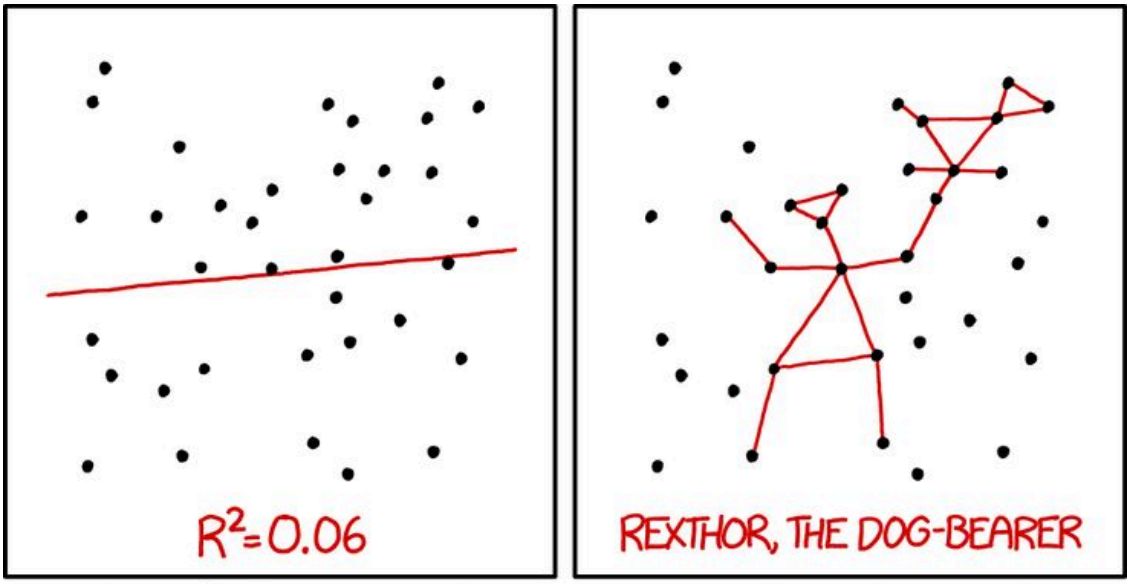
# Project 4: Regression Analysis and Task Design

