UCLA

ECE 219 - Large-Scale Data Mining: Models and Algorithms

Winter 2024

Project 4: Regression Analysis and Define Your Own Task!

Sunday, March 17, 2024

Authors:

Gian Zignago (UID: 706294998)

Xinyi Pan (UID: 006302303)

QUESTION 1.1 Plot a heatmap of the Pearson correlation matrix of the dataset columns. Report which features have the highest absolute correlation with the target variable. In the context of either dataset, describe what the correlation patterns suggest.

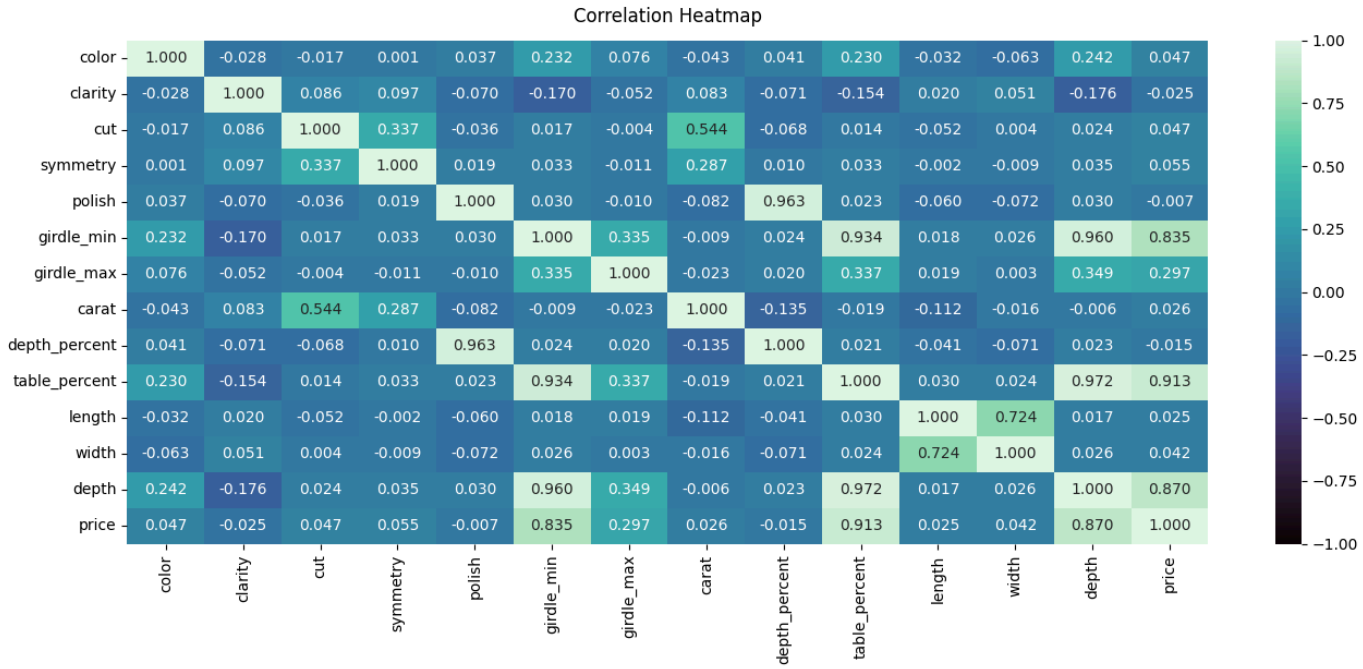


Figure 1: Heatmap of the Pearson Correlation Matrix of the Diamond Dataset Columns

As listed in section 2.1, the target variable is each diamond's price in US dollars. Below is the "price" column of the Pearson correlation matrix isolated to show which features have the highest absolute correlation with the price variable.

Figure 2: "Price" Column of the Pearson Correlation Matrix

As can be seen, the three features with the highest absolute correlation with price are 'table_percent', 'depth', and 'girdle_min'. While 'girdle_max' has a correlation of 0.29 with price, the remaining variables have negligible correlation with the target variable. 'Polish', 'depth_percent', and 'clarity' are the only variables that have a negative correlation with price.

The correlation patterns suggest several discrete points. The feature 'table_percent' has the highest positive correlation with price, which indicates that as the table percentage of a diamond increases, the price tends to increase as well. The table percentage relates to the width of the diamond's table (the flat top facet) compared to the width of the diamond itself. According to the dataset, a higher table percentage is indicative of certain cuts or shapes that are more desirable or more costly to produce. The feature 'depth' is also positively correlated with price, which suggests that the depth of a diamond – its height in proportion to its width – could be an indicator of a diamond's desirability or quality, impacting its price. 'girdle_min' is the third feature that has a significant positive correlation with price. The girdle is the widest part of the diamond, and this measurement may reflect certain desirable proportions that are valued in the diamond market.

Other variables showing negligible correlations imply that these features do not have a strong linear relationship with price within the dataset. This doesn't necessarily mean they have no effect on price, but rather that any effect is not linear or is possibly masked by other factors. The negative correlations of polish, depth_percent, and clarity with price, though not strong, suggest that better polish or clarity ratings, or deeper diamonds, are associated with slightly lower prices in this particular dataset.

QUESTION 1.2 Plot the histogram of numerical features. What preprocessing can be done if the distribution of a feature has high skewness?

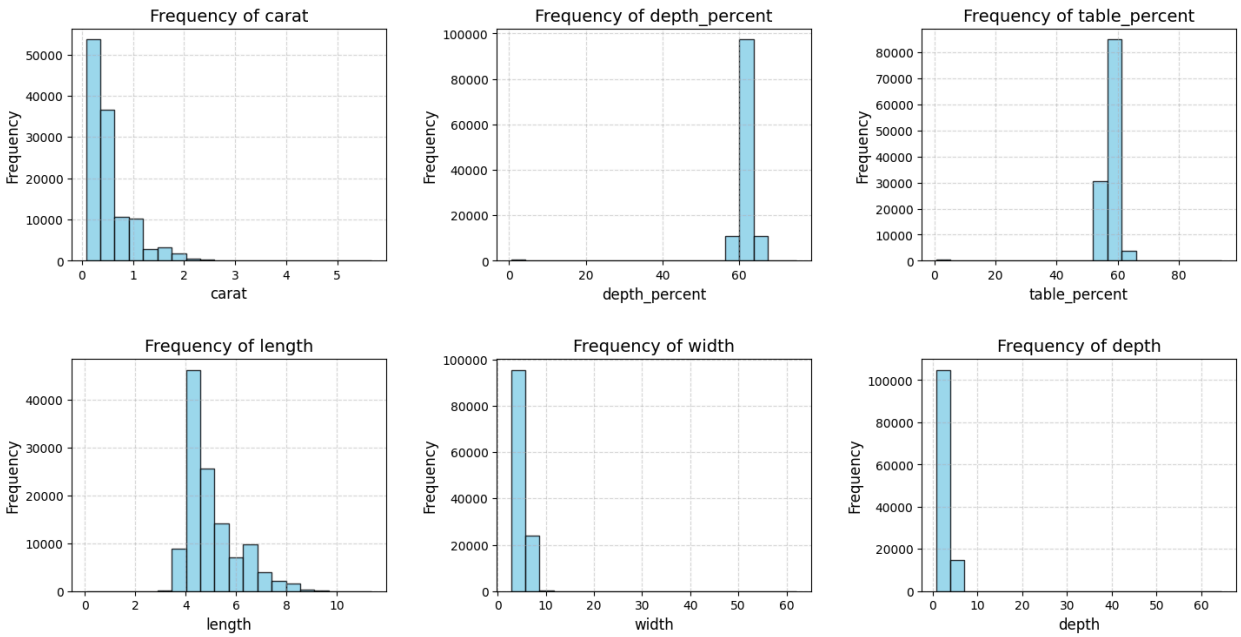


Figure 3: Histogram of Numerical Features

When encountering features with high skewness in their distribution, several preprocessing techniques can be employed to normalize the data. A log transformation can effectively reduce positive skewness. For more complex or severe skewness, a Box-Cox transformation could normalize the distribution. Alternatively, Winsorization can limit extreme values to reduce skewness impact. Standardizing the feature normalizes the distribution to a mean of zero and unit variance, though it doesn't inherently address skewness. Binning, or discretization, groups continuous feature values into intervals, which can also mitigate the influence of outliers. The choice of technique should be informed by the specific characteristics of the data and the objectives of the subsequent analysis.

