Question 1:

We choose the diamond dataset for the following regression analysis.

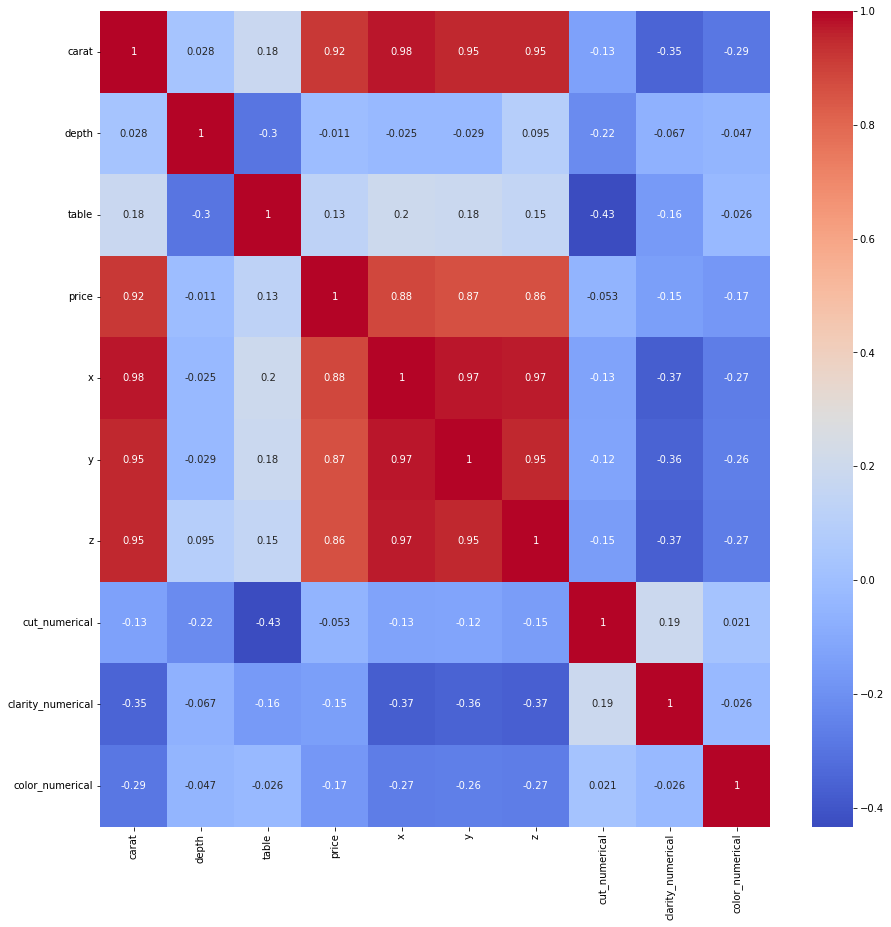
* For the categorical to numerical label conversion:

Cut: {Ideal: 5, Premium: 4, Very Good: 3, Good: 2, Fair: 1}

Clarity: {I1: 1, SI2: 2, SI1: 3, VS2: 4, VS1: 5, VVS2:6, VVS1: 7, IF:8}

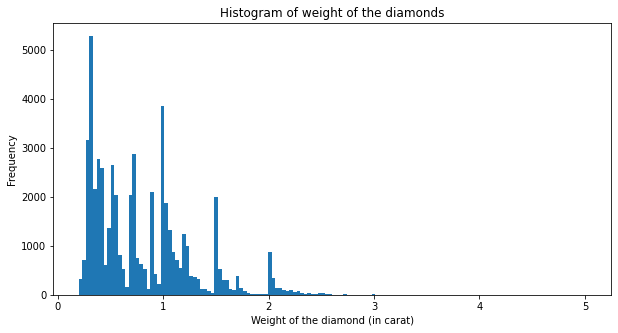
Color: {J: 1, I: 2, H: 3, G: 4, F: 5, E: 6, D: 7}

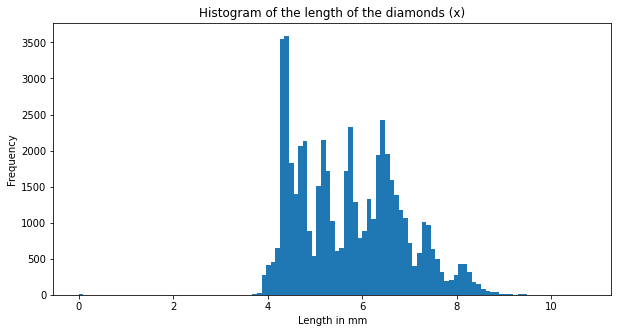
* Question 1.1:



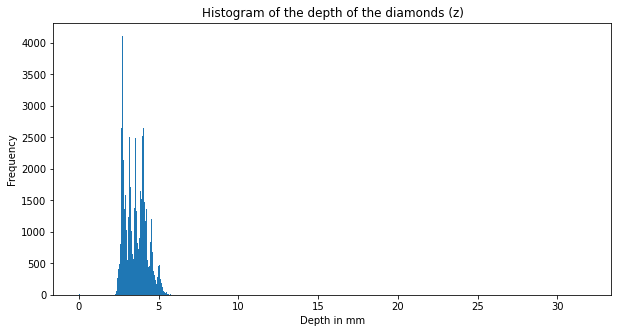
As shown in the heatmap, the carat feature has the highest correlation. The x, y, and z features correlate highly with the target variable. The correlation patterns suggest carat, x, y, and z contain more information in predicting the price of a diamond, compared to the features having a lower correlation.