Authors: Alex Chen, Jiamin Xu

Electrical and Computer Engineering 219: Winter 2024

Professor Vwani Roychowdhury

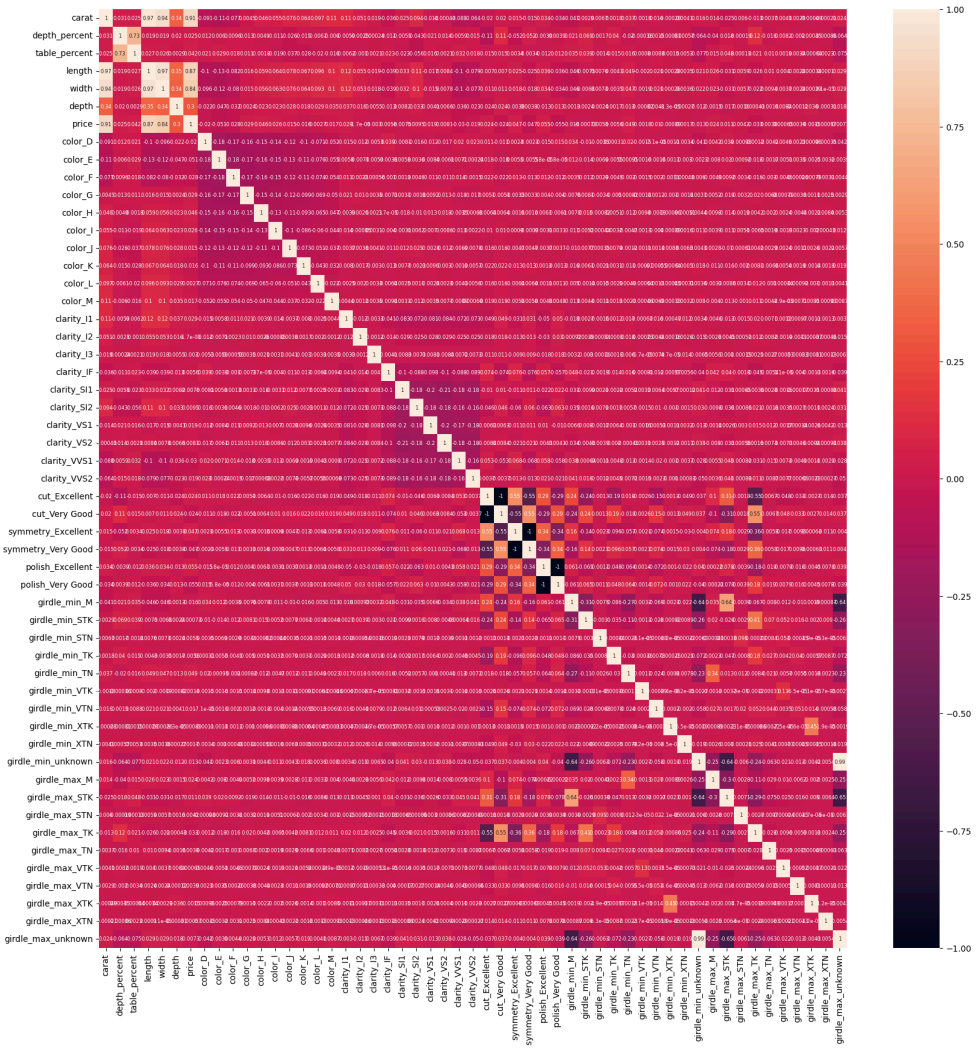


# Question 1

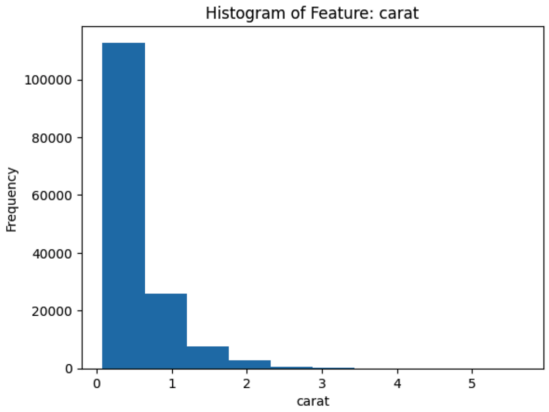
## 1.1

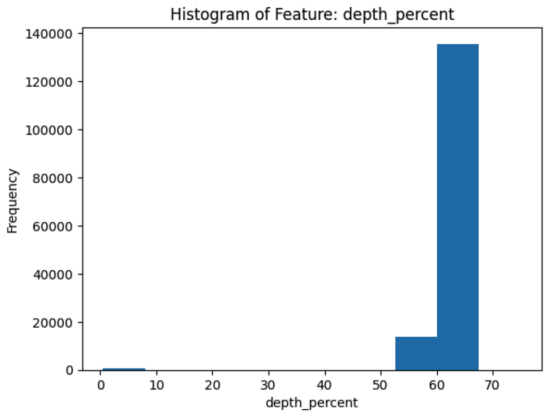
Based on the created correlation matrix, it seems that carat, length, and width are the features that are most highly correlated with price (0.91, 0.87, 0.84, respectively). Depth is also correlated (0.3) – more than other features such as color or clarity, but definitely less significantly than the aforementioned features. Furthermore, there are no features that have significantly high negative correlation.

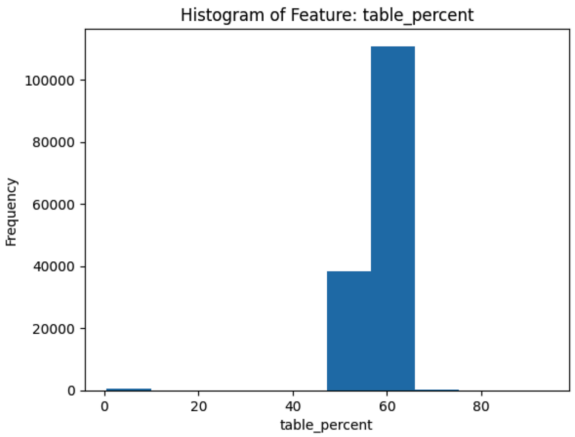
These correlation patterns suggest that as the carat, length, or width of a diamond increase, the price also increases. As carat increases, price increases most. If length of the diamond increases, the price increases, but not as much. Then, as width of the diamond increases, the price increases by the least of the three, but still by much more than increasing any other feature would. On a high level, these pattern suggest that the quality and size of the diamond matter most in determining price.