As an example, here is the histogram for the ‘price’ feature after running a power transformation:

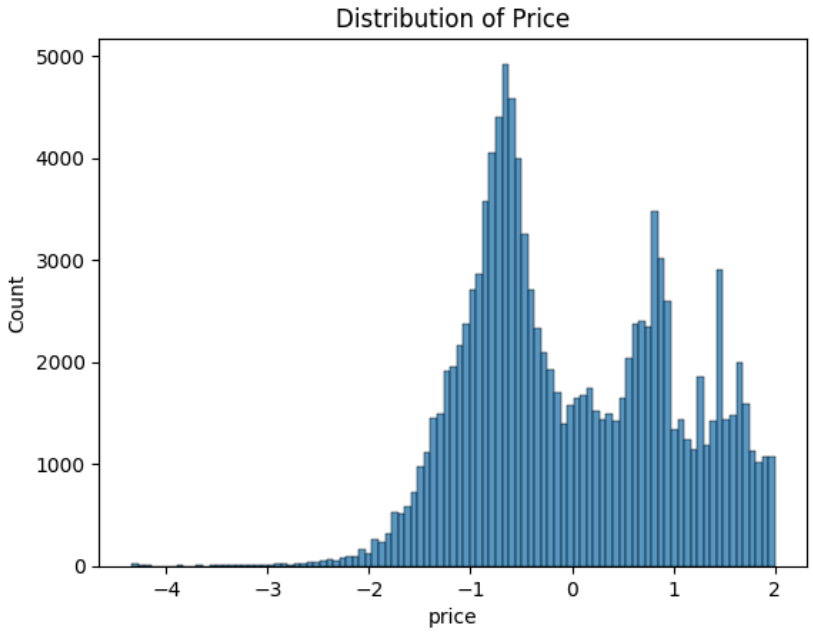
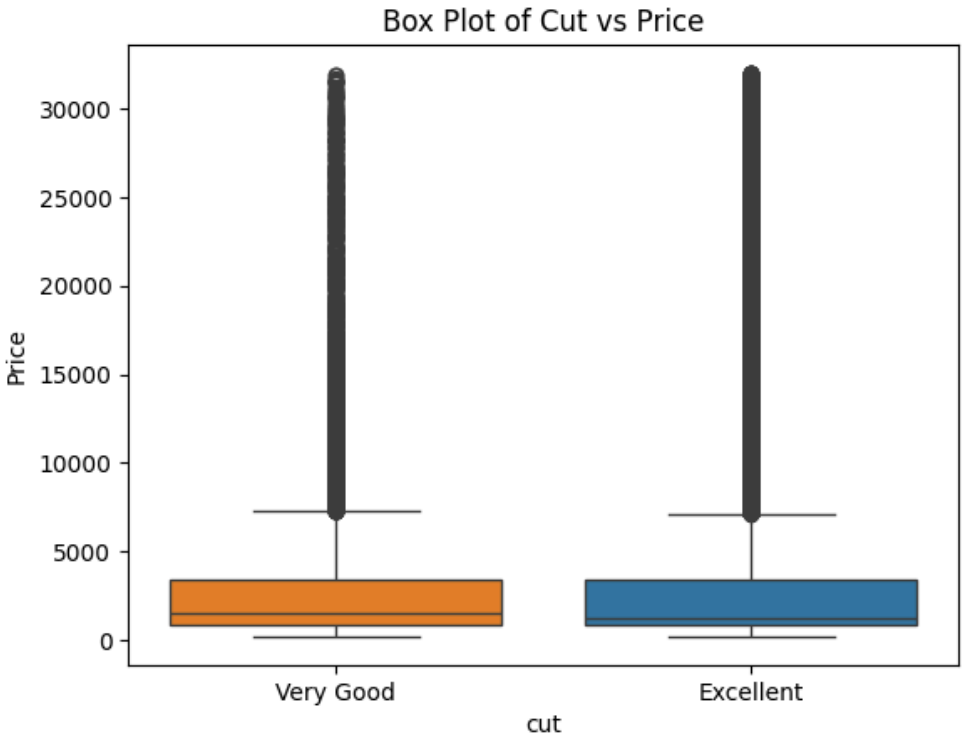


Figure 4: Histogram of Price

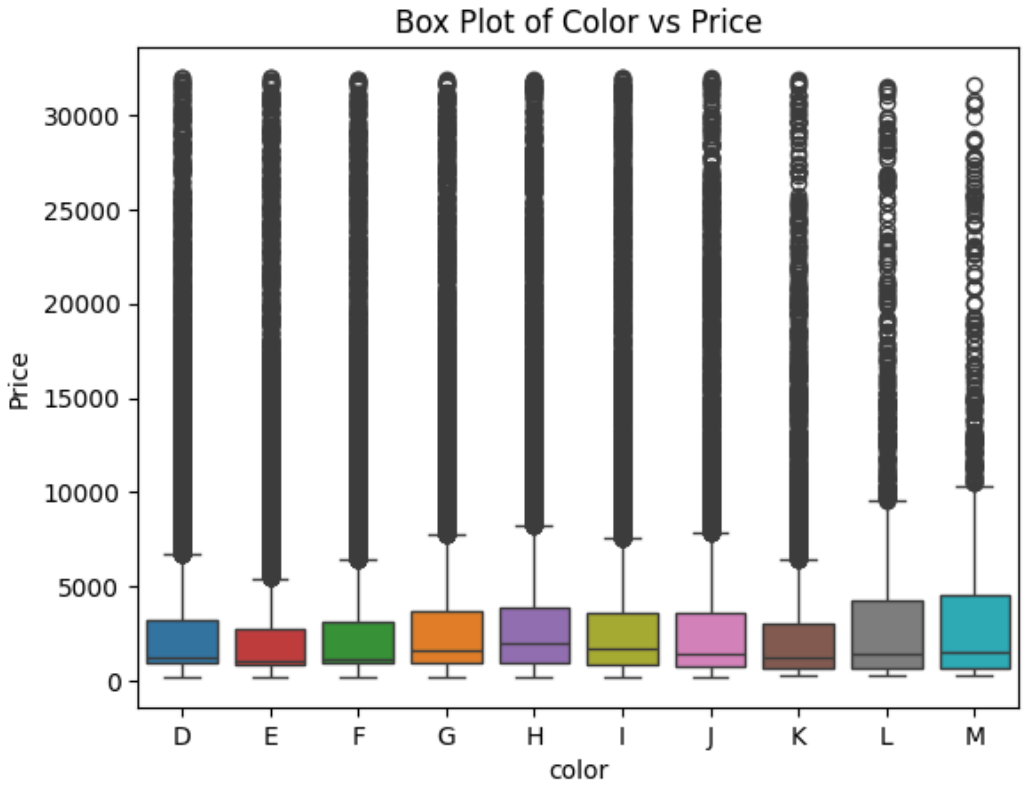
QUESTION 1.3 Construct and inspect the box plot of categorical features vs target variable. What do you find?