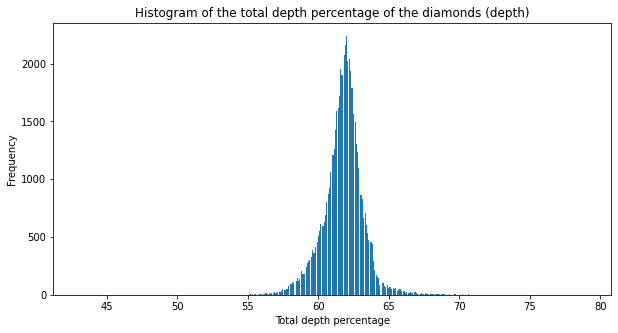
* Question 1.2:

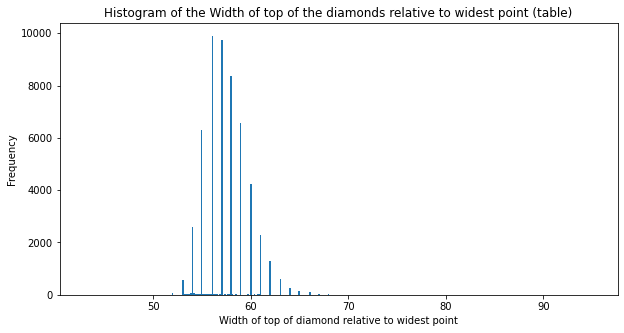


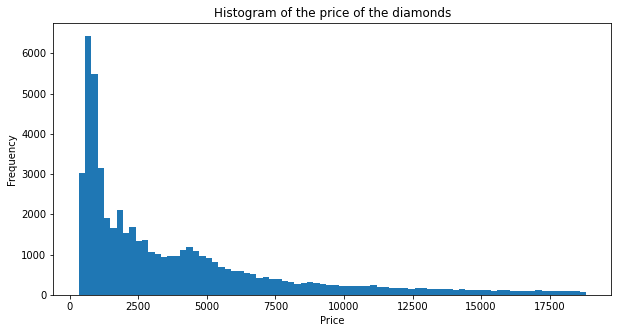




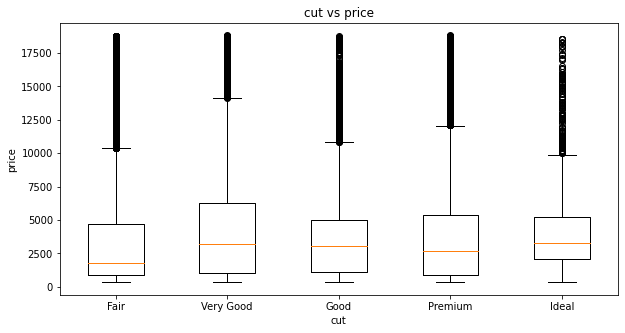




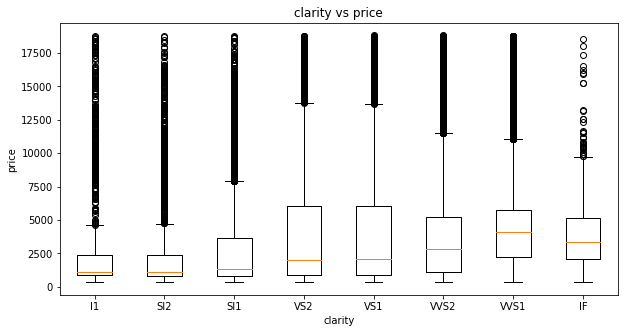




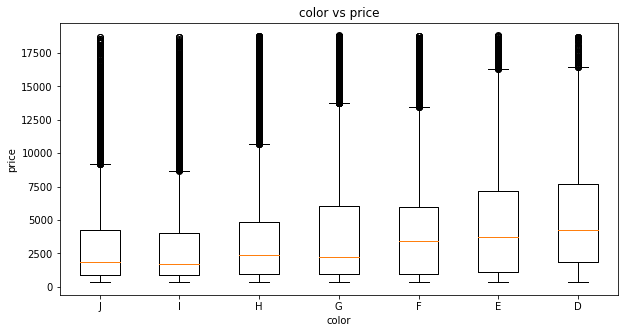
For the distribution of a feature with high skewness, we can apply a concave function like log, ln, sqrt, and so on to fix the high skewness.

* Question 1.3:

The average diamond price remains relatively constant as the cut quality increases. However, the lower quartile of the diamond price increases as the cut quality increases. This indicates that the cut quality guarantees the price floor of a diamond.

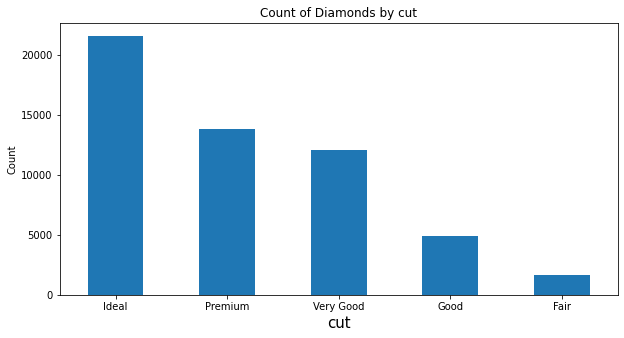


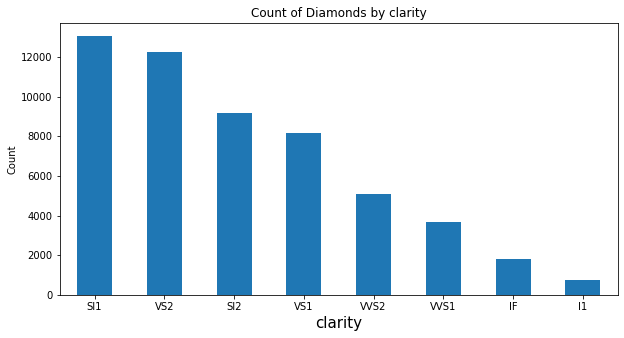
The average diamond price increases as the clarity quality increases. The lower quartile of the diamond price also increases as the clarity quality increases. Furthermore, the number of outliers decreases as clarity quality increases

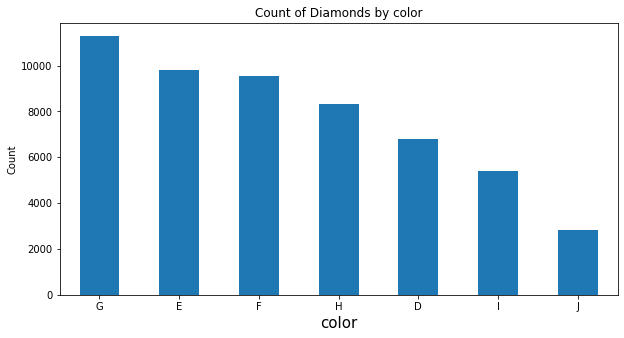


The average diamond price increases as the color quality increases. The lower quartile of the diamond price remains constant as the color quality increases. However, the upper quartile of the diamond price increases as the color quality increases. The number of outliers decreases as the color quality increases.