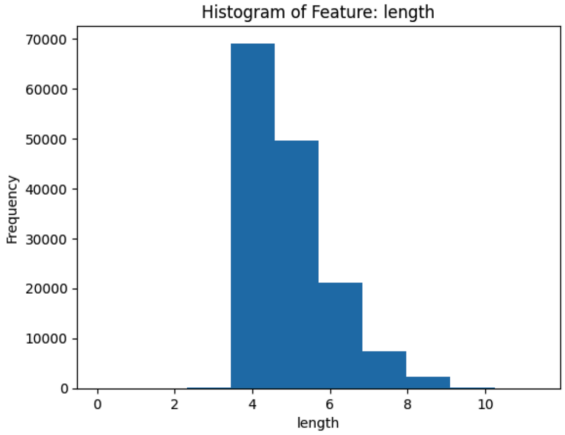


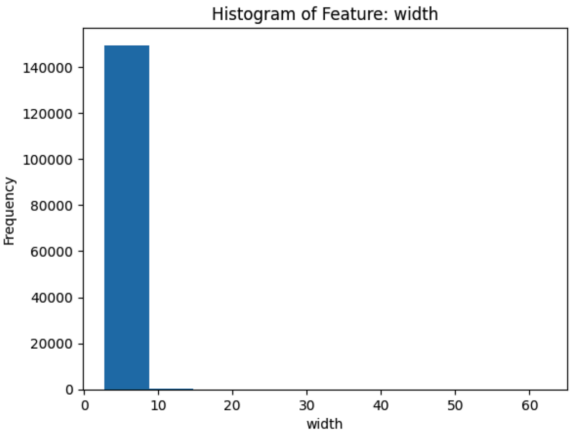
1.2

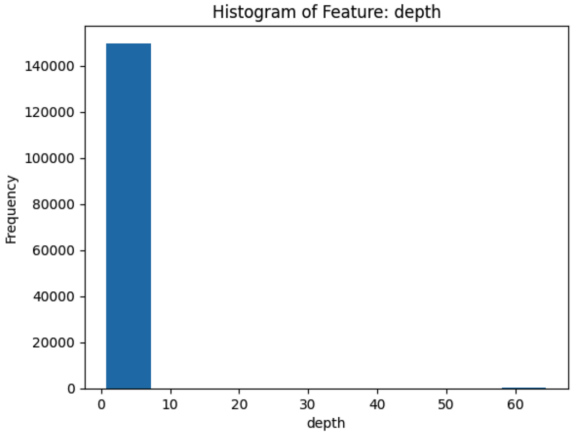

Histogram of Feature: carat
Frequency
carat

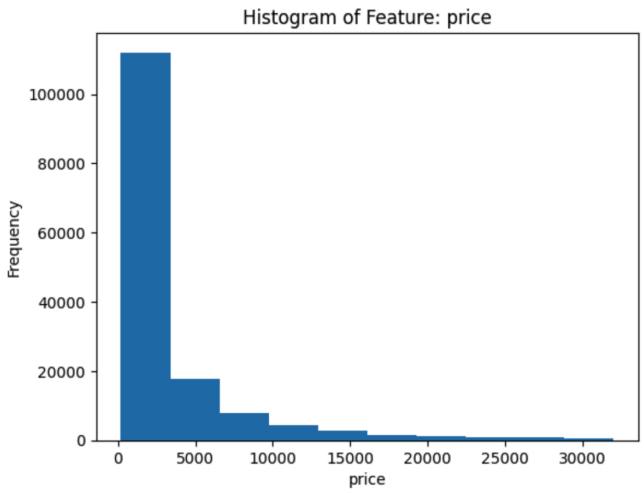

Histogram of Feature: depth_percent
Frequency
depth_percent


Histogram of Feature: table_percent
Frequency
table_percent


Histogram of Feature: length
Frequency
length

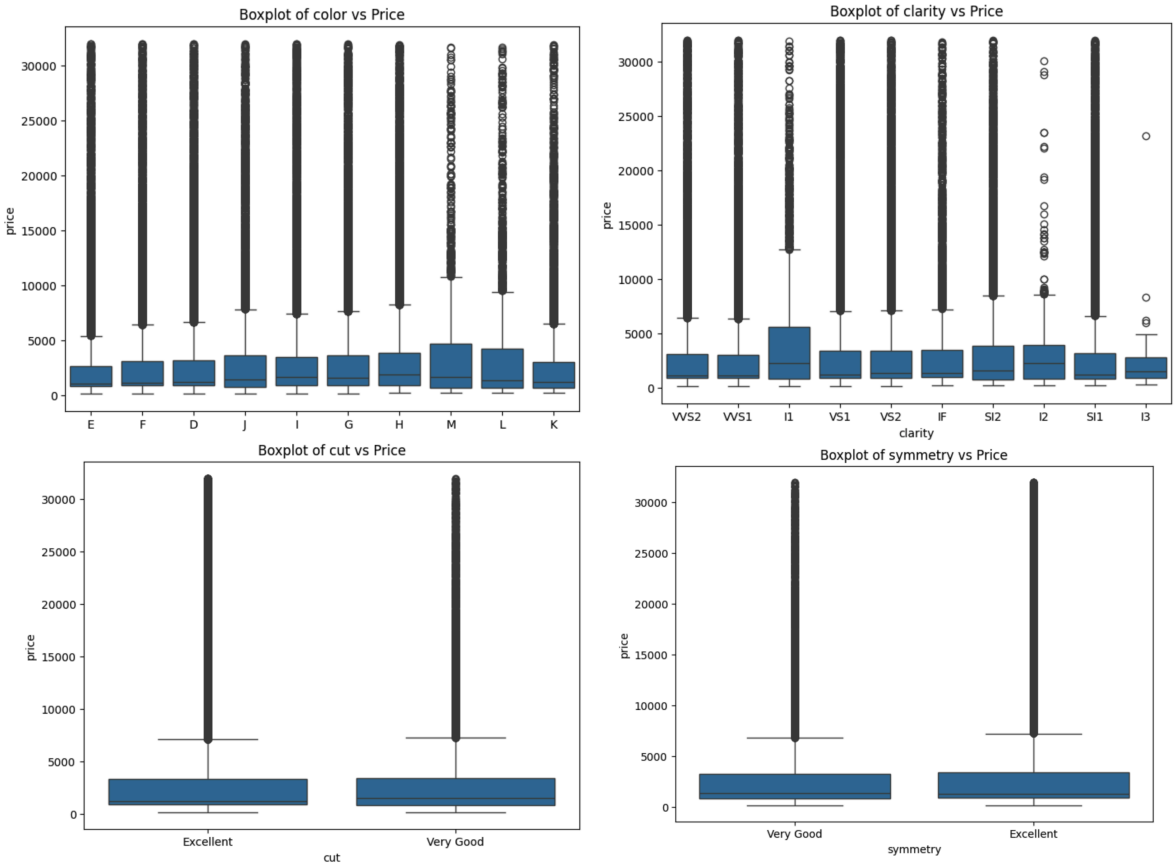

Histogram of Feature: width
Frequency
width

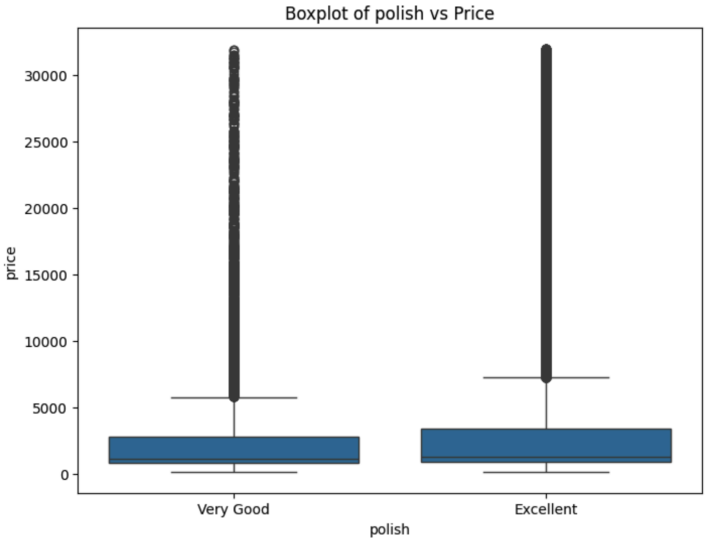

Histogram of Feature: depth
Frequency
depth

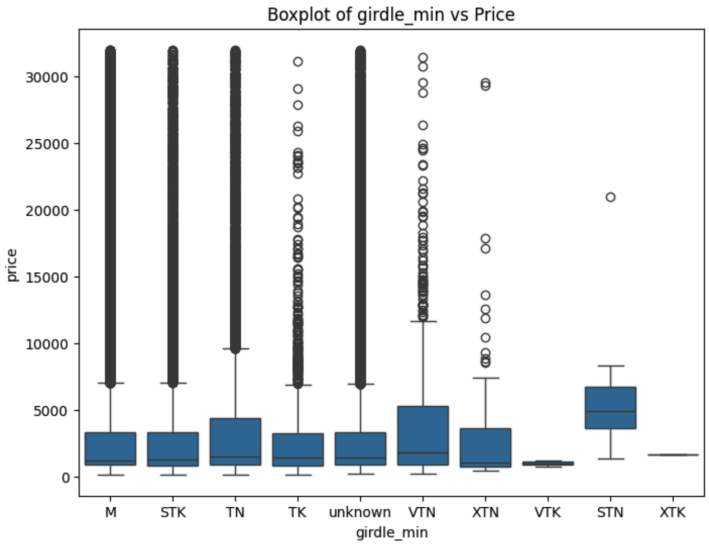


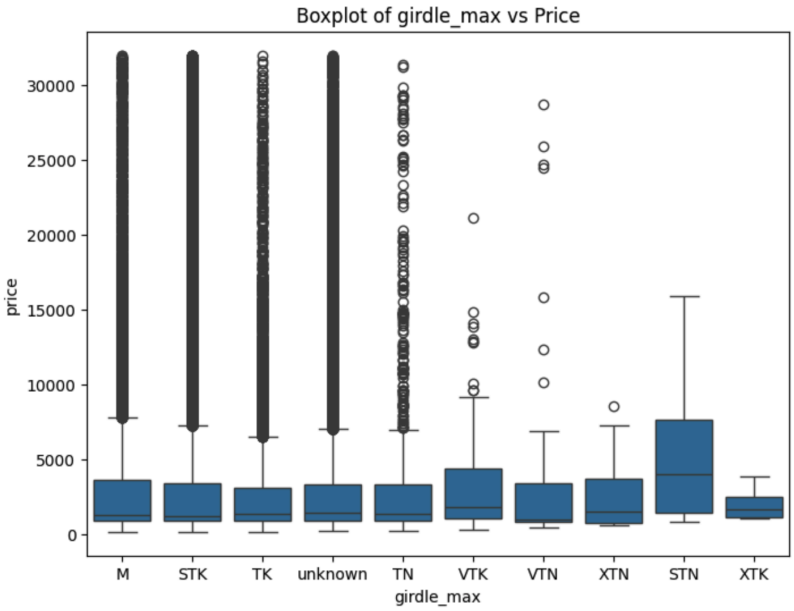

The histograms for each numeric feature, as well as the price target, are shown above. One method to handle high skewness in a feature's distribution are power transformations. For example, a quartic transformation can be used which would transform each value to its 4th power. This would work for left-skewed data because it would pull in the left-tail, while dragging out the right-tail, since larger values will be affected more than smaller values. On the other hand, right-skewed data can have log, or square root, or similar transformations applied to them. The effect of this is that larger values would be decreased by a larger amount than small values, and thus, the right tail would be pulled in more.

## 1.3



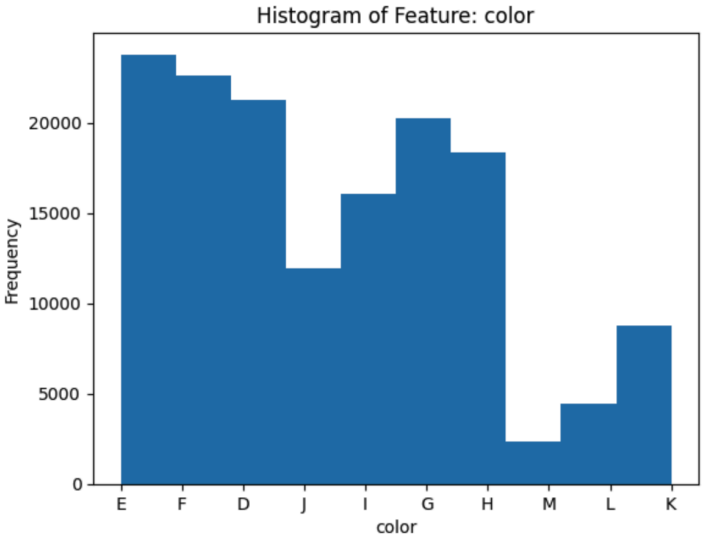


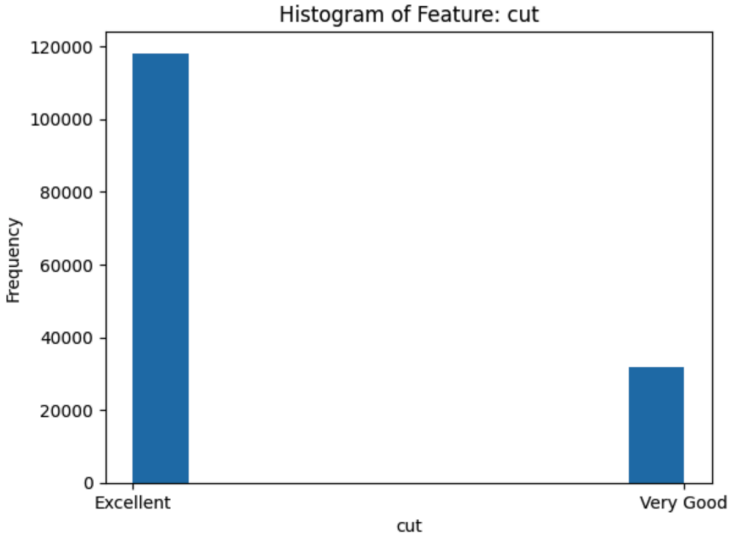



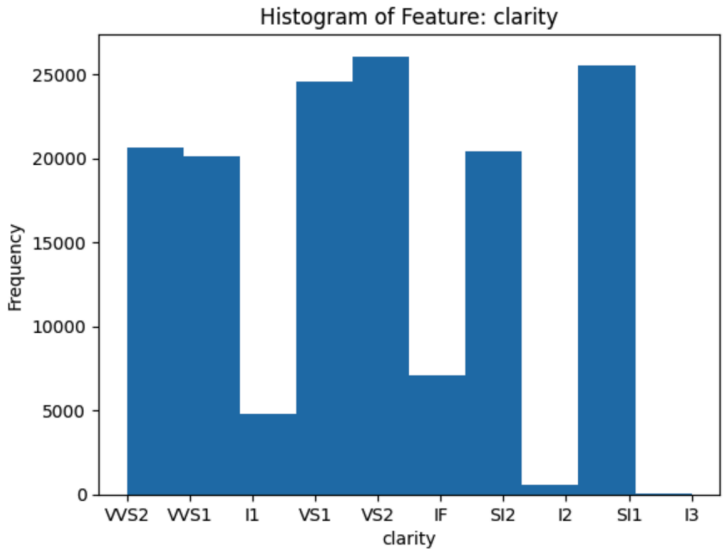