Figure 5(a)-(g): Categorical Features vs Target Variable Box Plots for the Diamond Dataset



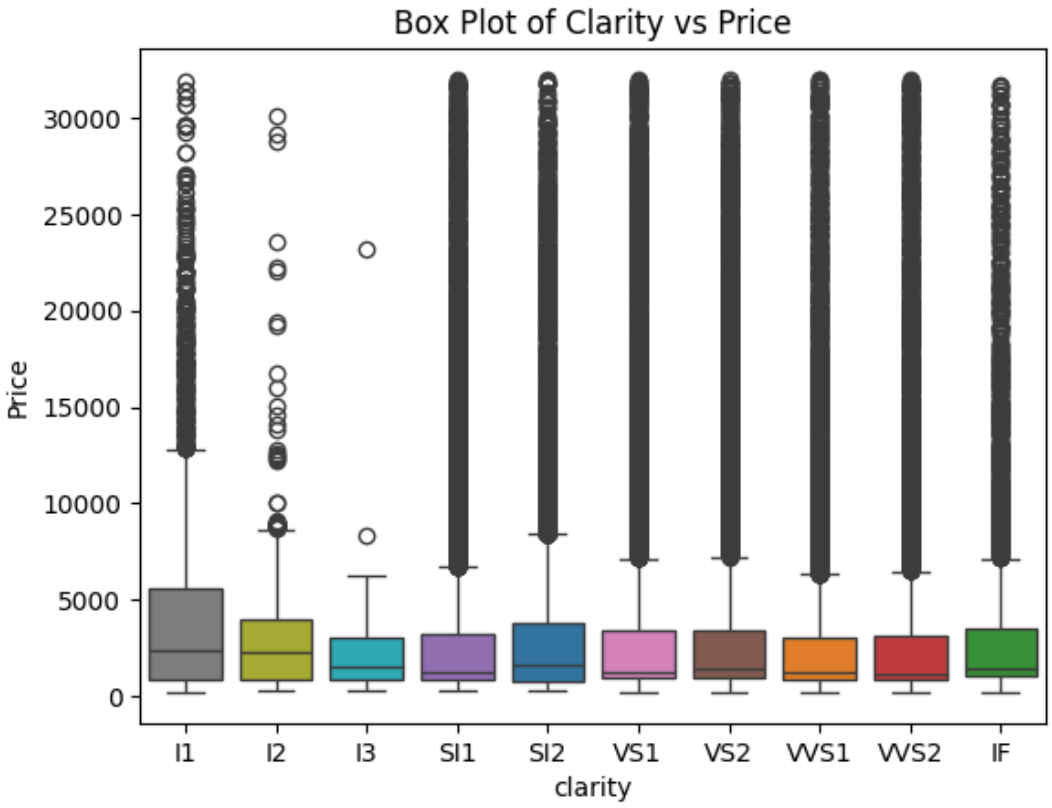
(a) Cut Qualities vs Price

The average diamond price remains constant between the 'Very Good' and 'Excellent' cut qualities. As these are the only two cut qualities of the diamonds present in the dataset, there is little variation regarding the interaction of the two variables. Because of this lack of diversity in diamond cuts, it is difficult to glean whether the cut quality has a significant effect on the price of a given diamond.



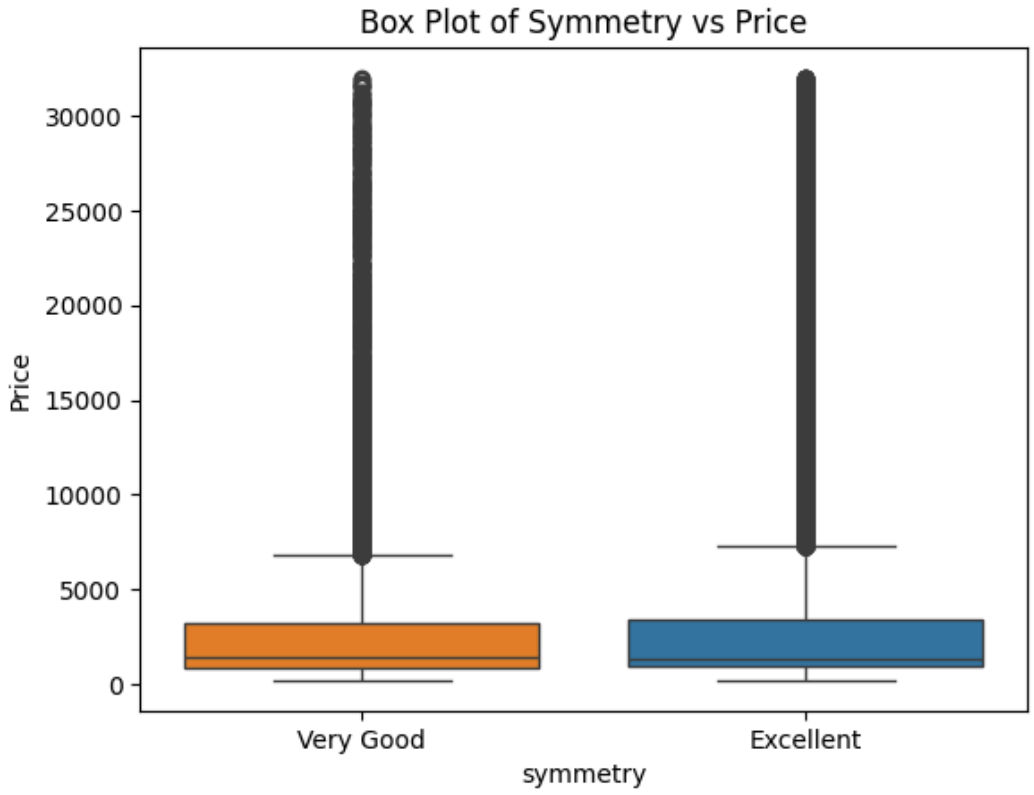
(b) Color Feature vs Price

The box plot shows variation in price distribution across different color grades, with some overlap. Higher quality colors (D, E, F) don't necessarily correspond to the highest prices, which suggests that other factors may be influencing price more significantly than color alone. There's a wide range of prices within each color grade, as evidenced by the vertical spread of the data points and the length of the "whiskers" in the plot.



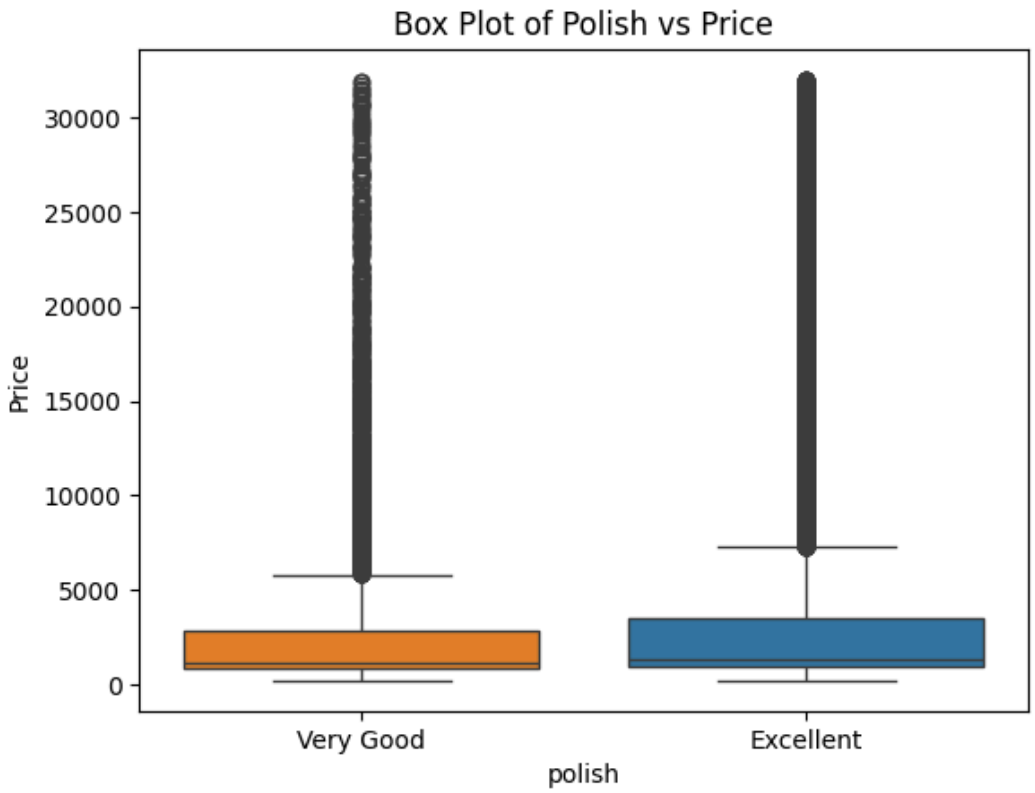
(c) Clarity Feature vs Price

There's a wide spread of prices across clarity grades, indicating that diamonds of all clarity levels can vary significantly in price. Interestingly, some lower clarity grades, such as SI1 and VS2, show a high range of prices, sometimes even higher than the better clarity grades like VVS1 and IF. This could indicate that clarity is not the only determining factor for the price and that diamonds with lower clarity might have other desirable attributes.