* Question 1.4:







Question 2:

* Question 2.1(Standarize the feature columns):

array([[-1.19816781, -0.17409151, -1.09967199, ..., 0.98147332,

-1.24521508, 0.93716275],

[-1.24036129, -1.36073849, 1.58552871, ..., 0.08588908,

-0.63809506, 0.93716275],

[-1.19816781, -3.38501862, 3.37566251, ..., -1.70527938,

0.57614496, 0.93716275],

...,

[-0.20662095, 0.73334442, 1.13799526, ..., -0.80969515,

-0.63809506, 1.52502147],

[ 0.13092691, -0.52310533, 0.24292836, ..., 0.08588908,

-1.24521508, -0.8264134 ],

[-0.10113725, 0.31452784, -1.09967199, ..., 0.98147332,

-1.24521508, 1.52502147]])

* Question 2.2:

If we select features that yield better regression results, it may boost the performance of our models by having less test RMSE. This is because we prevent overfitting the training data by dropping irrelevant features. However, it is not valid for all the models. Throwing away the features with a low dependency may cause a loss of information. Therefore, it can cause the lower performance of some regression models.

The two features for the diamond dataset with the lowest MI are depth and table features.

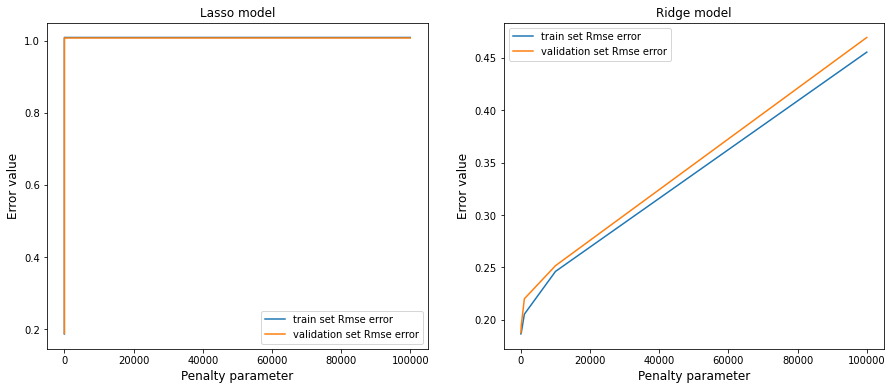
We took the logarithm of the price (target variable) for the following analysis to speed up the Neural Network regression. Therefore the RSME becomes relatively small but still comparable between different models.

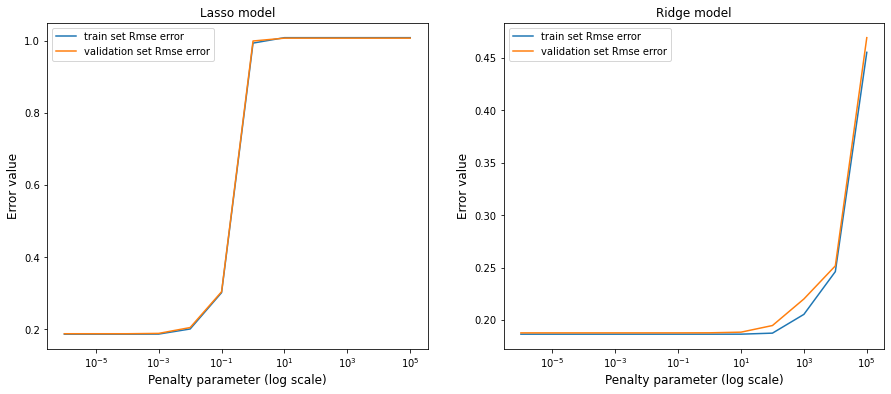
Question 3:

* Out-of-Bag Error: In a decision tree, each tree is trained on a random sample of the data. However, the unused data in the training process is called Out-of-Bag data. Out-of-Bag Error means the model's prediction error using this Out-of-Bag data. In other words, Out-of-Bag Error is the number of incorrectly classifying the Out-of-Bag data.
* score: This measures how well the model fits the data. To be more specific, score ranges from 1 to 0. score = 1 means a perfect fit, while score = 0 means not fitted at all. The ratio between the variance of the predicted data and the actual data determines the score.

Question 4:

* Question 4.1:
  + Linear regression: OLS model minimizes the sum of squared residuals between the predicted values and the actual values. Therefore, it has no regularization scheme.
  + Lasso regression: it adds the absolute value of the penalty term to the cost function and shrinks some parameters to zero. Therefore, this regularization scheme removes irrelevant features from the model.
  + Ridge Model: it adds the squared value of the penalty term to the cost function and shrinks the parameters to near zero. Therefore, this regularization scheme keeps all the features from the model. It can prevent overfitting problems and keep the model relatively stable.
* Question 4.2:
  + The following figures show the RMSE from the train and validation sets in both regular and log scales. As we can see, the optimal regularization scheme for the Lasso model is and for the Ridge model.