Each categorical feature has its own box plot showcasing each value of the feature versus price. More specifically, for a feature, how do the price points of diamonds with a certain value of that feature vary? One observation is that each value for each category has a large number of outliers. For example, if we look at girdle_min, no matter what value girdle_min takes on, there are a huge number of outliers. This indicates that each of these categorical features do not accurately and/or individually indicate price. For example, only setting polish to excellent is not enough to limit the range of prices of diamonds. However, there are a few features and values for those features that do represent the prices well. The main ones are STN and XTK for girdle_max, and VTK and XTK for girdle_min. For these attributes, they indicate that no matter the other attributes of a diamond, if a diamond has that attribute, for example, an extra thick max girdle size, the diamond will be worth less (no large outliers).

### 1.4




## Question 2

### 2.1

Data has been scaled using scikit-learn's StandardScaler class.

### 2.2

Feature selection should improve the results of my trained models in terms of test RMSE. By selecting the features that have higher mutual information (MI) scores, we select features that are more correlated with the target. Thus, by eliminating other features, our model can be as accurate as possible and perform best on test data.

This is not true for all models. As an example, with a neural network model, we do not want to perform feature selection, since the model has the ability to perform it's own feature selection and determine which features are important and their weights. Feature selection is more important in something like a regression model. Furthermore, with something like Lasso regression, the regression model itself shrinks many coefficients to zero due to its regularization technique, thus performing feature selection itself.