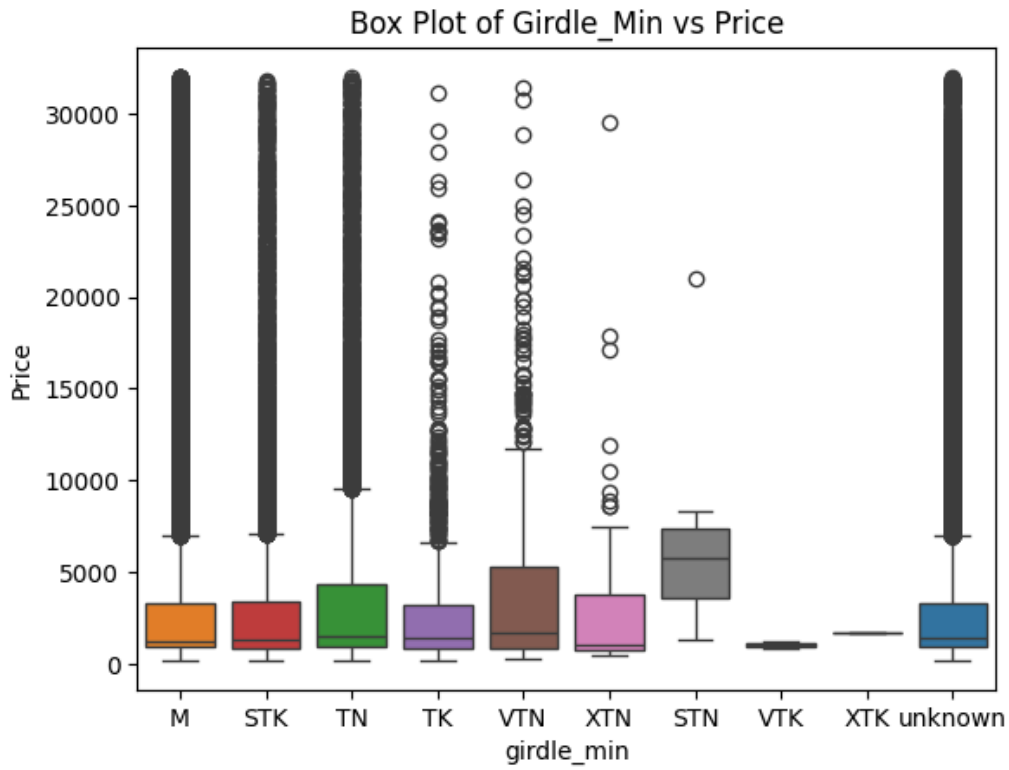
(d) Symmetry Feature vs Price

The box plot for symmetry grades shows that excellent symmetry tends to have a slightly higher median price than very good symmetry. However, the range of prices for both categories is extensive, with several outliers indicating that some diamonds with very good symmetry are priced higher than those with excellent symmetry.



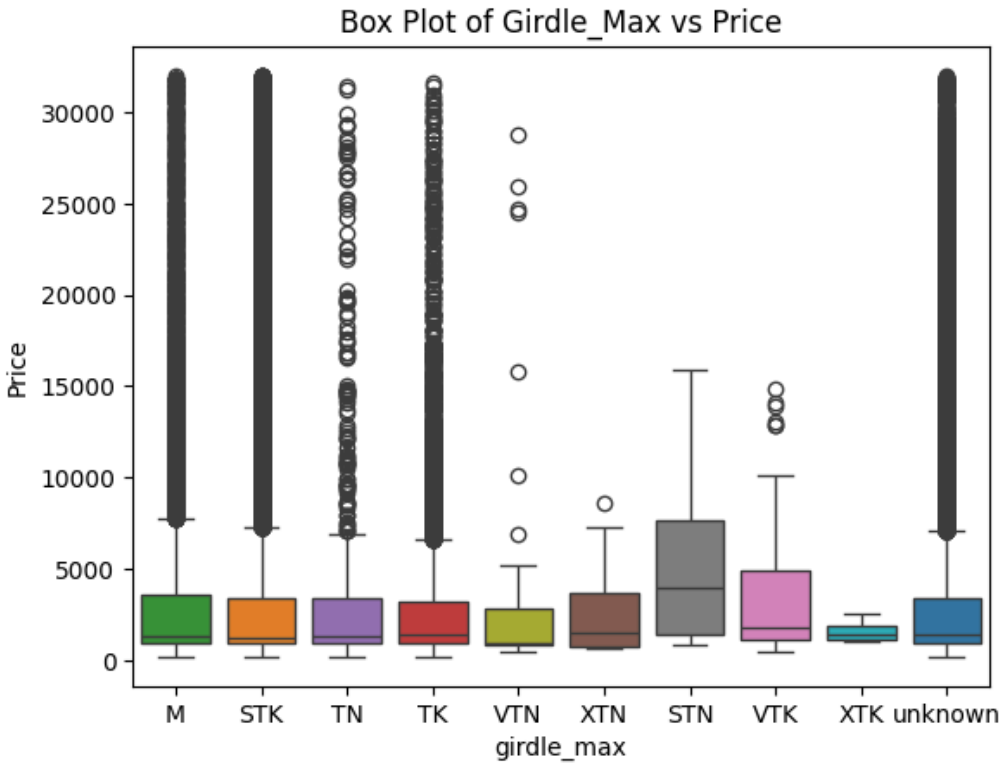
(e) Polish Feature vs Price

The box plot indicates that diamonds with excellent polish do not have a substantially different price range compared to those with very good polish, but the median price for excellent polish is slightly higher. This suggests that while polish may affect price, its impact is not as pronounced as other factors might be.



(f) Gridle_min vs Price

The box plot shows that the price distribution varies with different girdle thickness levels, but the relationship isn't linear or clear-cut. Some girdle thickness categories like STK and STN have a higher median price, but the category with the highest median price is unknown, suggesting missing or unclassified data in this feature.