* + After comparing the RSME result, shown in the table, the best regularization scheme with the optimal penalty is the Ridge model with penalty parameter = . However, it is worth noting that the RSME for each model is almost identical. The choice may vary if we extend the range of penalty parameter range.

|  | OLS | Lasso (penalty parameter = ) | Ridge (penalty parameter = ) |
| --- | --- | --- | --- |
| RSME | 0.1876069785438673 | 0.1876069786112494 | 0.1875878431574759 |

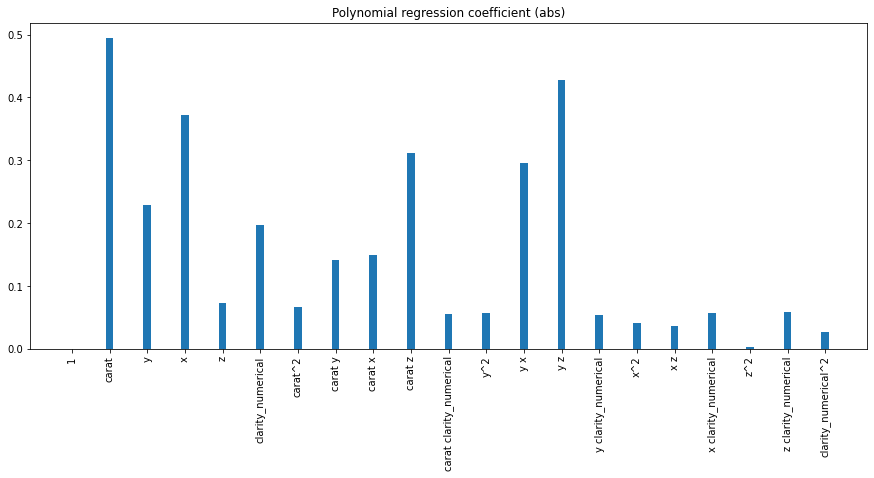
* + Furthermore, we have also tried the GridSearchCV function for optimal regularization. The result is the same.
* Question 4.3:
  + The feature standardization may influence the model performance. For example, if each feature has a different scale. The regularization scheme can penalize some features unfairly. However, it depends on the dataset. We believe standardization does not play a role in model performance for the diamond dataset because all the features have a similar scale ( to ) .
  + To justify our answer, we have computed the RSME for regular and standardized features in the Ridge model.

|  | w/o standardized features | w/ standardized features |
| --- | --- | --- |
| Ridge train | 0.18637135150134565 | 0.18637135150134562 |
| Ridge validation | 0.18760697859825154 | 0.18760697861124945 |

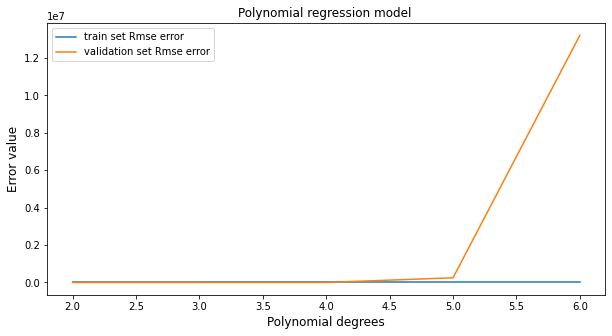
* Question 4.4:
  + P-value is a crucial concept in the basic statistics. It is essential in the null hypothesis test to determine the statistical significance with a given confidence interval. For determining whether a feature is significant in a 95-confidence interval. We can reject the null hypothesis and suggest that the feature is statistically significant if the p-value is less than the critical value.

Question 5:

* Question 5.1:
  + We performed polynomial regression (degree = 2) on the top 5 features selected in part 3.1.4. They are: 'carat', 'y', 'x', 'z', and 'clarity\_numerical'.
  + The most salient features are the features that have the most substantial influence on the target variable. Therefore, the features having a high coefficient in the polynomial regressions are the most salient. As shown in the figure, ‘carat’, ‘y\*z’, and ‘x’ are the most salient features.



* Question 5.2:
  + Degree = 2 is the best polynomial degree. We find this optimal degree by selecting the model with the lowest RMSE in the validation set in the 10-fold CV.
  + As shown in the figure, a very high-order polynomial implies the model is overfitting the training data. The model becomes specific on the training data and performs poorly on the testing data. As the degree increase, the RMSE increases from 10^-1 to 10^7. This is an indication of overfitting.



Question 6:

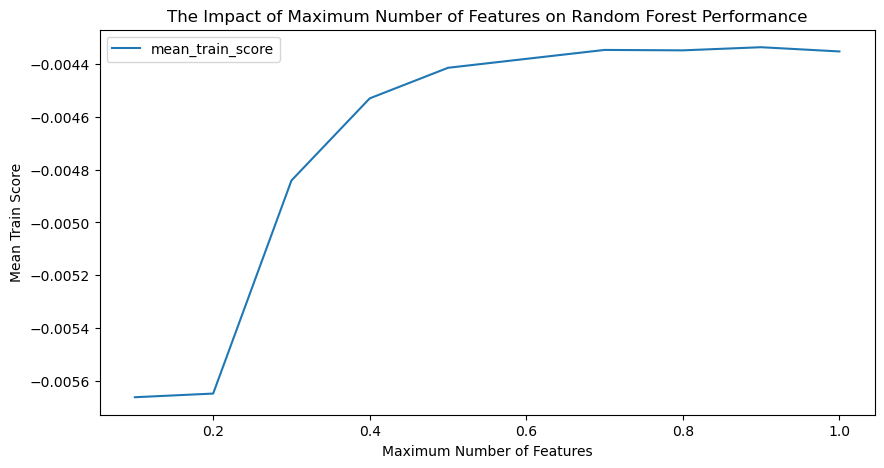
* Question 6.1
  + The 5 hidden layer sizes we set for the grid search are:  
    (10,), (20,), (10,10), (20,20), (40, 40)
  + The 4 penalty parameters we set for the grid search is:  
    10\*\*-3, 10\*\*-2, 10\*\*-1, 1
  + The best hyperparameter pair we found for the neural network is with hidden layer size of (40, 40) and penalty parameter of 0.1.
* Question 6.2
  + Here is a table of average RMSE for different linear regression models:

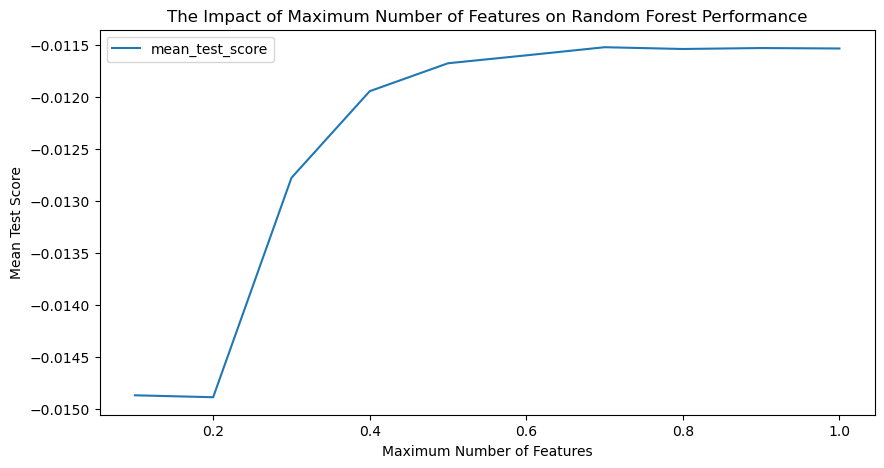
|  | **Average RMSE for Training Set** | **Average RMSE for Validation Set** |
| --- | --- | --- |
| **Ordinary Least Squares** | 0.18637135150134565 | 0.18760697854386735 |
| **Lasso Regression** | 0.18637376397656663 | 0.18758784315747593 |
| **Ridge Regression** | 0.18637135150134562 | 0.18760697861124945 |
| **Neural Network (MLPRegressor)** | 0.018089401816264884 | 0.01926359854015128 |

* + Note that for the lasso regression, we set alpha=10\*\*-4, for the ridge regression, we set alpha=10\*\*-6, and for the neural network, we set hidden\_layer\_sizes=(40, 40) and alpha=0.1.
  + We can see that the MLPRegressor performs significantly better than the three different linear regression models above.
  + The neural network model yields better performance since our data is complex and has non-linear relationships between input and output. Neural network models can capture these nonlinear patterns within the dataset, while linear regression models only focus on linear patterns.
* Question 6.3
  + We use ReLu, which is the default setting in sklearn, as the activation function for our output since the output we predict is a numerical value representing the price of a diamond. The value is a continuous nonnegative value. The output of ReLu is always greater than or equal to 0. To make the output more smooth and more continuous, we can also use the softplus activation function.
* Question 6.4
  + If we increase the depth of the network too far, the problem of overfitting will occur. The overall accuracy of the predicted result will not improve much, while the model training will be more time-consuming.

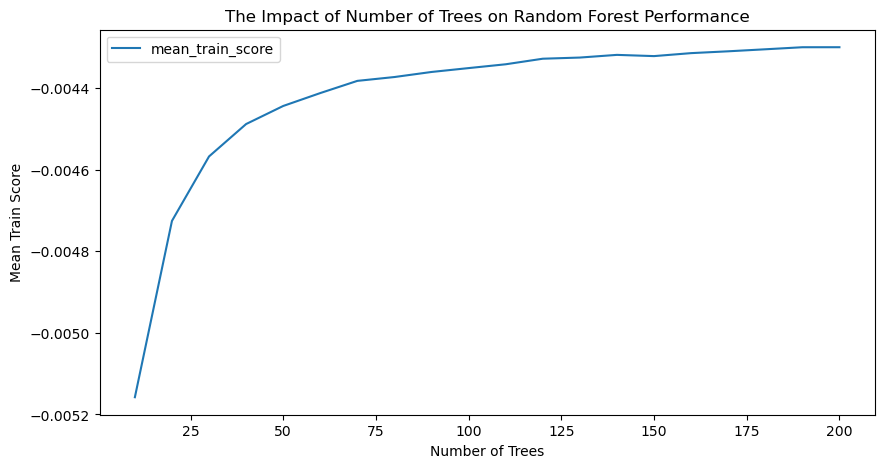
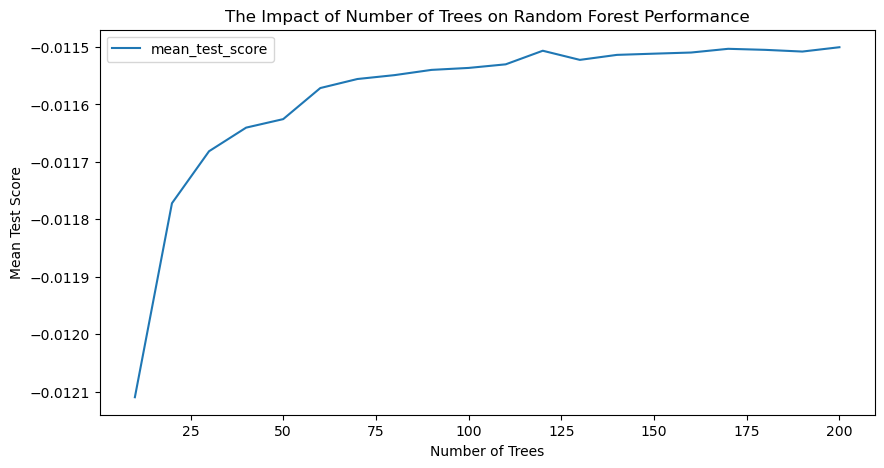
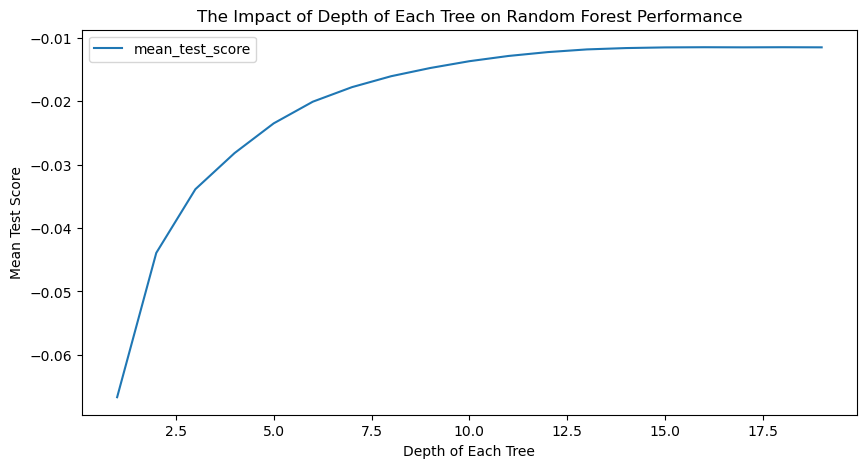
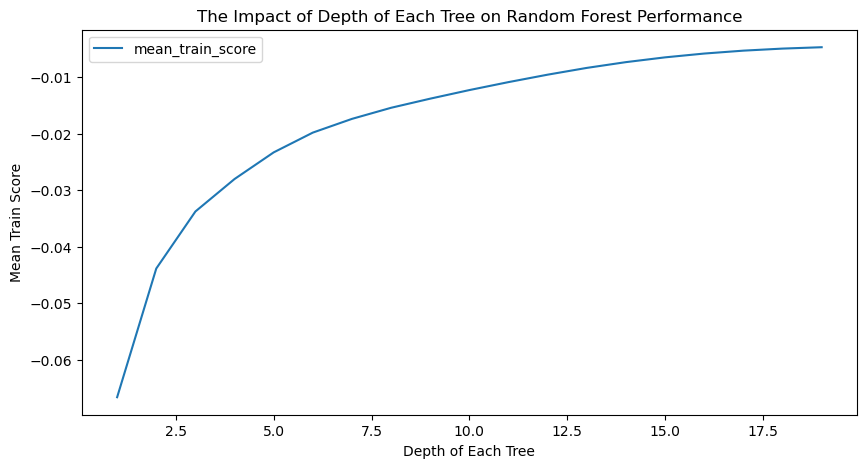
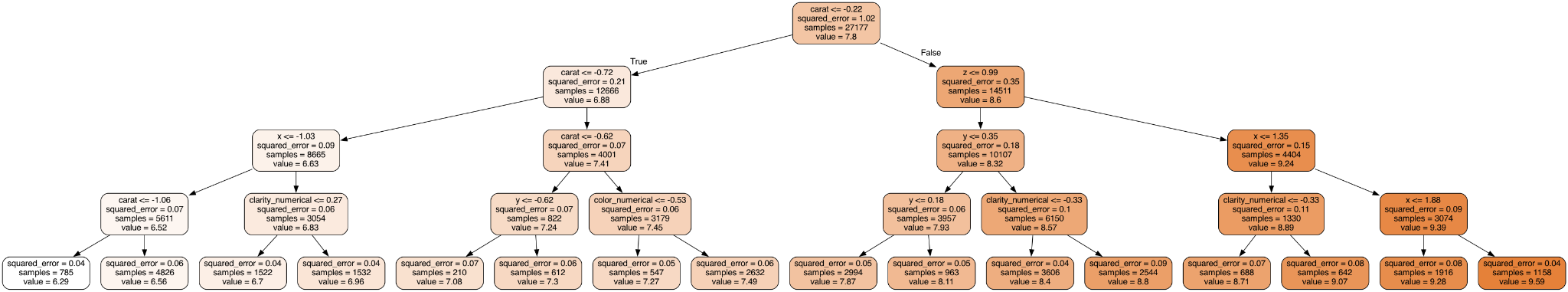
Question 7:

* Question 7.1
  + To illustrate the effects of a certain hyperparameter on the overall performance of the random forest model, we set all other parameters to be default value and only change one parameter at a time in grid search and plot the average RMSE to visualize the effects
  + We first change the max\_features from 0.1 to 1.1 with the step size of 0.1

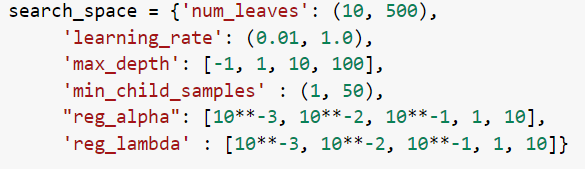




From these two plots, we can find that increasing the maximum number of features will increase the negative mean square error which indicates improving performance. Since the negative RMSE score flattens as the maximum number of features gets to 0.6, the regularization effect of max\_features happens at the value of 0.6

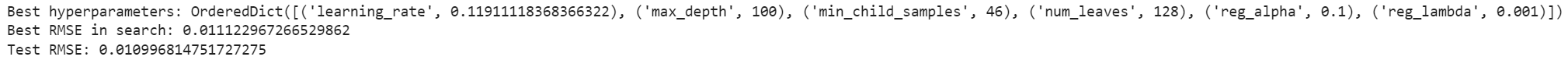
* + We change the number of trees, n\_estimators from 10 to 210 with a step size of 10 From these two plots, we can see that increasing the number of trees can also contribute to the increase in train and test scores. It can also improve the overall performance of the random forest model. Since the negative RMSE score increases dramatically when the number of trees is smaller than 100 and increases slowly when it is greater than 100, setting the n\_estimator value does help prevent the model from overfitting. The regularization effect is more evident when n\_estimators > 100.
  + We change the depth of each tree, max\_depth, from 1 to 20 As we increase the depth of each tree, the negative mean square error also increases. Thus, increasing the depth can also improve the performance of the model. In the training score plot, as the depth of each tree increases, the score also increases while the slope of the curve decreases. Thus, max\_depth also has regularization effects during model training.
* Question 7.2
  + Since each individual tree in the random forest spits out a class prediction, and there is only a low correlation between each decision tree model, the individual prediction error among a decision tree will not affect other trees in the forest. The final ensemble predictions can be more accurate, and the combined decision boundaries are highly non-linear. The threshold we set on a feature at each layer only limits the extreme boundary of each individual tree.
* Question 7.3
  + “carat” is selected for branching at the root node. We can say that “carat” is the most significant feature compared with other features since a more important feature will be picked at higher nodes for branching. The plot shows that carat, z, x, y, and clarity\_numerical are selected as the most important features. It is consistent with what we get in previous parts.
* Question 7.4
  + For the random forest model with max\_features=0.6, n\_estimators=75, and max\_depth=10, we get that the OOB Score is 0.9890012635274665 (OOB error is 0.010998736472533466) and the score is 0.9894943021855327.
  + As we discussed in Q3, OOB score evaluates the performance of each tree on its corresponding unused training set, while score indicates the proportion of the variance in the output explained by the model.
  + The OOB error we get here is significantly low which shows that the decision trees in the random forest model perform well with unseen data. The score of 0.9895 is close to 1 showing that the model concludes the patterns of the training data well and captures most of the characteristics within the training data. Combining these two, we can conclude that the model we generate has good overall performance.