The two features with the lowest MI w.r.t price target are girdle_min_VTN, and girdle_max_XTK. Note that this is after one hot encoding has been performed.

The most significant 4 features are carat, length, width, and depth, which is as expected based on correlation matrix computed in question 1.

If we had more time, we would use other feature selection methods as well, either to corroborate or refine feature selection based on MI score. One other method is using hypothesis testing – fitting a linear regression model, for example, based on all features, then looking at the p-values for each feature and using only the features with p-values under a threshold (typically 0.05). These would be the statistically significant features.

# Question 4

## 4.1

For each of these linear regression models, we used the aforementioned selected features. Thus, we are only working with features of constant, carat, length, width, and depth.

For no regularization (OLS), all 5 features remained with non-zero learned parameters.

Average RMSE (OLS): 0.3992176822828949
Average learned params (OLS): {'const': 1.207573380160548e-07, 'carat': 1.2349678204938752, 'length': -0.319235435923662, 'width': -0.010591549849219908, 'depth': -0.0036583501957502236}

For Lasso regression, only the carat feature remained with non-zero learned parameter.

Average RMSE (Lasso): 0.4068718121479253
Average learned params (Lasso): {'const': 0.0, 'carat': 0.9034803355841561, 'length': 0.0, 'width': 0.0, 'depth': 0.0}

Lastly, for Ridge regression, similar to OLS, almost all features remained with non-zero learned parameters. The only difference is the constant term is now zero.

Average RMSE (Ridge): 0.39924936566711466

Average learned params (Ridge): {'const': 0.0, 'carat': 1.2148943139233737, 'length': -0.2967209873980527, 'width': -0.013421872956992867, 'depth': -0.0037921271877426757}

Note that all of these regression models had a similar average RMSE across all 10 folds.

## 4.2

For each of Ridge and Lasso regression, to compute the best penalty term, I used a range of penalty values (0.01, 0.1, 1, 5, 20, 100) and selected the one resulting in the highest $R^2$ value for the test set in a train test split. For Ridge regression, this turned out to be 100, while for Lasso regression, this turned out to be 0.01.

Then, I used these penalty terms to calculate the average RMSE of the validation sets across 10 folds. Based on these RMSEs, the best regularization scheme is OLS regression, having an average RMSE of 0.39922, compared to 0.39925 and 0.407 in Ridge regression and Lasso regression, respectively. OLS and Ridge regression are very close in average RMSE, however.

If performing this experiment once more, I'd expand the range of penalty values since Ridge regression may prefer an even higher penalty term.

## 4.3

For the cases with ridge regularization, feature standardization plays a large role in improving model performance. This is because a feature with a larger scale may have disproportionately large coefficients, and this will dominate the penalty term.

To compare model performance using the scaled versus unscaled features and price target, RMSE doesn't work since the error scale will also differ. Thus, we added $R^2$ scores in the model evaluation process. The average $R^2$ score across all splits was indeed higher when the data was scaled compared to unscaled and when trained using Ridge regression. Specifically, the average $R^2$ score for scaled data was 0.8406, while it was 0.8401 for unscaled data. However, I believe the difference is not significant because we performed feature selection beforehand.

## 4.4

As previously mentioned, p-values help determine statistical significance of a feature's coefficient in a model. An OLS model trained using a library such as statsmodel will provide these p-values for each feature alongside the coefficient.

The meaning of the p-value is the probability of that feature having the learned coefficient (or higher), assuming the null hypothesis is true, where the null hypothesis is that the feature's true coefficient is 0. Specifically, each feature has its own null hypothesis, and in other words, it is that the feature has no effect on the target (same as saying the coefficient is zero).

Thus, when the p-value is low, we know the feature is significant – it has a genuine impact on the target. Thus, the most significant features are the features for which the calculated p-value is low (usually below a threshold value such as 0.05).

# Question 5

## 5.1

The most salient features are carat, length, width, and depth. This is because these are the most significant characteristics that determine a diamond's price and value, logically speaking. We value diamonds by their carat and size, and these four salient features represent these two "common sense" features best.

## 5.2

The second order polynomial transformation works best. We found this by testing all transformations from the second to sixth degree. For each transformation, we trained a Ridge regression model on a train test split, and computed the best score across all degrees on the test split. Then, once we found that the second order performed best, we performed the 10-fold cross-validation experiment used in previous steps. The average RMSE here was 0.375, which is noticeably better / lower than when using linear regression, which is nice to see.

Average RMSE (Ridge): 0.3749243511891282
Average learned params (Ridge): {'1': 0.0, 'carat': 0.3008469302529113, 'length': 0.1208375632194754, 'width': 0.18932772958539335, 'depth': 0.026360794221534967, 'carat^2': -0.36464598202778864, 'carat length': 0.3429282350910052, 'carat width': 0.35425433883704316, 'carat depth': 0.004833868217729844, 'length^2': 0.131626392351658, 'length width': -0.2169652742131773, 'length depth': 0.012807652957123055, 'width^2': -0.0039545636732161394, 'width depth': -0.02192215207974793, 'depth^2': -0.000754661827148128}

Since the second transformation is applied, instead of just having the base features, we have those features, the squared features, and interaction terms between features, since this is what would happen in a quadratic model.

A high order polynomial may indicate overfitting on training data. This is because a high order polynomial will have a more complex curve, meaning that the intricacies between training data points was overfitted to. In consequence, the testing error should be larger.

# Question 6

## 6.1

Best parameters found:  {'mlp__hidden_layer_sizes': (12, 6), 'mlp__alpha': 0.01} Best cross-validation RMSE:  -0.37052294548676334
Test RMSE:  0.3764060721524689

## 6.2

The RMSE was slightly under the linear regression models', indicating similar but slightly worse performance.

## 6.3

The activation function is a linear activation function. This is good for regression problems because it does not restrict the range of output values and can effectively handle the prediction of continuous outcomes. The linear activation function essentially applies no activation and outputs the input directly.

## 6.4

Increasing the depth of a neural network can boost its ability to learn complex patterns but also risks overfitting, where the model performs well on training data but poorly on new, unseen data. Deeper networks face challenges like vanishing or exploding gradients, making them harder to train, and require more computational resources. Additionally, the benefits of adding more layers often diminish after a certain point, leading to diminishing returns on model performance improvement.