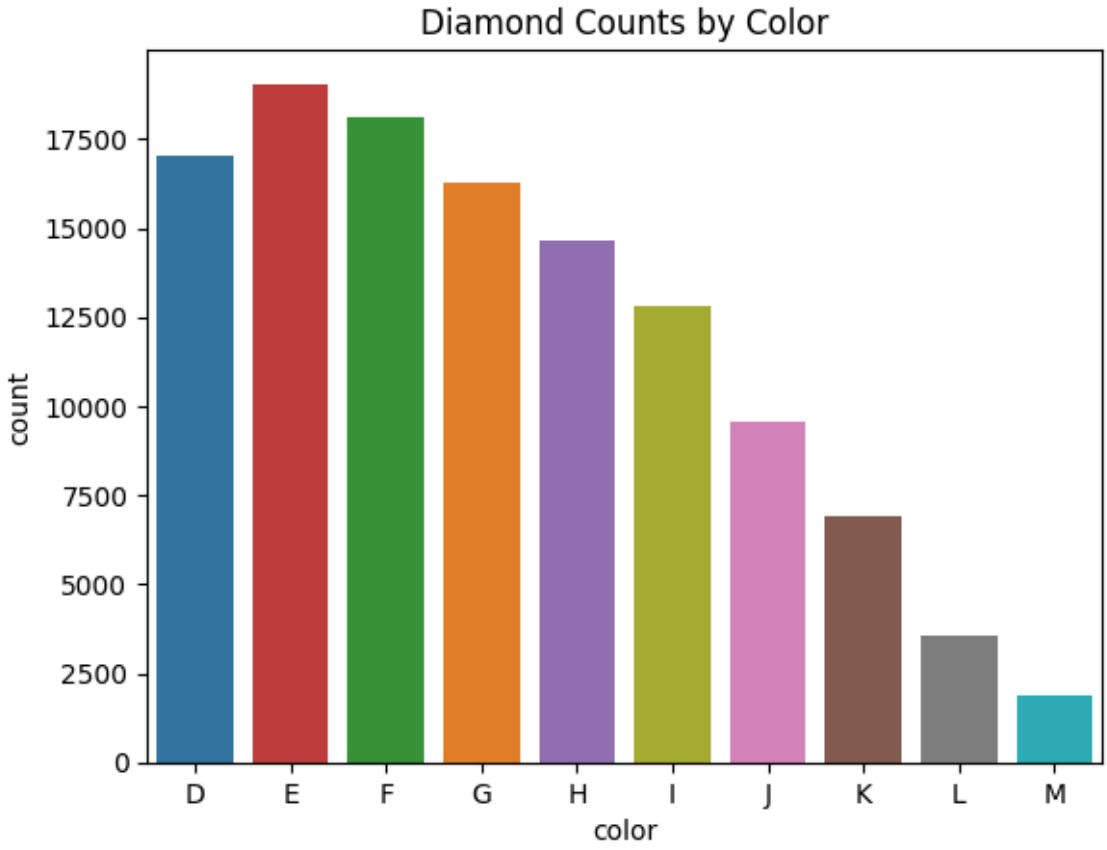


(g) Gridle_max vs Price

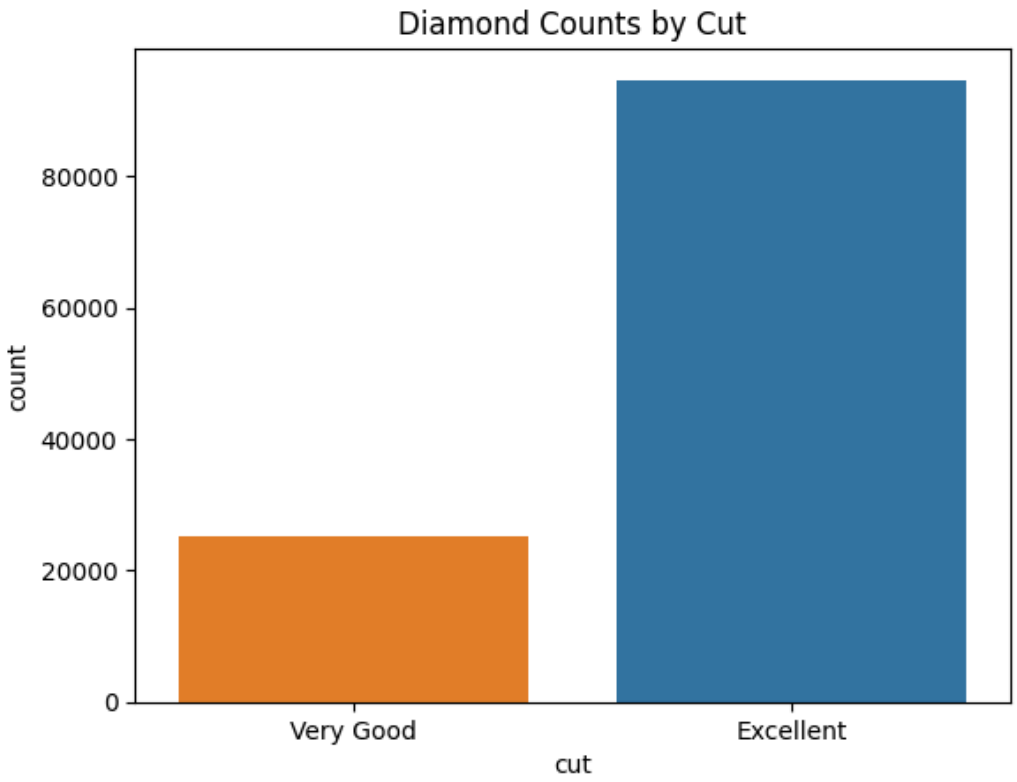
Similar to the girdle_min, girdle_max shows variation in price across different categories, but there's no clear trend in how girdle thickness relates to price. The "unknown" category also shows up with a significant spread in prices, possibly pointing to data categorization issues.

QUESTION 1.4 For the Diamonds dataset, plot the counts by color, cut and clarity

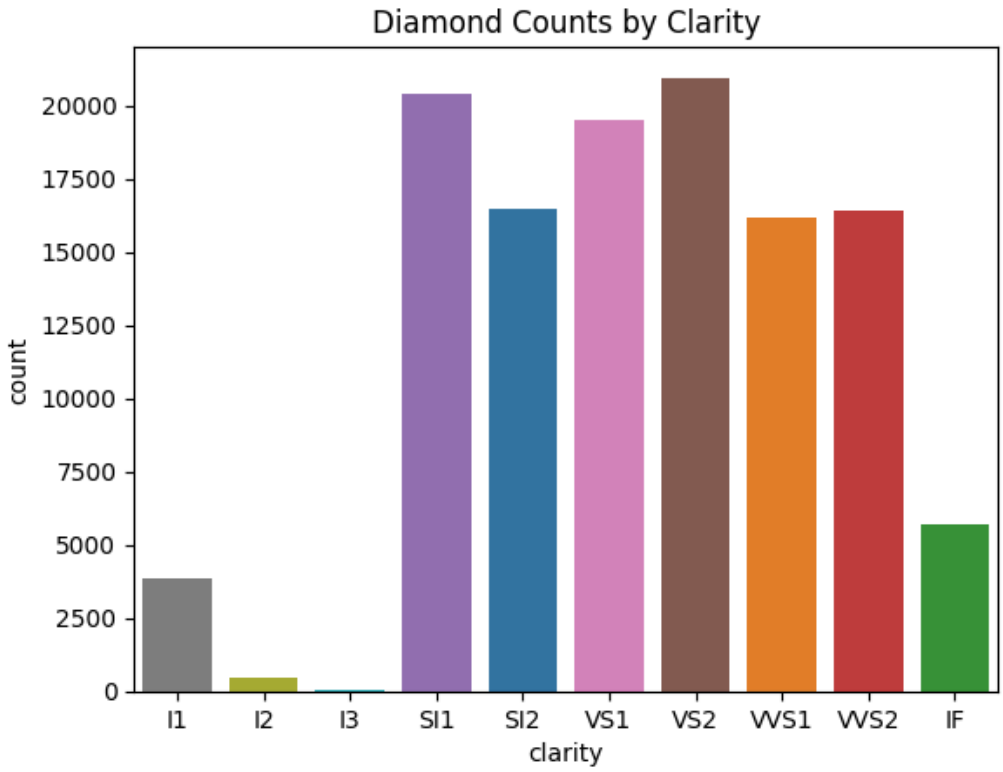
Figure 6(a)-(c): Histogram of Color, Cut and Clarity Features



(a): Histogram of Color



(b): Histogram of Cut



(c): Histogram of Clarity

QUESTION 2.1 Standardize feature columns and prepare them for training.

In preparation for training, the dataset underwent a process of standardization, which is crucial for many machine learning algorithms. These algorithms often assume features are on the same scale, typically with a mean of zero and a standard deviation of one. Without standardizing the features, models may not perform optimally as scale and variance discrepancies across features can unduly influence the model's learning process. Hence, we standardized the numerical feature columns of the diamond dataset to adhere to a normal distribution.

