58333333333334,
11. 145.16,
12. 164.375,
13. 182.28,
14. 188.04,
15. 286.5416666666667,
16. 360.24,
17. 392.24,
18. 425.9583333333333,
19. 393.16,
20. 499.5,
21. 557.875,
22. 470.375,
23. 494.5,
24. 909.0

Average Retweets: 1.5344
Average Number of Followers: 4662.37

#superbowl
Tweets Per hour

1. 8377.666666666666,
2. 11003.875,
3. 5824.625,
4. 5561.375,
5. 563.875,
6. 350.7083333333333,
7. 339.0,
8. 236.5,

9. 202.56,
10. 193.72,
11. 211.76,
12. 243.44,
13. 317.96,
14. 377.4,
15. 532.12,
16. 725.76,
17. 703.64,
18. 850.36,
19. 870.12,
20. 1108.125,
21. 1155.7916666666667,
22. 1112.75,
23. 1539.1666666666667,
24. 7955.375

Average Retweets: 2.391
Average Number of Followers: 8814.96

#sb49
Tweets Per hour

1. 3822.5652173913045,
2. 3539.304347826087,
3. 2216.086956521739,
4. 666.7826086956521,
5. 407.1818181818182,
6. 193.3913043478261,
7. 234.36363636363637,
8. 132.47619047619048,
9. 107.35,
10. 74.9090909090909,
11. 82.89473684210526,
12. 106.75,
13. 123.45454545454545,
14. 165.22727272727272,
15. 216.69565217391303,
16. 295.7826086956522,
17. 267.75,
18. 324.32,
19. 1086.391304347826,
20. 4574.434782608696,
21. 4337.619047619048,
22. 3080.478260869565,

23. 2791.086956521739,
24. 3756.8333333333335

Average Retweets: 2.527
Average Number of Followers: 10374.16

#patriots
Tweets Per hour

1. 1591.625,
2. 861.0416666666666,
3. 1397.0,
4. 1881.4583333333333
5. 419.7916666666667,
6. 237.54166666666666,
7. 191.91666666666666,
8. 125.91666666666667,
9. 76.6,
10. 64.44,
11. 71.56,
12. 97.12,
13. 143.24,
14. 163.0,
15. 243.52,
16. 302.28,
17. 313.88,
18. 344.88,
19. 668.12,
20. 2255.4166666666665,
21. 2075.375,
22. 1511.1666666666667,
23. 1380.3333333333333,
24. 1838.2916666666667

Average Retweets: 2.527
Average Number of Followers: 10374.16

#gopatriots
Tweets per Hour

1. 209.25,
2. 105.78947368421052,
3. 183.25,
4. 183.94736842105263,
5. 19.36842105263158,
6. 8.277777777777779,

7. 8.058823529411764,
8. 7.8,
9. 5.3076923076923075,
10. 4.909090909090909,
11. 4.1875,
12. 5.461538461538462,
13. 6.166666666666667,
14. 6.25,
15. 9.7,
16. 9.9,
17. 12.25,
18. 13.333333333333334,
19. 22.40909090909091,
20. 28.047619047619047,
21. 37.76190476190476,
22. 43.78947368421053,
23. 67.4,
24. 206.10526315789474