# Question 7

## 7.1

Test RMSE: 0.3558660713475074

max_features:  determines the number of features to consider when looking for the best split during the tree building. Lower values can lead to deeper trees with more splits but can also reduce variance and increase bias.

n_estimators: number of trees in the forest. A higher number of trees increases the model's robustness and accuracy, up to a point, beyond which improvements can be marginal. However, it also linearly increases computational cost and memory usage.

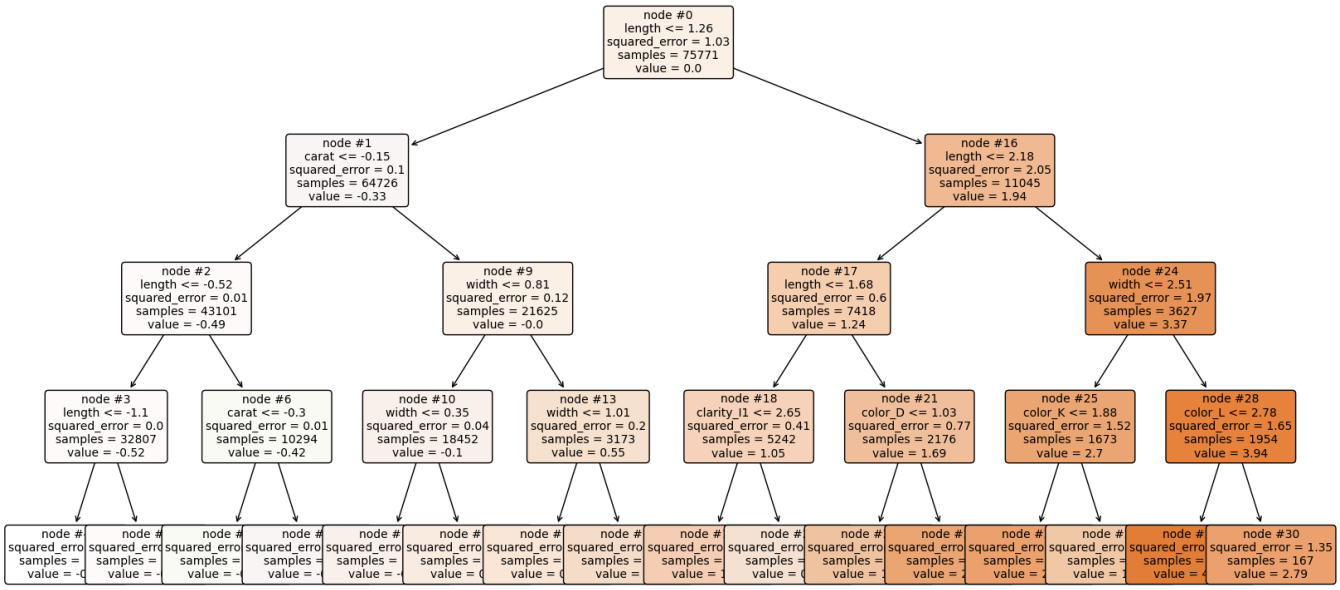
max_depth:  controls the maximum depth of each tree. Deeper trees can model more complex relationships by creating more splits but can also lead to overfitting

## 7.2

Random Forests create non-linear decision boundaries by aggregating the outcomes of multiple decision trees, each splitting the data using simple, linear thresholds on features. Despite each split being straightforward, the collective effect of many such splits across numerous trees enables the modeling of complex patterns. This is enhanced by training each tree on a random subset of the data and using a random subset of features for each split, introducing variability and reducing overfitting.

## 7.3

The feature at the root node is length <= 1.26. This suggests its significant role in explaining the variation in the target variable. The importance of this feature relative to others can be inferred as higher, especially for predicting the target variable, because the decision tree algorithm chose it as the first split. This tree makes sense, as the important features here correspond to the selected features in the earlier exercises, like carat, length, and width are all in the tree.




## 7.4

OOB Error:  0.12551211047396027
R^2 Score:  0.8756231353967725

The Out-of-Bag (OOB) error is an estimation of the model's prediction error. For each tree in the ensemble, the input that was not used to train that tree is used to test it. The OOB error aggregates these test results to give an overall error estimate. This error estimate is valuable because it's obtained without having to withhold a separate test set, making efficient use of the

data. The R2 score is a measure of how well the regression predictions approximate the real data points. An R2 score of 1 indicates perfect agreement between the actual and predicted values, while a score of 0 would mean that the model performs no better than a model that simply predicts the mean of the target variable for all observations. Negative values indicate that the model performs worse than this simple mean model.

# Question 8

## 8.1

1. learning_rate: Controls the speed of model learning. A smaller value makes the learning process more gradual, potentially leading to better performance, but requires more trees (iterations) to train. Search Space: [0.01, 0.05, 0.1]

2. depth: Deeper trees can model more complex relationships but also increase the risk of overfitting and require more computational resources. Search Space: [4, 6, 8]

3. 3. iterations: Number of trees to build. More trees can improve model performance but also increase the risk of overfitting and computational cost. Search Space: [100, 500, 1000]

4. 4. l2_leaf_reg: Coefficient for L2 regularization of the weights. Higher values can help prevent overfitting by making the model more conservative. Search Space: [1, 3, 5, 10]

## 8.2

Best parameters found:  OrderedDict([('depth', 8), ('iterations', 500), ('l2_leaf_reg', 1), ('learning_rate', 0.09985577822694015)])
Best RMSE from CV:  0.13210428775924388
Test RMSE with optimized hyperparameters: 0.1272260996897493

## 8.3

Effect of Hyperparameters

1. Depth: Increasing the depth allows the model to capture more complex patterns, increasing performance. Deeper trees can potentially lead to overfitting since they might capture noise in the training data as if it were a significant pattern. In terms of fitting efficiency, deeper trees are computationally more expensive to grow and can slow down the training process.
2. Iterations: More iterations can increase performance with more additional trees. CatBoost has mechanisms to prevent overfitting, especially when combined with a suitable learning rate and L2 regularization.  Increasing the number of iterations of course also increases the training time.

3. L2 Leaf Reg. Controls the L2 regularization term on the weights, which can help in preventing overfitting by penalizing large weights.
4. Learning Rate: Controls how quickly the model adapts to the problem. This can affect performance and regularization if the learning rate is small and increases time complexity or converges to a suboptimal solution. A higher learning rate typically means the model requires fewer iterations to converge, potentially reducing training time.