Question 8:

* In this question, the LightGBM boosted tree method is applied to the diamond data. A Bayesian optimization search is performed to analyze the importance of several hyperparameters of the LightGBM.
* Question 8.1
  + A search space is designed with hyperparameters of ‘num\_leaves’, ‘learning\_rate’, ‘max\_depth’, ‘min\_child\_samples’, and ‘n\_estimators’, which are considered important. The value range of each hyperparameter is shown below.
* Question 8.2
  + The Bayesian optimization is performed, corresponding to the RSME for the above search space. The best hyperparameter set is gained and shown below.

| num\_leaves | 128 |
| --- | --- |
| learning\_rate | 0.119 |
| max\_depth | 100 |
| min\_child\_samples | 46 |
| reg\_alpha | 0.1 |
| reg\_lambda | 0.001 |

* + Best RMSE in Bayesian optimization = 0.01112
  + RMSE on test set = 0.01099



* Question 8.3
  + num\_leaves: This parameter defines the maximum number of tree leaves in a single tree. It determines the complexity of the model. A high value of num\_leaves will result in the model being too complex and overfitting, while a low value will result in the model is too simple. A good num\_leaves will help the performance.
  + learning\_rate: This parameter determines the step size during gradient descent. A lower learning rate can result in lower fitting efficiency but can help avoid local minima and help generalization.
  + max\_depth: This parameter determines the depth of the trees. It also affects the complexity of the model. A high value of max\_depth will make the model complex but might result in overfitting if it’s too complex. On the other hand, a higher value of max\_depth will result in lower fitting efficiency and take a longer time to fit. A good max\_depth will help the performance of the model.
  + min\_child\_samples: This parameter determines the minimum number of samples split by a node on a tree. A low value of min\_child\_samples can result in overfitting because each split has too few child samples to be statistically significant. A good min\_child\_samples can help the performance.
  + reg\_alpha: This parameter determines the L1 regularization on weights, which can help with regularization
  + reg\_lamda: This parameter determines the L2 regularization on weights, which can help with regularization

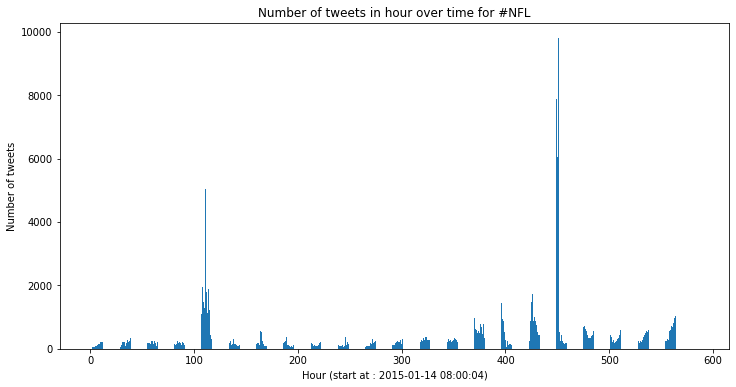
Question 9:

* Question 9.1:

|  | Average number of tweets per hour | Average number of followers of users posting the tweets per tweet | Average number of retweets per tweet |
| --- | --- | --- | --- |
| #gohawks | 292.598615916955 | 2217.9237355281984 | 2.0132093991319877 |
| #gopatriots | 40.95993031358885 | 1427.2526051635405 | 1.4081919101697078 |
| #patriots | 751.9129692832764 | 3280.4635616550277 | 1.7852871288476946 |
| #sb49 | 1277.7474226804125 | 10374.160292019487 | 2.52713444111402 |
| #SuperBowl | 2074.8940170940173 | 8814.96799424623 | 2.3911895819207736 |
| #NFL | 397.64846416382255 | 4662.37544523693 | 1.5344602655543254 |

* Question 9.2:





Question 10:

* Description of the task:
  + In the training tweet data provided, we notice that many exciting properties of tweet text with different hashtags can be analyzed. Our customized task is the following.
  + We will build three libraries to make predictions of a given tweet. They are predictions of the number of retweets, the team tweets the author supports, and the relative post time of a tweet. To be more specific, we will conduct different feature engineering processes and trained models for building different libraries. The detailed feature processing and choice of baseline models will be discussed in each sub-section.
* Exploring the data:
  + There are six sets of raining tweet data with different hashtags. We decided to use #SuperBowl for our customized task because it contains the most significant data size and tweets from different team fans.

We conclude the following characteristics in data inspection:

* + There are 1213813 tweets in the #SuperBowl files.
  + Tweets contain not only English tweets but also tweets in other languages.
  + Some of the tweets contain only emojis.
  + Provided numerical features of tweets are limited, which need to be taken care of in the feature processing.
  + The distribution of Patriots and Hawks mentioned times:



## Predicting the number of retweets

* Features Engineering Process:
  + The first compelling features engineering to use tf-idf matrix to extract the features from the tweet text and reduce its dimension to predict the number of retweets of a given tweet. However, this feature engineering is not likely to give accurate results. The number of retweets cannot be predicted solely depending on the text features. For example, if a tweet user who has only ten followers posts an identical tweet as a Twitter celebrity. The number of retweets will be completely different even though the text features are identical
  + Therefore, we decide to manually extract and generate the numerical features from a given tweet. We need features that reflect a user's influence and the tweet's content. The features we created are the following:
    - active factor: the number of posts by a user during the Superbowl game.
    - number of words: the number of words in a tweet.
    - followers: the number of followers of the author of the tweet.
    - sentiment score: the score of a tweet after the cleaning process, lemmatization, and sentiment analysis.
* Thorough Evaluation:
  + Read in data from tweets\_#superbowl.txt and convert it to DataFrame format.
  + As mentioned, there are 1213813 tweets in the txt file. Using all of them for training will cause a considerable computational burden. Therefore, random.sample function was used to load a portion of data in the order of 10^4.
  + Filter out the foreign language tweet by langdetect.
  + Extract numerical features: active factor, number of words, and followers.
  + Apply the cleaning and lemmatization function on the tweet text.
  + Conduct the sentence analysis on cleaned and lemmatized text. It is worth noting that the sentiment score is calculated after cleaning and lemmatizing the text. This is because lemmatization reduces the number of unique words in the text. As a result, the accuracy of sentiment analysis algorithms is improved by reducing the sparsity of the data.
  + Standardize the numerical features by StandardScaler.
  + Train a Neural Network regression model and apply GridSearchCV to calculate the best RSME for 10-fold cross-validation by sweeping the parameters in the model, such as hidden layer sizes and penalty factors.
* Baselines For the ML Model:
  + Multiple baselines were used for our ML model. To be more specific, a decision tree model (alternate network architecture) and was used in addition to the simple random baseline.
  + The RMSE shows the performance ranking: Neural Network > Decision Tree > Random. (Their RMSE are: 9.2, 11.2, and 301.8, respectively) for retweets prediction.
  + The parameters for our trained neural network regression model are {'alpha': 10, 'hidden\_layer\_sizes': (20,)}.