In addition to standardizing numerical features, we also encoded categorical variables such as cut, color, and clarity into numerical form to facilitate the training process. We devised an encoding scheme where ordinal and categorical variables were assigned numerical values corresponding to their qualitative significance or rank. For example, the 'cut' feature was encoded using a range of numbers where 'Very Good' was mapped to 1 and 'Excellent' to 2. Similarly, the 'color' feature was encoded from 'D' being 1 to 'M' as 10, and the 'clarity' feature from 'I1' as 1 to 'IF' as 10. Additional features like 'symmetry', 'polish', and 'girdle' measurements were encoded in a comparable manner.

The dataset's transformation into a standardized and encoded format is illustrated in the accompanying chart. The process converted character-based metrics for features such as cut, color, and clarity into numerical data, streamlining their integration into subsequent training algorithms.

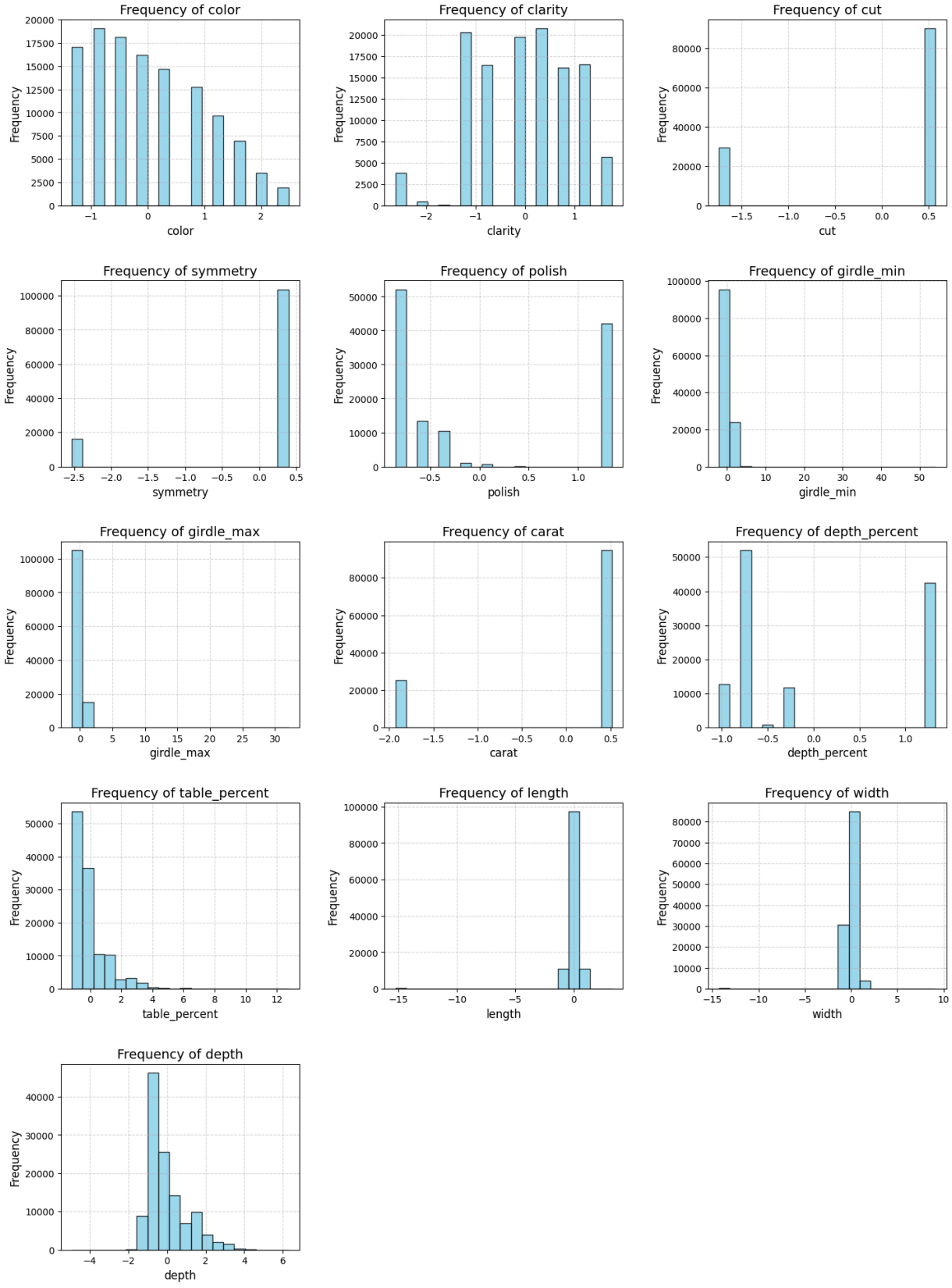


Figure 7: Frequency Plots for Different Features

Below is the resulting table after standardization. Each of the categorical features changed from characters to numerical data, enabling the training process to train using these features.

Table 1: Resulting Table of Different Features after Standardization

	color	clarity	carat	cut	symmetry	polish	depth_percent	table_percent	length	width	depth	girdle_min	girdle_max	price
0	-0.916097	1.267465	-1.157106	0.518390	-1.746964	-2.522184	0.215866	0.345119	-2.146391	-2.078247	-0.730430	-0.853299	-1.032312	-0.659094
1	-0.916097	1.267465	-1.157106	-1.929051	-1.746964	-2.522184	0.014689	0.345119	-2.156289	-2.059209	-0.735681	-0.609539	-0.770367	-0.659094
2	-0.916097	1.267465	-1.157106	0.518390	-1.746964	-2.522184	-0.186488	0.345119	-2.116697	-2.049690	-0.740932	-0.365779	-1.032312	-0.659094
3	-0.916097	1.267465	-1.157106	0.518390	-1.746964	-2.522184	0.039836	0.345119	-2.136493	-2.068728	-0.735681	-0.853299	-0.770367	-0.659094
4	-0.916097	1.267465	-1.157106	-1.929051	-1.746964	0.396482	0.769101	0.218693	-2.205778	-2.116324	-0.714676	-0.609539	-0.770367	-0.659094

QUESTION 2.2

- sklearn.feature selection.mutual info regression function returns estimated mutual information between each feature and the label. Mutual information (MI) between two random variables is a non-negative value which measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency.
- sklearn.feature selection.f regression function provides F scores, which is a way of comparing the significance of the improvement of a model, with respect to the addition of new variables.

You **may** use these functions to select features that yield better regression results (especially in the classical models). Describe how this step qualitatively affects the performance of your models in terms of test RMSE. Is it true for all model types? Also list two features for either dataset that has the lowest MI w.r.t to the target.

Table 2: Top 5 Features using Two-Feature Selection

Mutual Info Regression	carat	width	length	depth	color
F Regression	carat	length	width	depth	polish

Full results of mutual_info_regression:
['carat', 'width', 'length', 'depth', 'color', 'clarity', 'depth_percent', 'girdle_max', 'cut', 'girdle_min', 'symmetry', 'table_percent', 'polish']

Results of f_regression:
['carat', 'length', 'width', 'depth', 'polish', 'color', 'symmetry', 'table_percent', 'clarity', 'depth_percent', 'cut', 'girdle_max', 'girdle_min']

Incorporating feature selection into the model training process, particularly using techniques such as mutual information regression and F-score regression, can refine the model's focus on the most predictive features and thereby potentially improve the test RMSE. When non-informative or less significant features are filtered out, the model may become less complex, more efficient to train, and can generalize better to new data. This can lead to a more accurate prediction on the test set, as evidenced by a lower RMSE.

The enhancement in performance after feature selection, however, may vary across different types of models. Models that are more prone to overfitting or that rely on linear relationships, like classical linear regression, may benefit more distinctly from feature selection. Conversely, models that are inherently equipped with feature selection mechanisms, such as ensemble tree-based methods, may not show as dramatic an improvement in RMSE because these models are already designed to handle a large number of features and their interactions.