# Question 9

## 9.1

The following statistics were computed for each of the files. The first average tweets per hour statistic was computed using `tweet["citation_date"]`, while the second statistic was computed using `tweet["firstpost_date"]`. The second statistic should be the accurate one, based on cross-reference with the actual tweet post on Twitter. The first statistic is the one recommended by the project specification.

Average tweets per hour (#gohawks): 292.598615916955
Average tweets per hour (#gohawks): 173.99382716049382
Average followers per user (#gohawks): 2217.9237355281984
Average retweets per tweet (#gohawks): 2.0132093991319877

Average tweets per hour (#gopatriots): 40.95993031358885
Average tweets per hour (#gopatriots): 34.42313323572474
Average followers per user (#gopatriots): 1427.2526051635405
Average retweets per tweet (#gopatriots): 1.4081919101697078

Average tweets per hour (#nfl): 397.64846416382255
Average tweets per hour (#nfl): 251.64362850971924
Average followers per user (#nfl): 4662.37544523693
Average retweets per tweet (#nfl): 1.5344602655543254

Average tweets per hour (#patriots): 751.9129692832764
Average tweets per hour (#patriots): 449.61326530612246
Average followers per user (#patriots): 3280.4635616550277
Average retweets per tweet (#patriots): 1.7852871288476946

Average tweets per hour (#sb49): 1277.7474226804125
Average tweets per hour (#sb49): 1277.7474226804125
Average followers per user (#sb49): 10374.160292019487
Average retweets per tweet (#sb49): 2.52713444111402

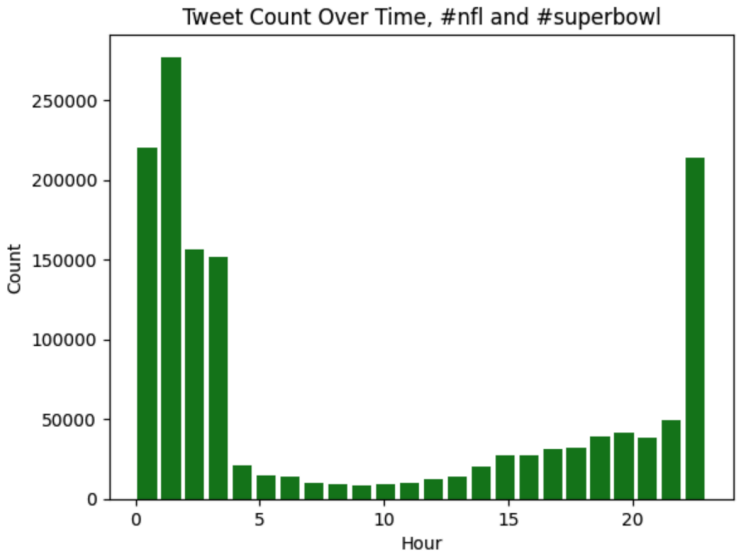
Average tweets per hour (#superbowl): 2074.8940170940173
Average tweets per hour (#superbowl): 1268.3521421107628

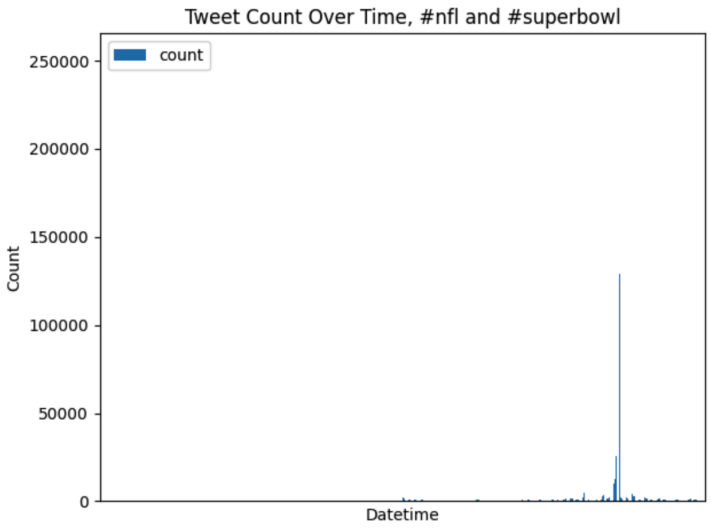
Average followers per user (#superbowl): 8814.96799424623
Average retweets per tweet (#superbowl): 2.3911895819207736

### 9.2

Although there were two methods to plot tweets over time, we decided to plot it by hour and truncate other non-hour attributes. Thus, we are viewing popular times of the day in terms of tweets in #nfl and #superbowl.

If plotting over true date, the range of tweets was found to be several months, making the histogram very sparse.




## Question 10

### Describe your task

For our task, we have selected to create a library that can predict many features/qualities of a tweet. The main motivation behind this task is that this will allow us to exercise many models that we have learned in this course and have interest in further applying. This is because, with multiple prediction tasks, we have many more models we can put to the test, such as different clustering models, regression models, baseline models, and more. Furthermore, we are confident we can generate logical and reasonable results with our models – we do not want to work on a model where the results may be weak and not lead to any solid conclusions.

The specific features we are looking to predict in our library are, first, given a tweet, which hashtag did it come from? This is a multiclass clustering problem that we want to apply clustering models we learned about in the course and compare to the clustering models that we have heard about outside the course.

Next, we want to be able to predict the engagement of a tweet. The purpose behind this is, we have always been interested in what makes any sort of social media post go viral or gain more attention. In this context, this would be what makes a tweet popular and well-received. Shamelessly, we would like for our own tweets and other social media posts to go viral every now and then, so this is an extremely interesting task for us.

Lastly, we want to predict the relative time at which a tweet was posted. Specifically, we have defined relative time to mean occurring during either pre-superbowl, superbowl, or after-superbowl. Initially, we planned to create a more fine-grained breakdown of the time classes, such as pre-superbowl, first quarter, second quarter, halftime, etc., but unfortunately, the exact times at which each quarter begins or ends is not well-documented (ESPN / NFL should work on this in the future, it definitely limits data science analysis potential). The motivation behind this prediction is we want to perform sentiment analysis, since we believe sentiment is greatly important in predicting when a tweet was posted. We have not worked with sentiment analysis in the past, but it is something we are interested in playing around with.

## Explore the data and any metadata

We have explored the data and worked with the metadata to perform our feature engineering. We have decided that the dataset we are provided with is the only one we will need. Many features of the tweet data that we are provided with are helpful for our purposes – the tweet content, post time, engagement activity and numbers, user profiles, and so on. We have looked into the requirements of sentiment analysis and the models we plan to use, and all of these features provided are sufficient.