Average Retweets: 1.408
Average Number of Followers: 1427.25

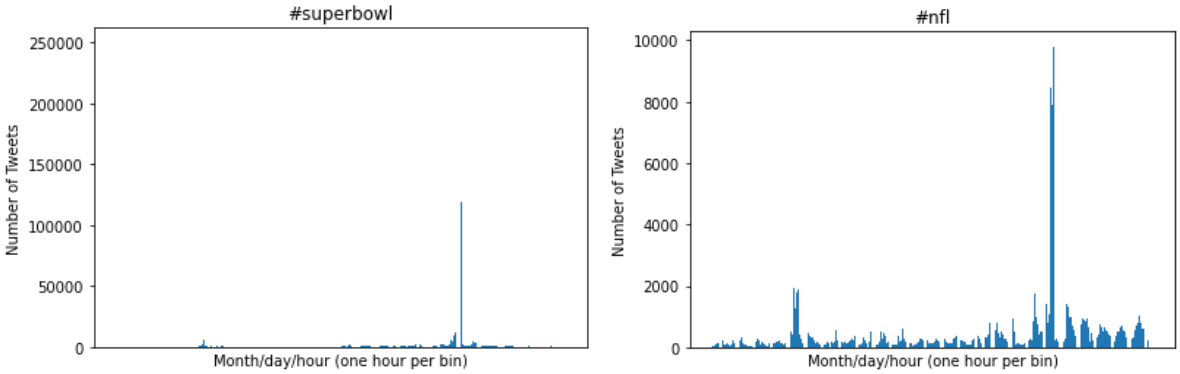
#gohawks
Tweets per Hour

1. 592.7916666666666,
2. 587.5833333333334,
3. 503.5833333333333,
4. 323.4583333333333,
5. 198.58333333333334,
6. 158.8695652173913,
7. 135.41666666666666,
8. 83.20833333333333,
9. 57.47826086956522,
10. 36.59090909090909,
11. 25.125,
12. 20.791666666666668,
13. 29.59090909090909,
14. 37.958333333333336,
15. 69.25,
16. 137.16666666666666,
17. 191.0,
18. 239.75,
19. 321.1666666666667,
20. 436.4583333333333,

21. 571.9166666666666,
22. 433.25,
23. 511.5833333333333,
24. 1358.7083333333333

Average Retweets: 2.013
Average Number of Followers: 2217.92

Question 9.2



Question 10

Describe your task:

We decided to explore two parallel tasks for this section of the project. For the first task, we performed text classification on the tweets in order to predict the hashtag that is attached to a specific tweet. For the second part, we looked at generative modeling of tweet data. In this part, we trained our models on data from only a single hashtag and then asked them to generate new tweets in the style of the training data.

Explore any data/metadata:

For our models, we only used the text of the tweets themselves. To extract this we streamed the data line by line from each file and then parsed the data into a dictionary using the json library. We were then able to easily extract the raw tweet text from each post. We mapped each hashtag to a numerical label so we were able to calculate loss using standard loss functions.

Model Design:

For both parts of this model we use neural networks as our models. For the classification task, we use a simple MLP. Our MLP contained six hidden layers with a hidden dimension of 128. The input dimension was equal to the dimension of our features extracted from our input data (see below for details on this process). The output dimension was equal to the number of classes, which in our case was six. We trained this model for only two epochs using cross entropy loss and the ADAM optimizer and a learning rate of 1e-3.

For the generative model we leveraged the power of recurrent neural networks, specifically LSTM cells. Recurrent neural networks have been shown to be effective when the input data is sequential, for example musical or textual data. However, vanilla RNNs suffer from the vanishing/exploding gradients problem more than linear layers do. LSTMs were introduced to solve this issue, and they have since become a staple in deep learning and NLP literature. Our specific model used an embedding layer, followed by two LSTM cells and finally a linear layer which mapped the output of the LSTM to a set of logits over the number of words in our vocabulary. This model was also trained with cross entropy loss and the ADAM optimizer with a learning rate of 1e-3, but we trained this model for fifty epochs.

In a similar fashion, we used Gated Recurrent Units (GRUs), to see how they performed compared to LSTM cells. GRUs are also a set of RNN, it is essentially a less complex version of LSTMs. The key difference between GRU and LSTM is that GRU's bad has two gates that are reset and update while LSTM has three gates that are input, output and forget. Typically, when a dataset is small, it is nominal to use GRUs due to its simplicity. However, working with tweets data, it proved that LSTMs were performing much better as shown through our validation scores. Which is why there are 2 models utilizing GRU structures. The first is just a GRU and the other model has two GRUs cascaded. The reason for the cascaded model was to see if it can perform better than just one GRU, which it does not, unfortunately.