Regarding the diamond dataset, the results from mutual information regression and F-score regression have identified 'carat', 'width', 'length', 'depth', and 'polish' among the top five features in terms of both mutual information and F-score, indicating these features have a strong relationship with the price of diamonds. Conversely, features like 'symmetry', 'table_percent', and 'polish' are positioned lower on these lists, with 'polish' being the least significant by mutual information regression yet one of the top five by F-score. This discrepancy can arise due to the different nature of these two statistics: mutual information captures any kind of dependency while F-score is more specific to linear relationships.

Based on the provided results, 'cut' and 'girdle_min' are the features with the lowest mutual information with respect to the target in the diamond dataset. This suggests that they have the weakest dependencies with the price of diamonds, and their inclusion in a predictive model may not significantly influence the accuracy of price predictions. It's crucial to consider the model's context and the specific data when deciding whether to include or exclude these features, as even weak relationships can sometimes provide valuable predictive signals when combined with other features.

Table 3: Todo

	carat	width	length	depth	color
0	-1.157106	-2.078247	-2.146391	-0.730430	-0.916097
1	-1.157106	-2.059209	-2.156289	-0.735681	-0.916097
2	-1.157106	-2.049690	-2.116697	-0.740932	-0.916097
3	-1.157106	-2.068728	-2.136493	-0.735681	-0.916097
4	-1.157106	-2.116324	-2.205778	-0.714676	-0.916097

QUESTION 4.1 Explain how each regularization scheme affects the learned parameter set.

The original objective function can be written as: $\text{minimize} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$, and regularization is an additional term added to the function.

1. Ordinary Least Squares (linear regression without regularization):

   The model aims to minimize the sum of squared errors without any additional constraints on the parameter values. It tends to produce parameter estimates that are purely based on minimizing the training error, potentially leading to overfitting when the number of features is large relative to the number of observations.

2. Lasso (L1 Regularization):

   In Lasso regression, an L1 penalty term is added to the objective function, which is the absolute sum of the coefficients multiplied by a regularization parameter λ. The objective function becomes:

   $$\text{minimize} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

   where β_j is the regression coefficient and p is the number of features. Lasso regularization tends to shrink some coefficients towards zero, effectively performing feature selection by setting some coefficients exactly to zero. This property makes Lasso useful for sparse feature selection and can provide more interpretable models.

3. Ridge (L2 Regression):

   Similar to Lasso, λ is the regularization parameter, and β_j^2 represents the squared coefficients.

   $$\text{minimize} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

   Ridge regularization also penalizes large coefficients but does not force them to become exactly zero. It tends to shrink the coefficients towards zero, but retains all features and often performs better than Lasso when the dataset contains highly correlated features.

In summary, while OLS aims to minimize the sum of squared errors directly, Lasso and Ridge regression add regularization terms to prevent overfitting by penalizing large coefficients. Lasso encourages sparsity by forcing some coefficients to be zero, whereas Ridge tends to shrink coefficients towards zero while keeping all features.

QUESTION 4.2 Report your choice of the best regularization scheme along with the optimal penalty parameter and explain how you computed it.

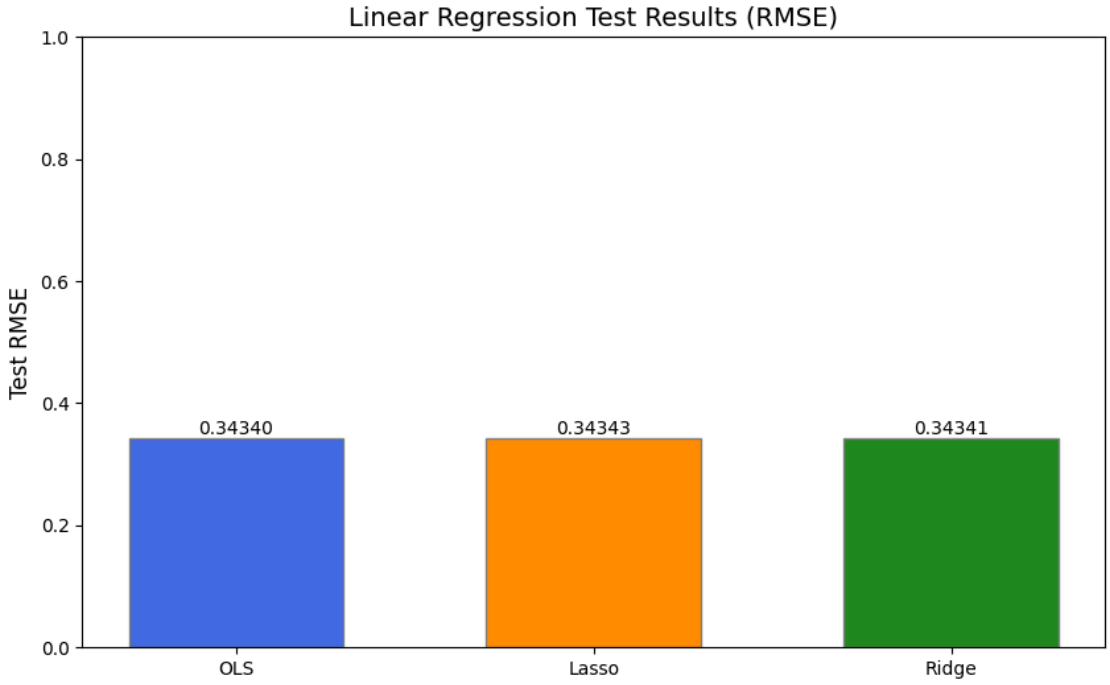
Feature selection is a process to select the top k features based on two different methods (mutual information and f-regression). This would help in reducing the complexity of the model and possibly improving the performance by only keeping the most relevant features. Each model is trained on the selected features, and the RMSE is computed using 10-fold cross-validation. Cross-validation helps in estimating the model's performance on an independent dataset and reduces the variance of the model assessment. For Ridge and Lasso, different alpha values (the regularization penalty parameter) would be tried to find the optimal setting that minimizes the validation RMSE. This part is not explicitly shown in the code but is mentioned in the results. After evaluating different models and their parameter settings, they are ranked based on their average validation RMSE scores. The one with the lowest RMSE is considered the best.

Test Results:
Best regularization model is Lasso with alpha = 0.00001
with average validation RMSE score = 0.3448231637488446

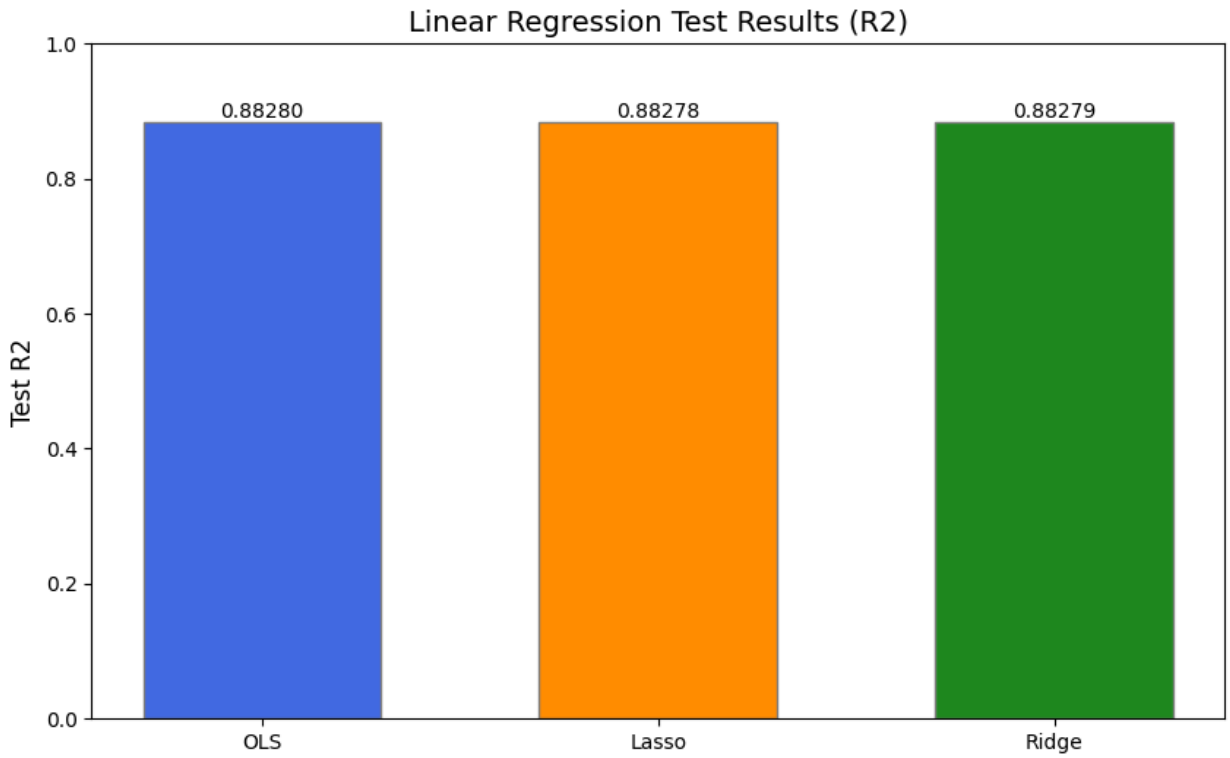
Best Ridge model has alpha = 0.000001 and ranked 4th
with average validation RMSE score = 0.3448231770743167