## Describe the feature engineering process

For predicting the hashtag from which a tweet comes, the process comprises several steps. First, we start by collecting tweet data from various files, each associated with a specific hashtag. This association forms the basis of the label assignment, with each text being assigned a label corresponding to the hashtag of the file it came from. Then, to manage computational resources more efficiently, we implemented a random sampling mechanism. After loading and assigning hashtag labels to the tweets, the next significant step is the transformation of the text data into a numerical format. This transformation is done using HashingVectorizer, which performs tokenization, then converts the tokens into a numerical format. Instead of holding a vocabulary in memory, it applies a hashing function to the tokens. This approach maps tokens to features in a fixed-size vector space, defined by n_features. The primary advantage is the reduced memory. We then do feature hashing, useful for large datasets, which uses a hash function to determine the index for each token in the feature vector, it allows for direct allocation without a lookup table. With the tweet data now in a numerical format, we trained an SGDClassifier to predict tweet labels based on test tweets.

For tweet engagement, the feature engineering leverages both textual and numerical data, employing techniques like sentiment analysis, TF-IDF vectorization, dimensionality reduction, and the combination of various feature types into a unified dataset for training an XGBoost

regressor. Firstly, we loaded the data in and randomly sampled a portion, while removing the bottom 10th percentile and top 90th percentile to account for outliers. Then, sentiment analysis is performed on the tweet texts using the NLTK library's ‘SentimentIntensityAnalyzer’. This tool computes sentiment scores that represent the emotional tone of the texts, ranging from negative to positive. The `compound` score, which aggregates the sentiment metrics into a single score, is extracted for each tweet. This sentiment score is intended to capture the emotional context of the tweet, which can be a valuable predictor of its popularity (measured by ‘impressions’). Next, the textual content of the tweets is transformed into a numerical format using TF-IDFvectorization, which reflects the importance of words within the corpus of tweets. The TfidfVectorizer is limited to the top 1000 features to focus on the most relevant terms and manage computational complexity. To further reduce the dimensionality of the TF-IDF features and improve computational efficiency, Truncated SVD is applied, reducing the feature space to 100 components.  The process then involves combining these reduced textual features with selected numerical features: the presence of URLs, the number of hashtags, the length of the tweet, the hour of posting, and the sentiment score. Finally, the combined features serve as input for training an XGBoost regressor model, with the goal of predicting the ‘impressions’.

Lastly, for predicting the relative time at which a tweet is posted, we are using lower-dimensional features. We believe TF-IDF analysis, which although would produce good results, is overkill here, in terms of time spent processing and model complexity. The reason is that a tweet posted before the Superbowl will likely show some sort of anticipation, excitement, or anxiety. A team’s fan would tweet about their anticipation for the Superbowl happening and their favorite team winning. If the fan is supporting the underdog team, they may tweet about their anxiety for the big day. In general, football viewers will show excitement about the upcoming Superbowl. Furthermore, after the game is over, there may be celebration and anger in the tweets. The winning teams will be tweeting about their celebrations or happiness, while the losing teams will be expressing anger or sadness or similar. Lastly, during the Superbowl, there can be many tweets showing tension due to the state of events. Thus, we believe sentiment is extremely important in predicting time of tweet posting. We are also using emoji count as a feature. The reason for this is that emoji usage is highly correlated with intense emotions, which can help categorize a tweet to have been posted during the game. For example, I would post a tweet with many celebration emojis upon my team scoring the first touchdown. Lastly, we are using engagement activity as features. Specifically, the number of retweets, favorites, impressions, and replies. We believe greater engagement activity is generally correlated with tweets occurring during outside of the Superbowl since, during the Superbowl, there will be more focus on tweeting your own thoughts and watching the game. Before and after the Superbowl, fans will be engaging in conversation and debate.

## Generate baselines for your final ML model

For predicting tweet hashtags, we used the SGDClassifier(loss='log', penalty='l2', max_iter=5000, tol=1e-3, random_state=42), as the baseline. This specifies the logistic regression model, handling multi-class classification problems through the "one-vs-all" strategy.

For predicting tweet engagement, we used xgb_regressor = XGBRegressor(n_estimators=100, learning_rate=0.005, max_depth=5, random_state=42), to handle estimating.

For predicting time of tweet, we decided to use a neural network as the baseline. For the neural network, we used a three-layer architecture, where the first two layers used 64 input neurons and the relu activation function, while the last layer was the output layer with 3 neurons, using the softmax function. Upon training and testing, the accuracy for the test set was 0.725, and we kept this as the baseline since neural networks trained on large datasets typically have good performance, when the layers and architectures are designed sufficiently.

## Evaluation

For predicting tweet hashtags, our model did so with an accuracy averaging 0.823224 over 20 different random data samples, proving to be quite consistent and effective. The accuracy was close to that of our baseline. For predicting tweet engagement, given that the average impressions is around 6909, our model predicts with a MSE error of about 1135, over 20 different random data samples, which is reasonably effective. The accuracy in these two prediction tasks were close to the accuracies generated by our baseline, which is good. We believed our model implementation was solid and would lead to good results, so our accuracy scores were as expected.

For predicting the time of tweet, we used several different methods to perform multiclass classification into one of the three tweet time classes – before the Superbowl, during the Superbowl, and after the Superbowl. Specifically, we used a few clustering methods – hierarchical clustering and KMeans – as well as a Naiive Bayes classifier. We wanted to see how these classification methods would perform, given that they had specific use cases as shown in previous projects in the course. With hierarchical clustering, there is the ability to cluster if data is naturally separated in hierarchies, while KMeans finds the most similar clusters using some distance metric. However, with all of these methods, we found that the accuracies were extremely low. All the accuracies were near 0.10 ~ 0.20. Specifically, when using agglomerative clustering, the accuracy was 0.219. When using KMeans, the accuracy was 0.213. When using Naiive Bayes classification, the accuracy was 0.082. Upon seeing these results, we performed further research and consideration of classification techniques used outside of the course. We decided to use a Logistic Regression model with lbgfs solver to perform the classification. What we found was that this model performed well and met our Neural Network baseline accuracy! Its accuracy on the test set was 0.737, which exceeds the baseline accuracy of 0.725 and thus is comparable. Confusion matrices for each method were also plotted.

## Conclusion

This is the tweet prediction library we have implemented. Overall, it was a nice experience testing utilizing a portion of the methods we have used in the previous project, in a task we chose ourselves.