Lastly, we have our Markov Model for generating text. This model was simply used to generate text, it was not put through training and validation testing. The reason for this model is to show other methods of attempting to generate tweets outside of neural network models we have. How the model works is a user feeds a start text and from there the markov model takes action. It generates tweets by a mathematical system that experiences transitions from one state to another according to certain probabilistic rules. Essentially, the probability of transitioning to any particular state is dependent on the current state and time elapsed. This process is similar to stochastic processes, but in the Markov chain it is 'memory-less'. That is, future actions are not dependent upon the steps that led up to the present state. Looking at our

text, is it not the best, it generates some coherent phrases, but overall they do not make sense, nonetheless, exciting to see some generated tweets.

Feature Engineering Process:

Before extracting any features from our text data we perform some preprocessing steps. We use regular expression matching to remove any words starting with http, as this allows us to remove any links from the tweets we are using as data. We don't want links as we want to process only natural language, and links are not a part of this vocabulary. We also remove any words that begin with a hashtag, since we don't want to give the model direct information as to which class it belongs to. We also remove hashtags that aren't directly connected to the class label for the same reasons we remove links from our input data. Finally, we remove all punctuation from the tweet since we want our model to operate on a sequence of words. Punctuation doesn't add any relevant information to the tweet.

To extract features for classification, we used GLoVE embeddings, similar to project 1. We chose to do this over TFIDF since we found that in project 1 they performed better, so we inferred that this trend would hold for our use case as well. In order to get the GLoVE embedding for a tweet, we iterated over all the words in the tweet and extracted the GLoVE embedding for each one. We then took the average over all these vectors and used this result as the input to our models. Due to the large size of the dataset and the memory constraints of our Colab notebook, we used GLoVE embeddings of size fifty for this project.

For our generative model, we extracted features in a different manner. We first created a vocabulary over all the words in the dataset. We did this by simply mapping each word to a numerical index that corresponded specifically to that word. After creating this vocabulary, we used it to create one-hot encoded vectors for all words in each tweet. This creates a sequence of one-hot encoded vectors for each tweet. Finally, we created a series of data points for each tweet by taking the first n-1 vectors as the input features and the final vector in the sequence as the label. This feature extraction method allowed us to pose the generation problem as a classification problem with a number of classes equal to the number of words in our vocabulary. It is worth noting that the vocab we created had 1554 words in it, which is a large number of classes and thus a difficult problem to solve. Furthermore, we needed to pad each sequence to a fixed length using the zero vector in order for each data point to be compatible with our network.

Results:

We report the best accuracies for both our models below. To evaluate our models, we created a random train/test split using sklearn. For classification, accuracy is a standard metric to report, and we felt it was more illustrative than reporting the model loss itself. For the generative task, we are able to report accuracy since we are essentially posing the problem as a classification problem with the number of classes equal to the number of words in the vocabulary. Since the label for each data point is the next word in the tweet (eg. the class label) we are able to calculate the cross entropy loss and accuracy of the models predictions.

MLP Classification Accuracy: 0.93
LSTM Generation Accuracy: 0.33
GRU1 Generation Accuracy: 0.28
GRU2 Generation Accuracy: 0.24

Basesline Comparisons:

We compare our model to three different baselines. The first baseline uses a simple MLP instead of LSTM cells to generate text. The second baseline uses a GRU and the final baseline uses a Markov Model.

For the generative MLP, we give it the average GLoVE embedding of the sequence of words in a tweet excluding the last one, and we ask the model to predict the class label of the next word. This class label is just the index of the word in the vocabulary we created for our original generative model. We trained this model in the same way we trained the other networks for this section. The performance of this model was comparable to our model, with a final validation accuracy of 0.32.