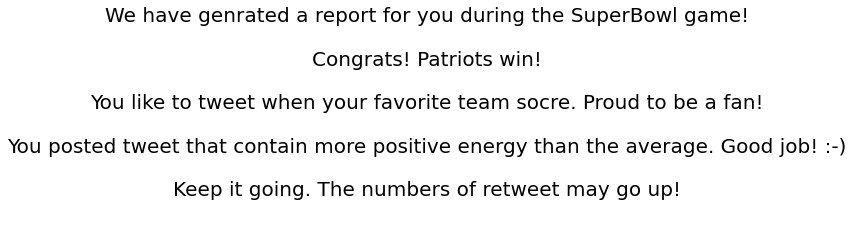
## Predicting the fan base:

* Features Engineering Process:
  + For fan classification, we used two other datasets, #gohawks, and #gopatriots, since those who add these two hashtags have a great possibility of being a fan of one of the teams. We combined these two datasets, adding team labels to each data group, and used the combined dataset to train the neural network and decision tree models. This task could then be generalized to a binary classification problem.
  + The feature we used for fan base prediction was the ‘title’ field of each tweet data, which content is the same as the ‘content’ subfield within the ‘tweet’ field. Since the classification is text-based, we first did the content cleanup to remove any URLs, special characters, and punctuation. Only alphabet numerical characters were kept.
  + Then, we convert the raw text data to a matrix of TF-IDF features. The term frequency-inverse document frequency is a good way of summarizing the pattern of a tweet. We also apply the SVD dimension reduction to reduce the complexity of the data to prepare the data for the following model training.
* Thorough Evaluation:
  + Data reading process is similar to the one used in predicting retweet numbers.
  + For fan classification using the neural network model, the hyperparameters obtained from the grid search are {'alpha': 0.0001, 'hidden\_layer\_sizes': (40, 40)}, with the best RMSE of 0.02497. For the decision tree model, the hyperparameters {'max\_depth': 9, 'min\_samples\_leaf': 4, 'min\_samples\_split': 10} yielded RMSE of 0.03374.
* Baselines For the ML Model:
  + For fan classification, a neural network model and a decision tree model were also implemented. The RMSE metrics show that the neutral network yields a better RMSE.

## Predicting the relative time

* Features Engineering Process:
  + In predicting the relative time of a given tweet, we assumed that there are relationships between the tweet and the time. Rather than the numeric feature in ‘Predicting the number of retweets’, we assumed that the time is more likely to be related to the content of the given tweet.
  + As a result, we used the contents of tweets as the features for the prediction. In processing the features, we first cleaned up the content by dropping URLs, special characters, and punctuation. Only alphabet numerical characters were kept. Then, we performed the TF-IDF conversion to turn the raw text data into feature matrices. Thirdly, we used SVD to reduce the dimension of matrices to improve the fitting efficiency.
  + We used the timestamps as the labels. In order to predict the relative time rather than absolute time, we first read out the ‘timestamp’ from the data frame, then we calculate the time offset against the minimum value in the timestamp and create a new list as the relative time.
* Thorough Evaluation:
  + Read in data from tweets\_#superbowl.txt and convert it to DataFrame format.
  + Apply the cleaning and lemmatization function on the tweet text.
  + Performed TF-IDF conversion for feature extraction.
  + Performed SVD for dimensionality reduction.
  + Calculate the timestamp offset to get the relative time as the label.
  + Create a grid search for the neural network through GridSearchCV and MLP regressor. It generated the best score of RMSE of 274842.
  + Create a grid search for the decision tree through GridSearchCV and decision tree regressor. It generated the best score of RMSE of 256222.
* Baselines For the ML Model:
  + A simple random baseline is used with the result of 804502.
  + The RMSE shows the performance ranking: Decision Tree > Neural Network > Random (their RMSE are: 256222, 274842, and 804502, respectively) for relative time prediction.
  + The best parameters for our trained decision tree regression model are {'max\_depth': 3, 'min\_samples\_leaf': 4, 'min\_samples\_split': 2}.

## Creative Task (Data profiling):

* Starting Point and Motivation:
  + After we built the prediction libraries, we decided to use this data to profile a given user based on their tweet and our ML model prediction.
  + This is a new trend in many applications like Spotify and Instagram. For example, Spotify users get their listening statistics report at the end of 2022. In the report, users can realize what genres they like the most, what time of the day they are on the app, and what songs they may like. Users can share this report with their friends. As a result, Spotify can attract more new users from competitors like Apple music.
  + We are motivated by this idea and made a simple user profile based on their tweet content and activity during the super bowl game.
* Thorough Evaluation:
  + We first use our ML model to predict what fan base a user belongs to. This process requires high accuracy because a user may get mad if a wrong prediction is made.
  + If a user’s favorite team has won, we generate a report congratulating the user. If a user’s favorite team has lost, we generate a report making the user feel better.
  + We also predict the relative time of the user’s post. If a user likes to tweet during the game, we can label this user as active. If a user likes to tweet after the game, we can label this user as busy watching the game so that he/she does not use Twitter during the game or just view the tweets.
  + We analyze a tweet’s sentiment score to give suggestions to a user so that he/she can get more likes or retweets.
* The report template:
  + With proper animation and smooth transitions in the UI design, we believe that this app feature can play an important role in attracting more users to use Twitter over other social platforms like Tik-Tok.