The Ordinary Least Square model is ranked 3rd
with average validation RMSE score = 0.34482317707430865

Figure 8: Linear Regression Test Results in Different Metrics



(a) Metrics RMSE



(b) Metrics R2

Table 4: Full Test RMSE and Test R^2 Values for Each Model

Model	Test RMSE	Test R^2
OLS	0.3433995726576627	0.8828015626784848
Lasso	0.34342589553076913	0.8827835945848821
Ridge	0.34341379439744146	0.882791855035698

Regularization helps mitigate overfitting in predictive models by imposing a penalty on the size of the coefficients. This penalty, integrated into the loss function, curtails the tendency of the model to overly rely on individual predictors. Two common types of regularization are Lasso and Ridge. Lasso introduces a penalty equivalent to the absolute values of the coefficients, promoting sparsity by driving some coefficients to zero. Ridge, on the other hand, penalizes the sum of the squared coefficients, thus constraining them but not necessarily reducing them to zero.

To assess the most effective regularization approach and its associated parameter – the alpha – we implemented a grid search strategy using scikit-learn's GridSearchCV. This method systematically works

through a predefined assortment of alpha values, assessing the model's performance at each point via cross-validation. The particular alpha value that yields the lowest validation error is then chosen as the optimal parameter.

The hyperparameter search included both Ridge and Lasso models, with alphas ranging logarithmically from 1e-6 to 1e6. The Pipeline object in scikit-learn streamlined the process, encapsulating the model fitting within a singular framework. We specified the models and corresponding alphas in the params list, and initiated the grid search with GridSearchCV, passing in a 10-fold cross-validation configured to shuffle the data, thereby enhancing the validity of the validation process. The grid search was executed with $n\_jobs = -1$ to utilize all available CPU cores for parallel processing, speeding up the computation.

Upon fitting the grid search with our dataset, we extracted the best performing model and alpha value. The selection was based on the model's negative root mean squared error, with a lower value indicating better predictive performance. This approach ensured we identified the best regularization scheme and the alpha that minimized predictive errors, improving our model's generalizability.

```
default_pipeline = Pipeline([
    ('model',LinearRegression())
    ])

params = [
     {
          "model":[LinearRegression()]
     },
     {
          "model":[Ridge()],

"model__alpha":[1e-6,1e-5,1e-4,1e-3,1e-2,1e-1,1,1e1,1e2,1e3,1e4,1e5,1e6]
     },
     {
          "model":[Lasso()],

"model__alpha":[1e-6,1e-5,1e-4,1e-3,1e-2,1e-1,1,1e1,1e2,1e3,1e4,1e5,1e6]
     }
]

g_s = GridSearchCV(default_pipeline, params,cv=KFold(n_splits=10,shuffle =
True, random_state = 42),n_jobs=-1,
scoring=('neg_root_mean_squared_error'),verbose=10,
                    return_train_score=False)
g_s.fit(diamond_standard_x,diamond_standard_y)

df_g_s = pd.DataFrame(g_s.cv_results_)
sorted_g_s = df_g_s.sort_values(by='rank_test_score', ascending=True)
```

```
sorted_g_s[:20]
```

QUESTION 4.3 Does feature standardization play a role in improving the model performance (in the cases with ridge regularization)? Justify your answer.

Feature standardization, which typically involves subtracting the mean and scaling to unit variance (Z-score normalization), plays a significant role in improving model performance, particularly in the context of models that utilize regularization, such as Ridge regression.

Ridge regression applies a penalty to the coefficients based on their squared values, so all features should be on a similar scale for this penalty to be applied uniformly. If features are on different scales, the regularization term will not penalize large coefficients of small-scale features as effectively as it should, which can lead to a biased model that does not generalize well. Standardization ensures that the regularization term applies equally to all features, improving the model's ability to learn from the data without being influenced unduly by the scale of the input variables.

Without standardization, features with larger scales will dominate the penalty term, skewing the model's attention towards those variables, possibly at the expense of other important but smaller-scale features. When all features are standardized, the Ridge model is more balanced and can more effectively shrink coefficients to reduce overfitting, leading to better performance on new, unseen data.

In practice, standardizing features before applying Ridge regression is a recommended preprocessing step. Empirically, this can be observed by comparing the performance of models with and without feature standardization and noting the improvements in error metrics such as RMSE, as well as in terms of the stability and interpretability of the model coefficients.

QUESTION 4.4 Some linear regression packages return p-values for different features2 . What is the meaning of these p-values and how can you infer the most significant features? A qualitative reasoning is sufficient.

What is the meaning of these p-values?

The p-values associated with different features indicate the probability of observing the given coefficient value under the null hypothesis that the true coefficient is zero (i.e., the feature has no effect on the response variable). When a linear regression model is fitted, each coefficient represents the change in the response variable for a one-unit change in the corresponding feature, holding other features constant. The p-value for each coefficient tests the null hypothesis that the coefficient is equal to zero (no effect) versus the alternative hypothesis that the coefficient is not equal to zero (some effect). A low p-value (< 0.05) indicates that the null hypothesis can be rejected and that there is a statistically significant association between the feature and the response variable.

How can you infer the most significant features?

Lower p-values suggest stronger evidence against the null hypothesis and indicate that the corresponding feature is more likely to be significant in predicting the response variable. Therefore, features with lower p-values are considered more significant in the model.

QUESTION 5.1 What are the most salient features? Why?

Top Six Salient Features:

1. Carat
2. Width
3. Length
4. Color
5. Polish
6. Depth

Best parameters for PolynomialFeatures: {'poly_transform__degree': 2}

Best score for PolynomialFeatures: -0.31559236079123537

The top six salient features – Carat, Width, Length, Color, Polish, and Depth – are the ones that have shown the most significant and consistent relationships with the target variable ‘price’ in both the initial feature selection and the post-transformation Ridge regression.

To fully assess the six most salient features, it is important to take into account the real-world significance of each feature. As the weight of the diamond increases, so does its price, typically in a non-linear fashion. The width and length of a diamond both affect its perceived size and brilliance. The color grade of a diamond is a key factor in its valuation, with clearer diamonds generally being more valuable. The quality of a diamond's polish affects its light performance and can thus influence its price. Finally, the depth of a diamond relates to its proportions and can affect its sparkle.

The combination of SelectKBest and the Ridge regression's coefficient analysis provided a robust method for determining the most salient features. The polynomial transformation's degree of 2 that resulted in the best score further suggests that the relationships between these features and the diamond's value are not purely linear, but include quadratic terms which capture more complex interactions between the features.

QUESTION 5.2 What degree of polynomial is best? How did you find the optimal degree? What does a very high-order polynomial imply about the fit on the training data? What about its performance on testing data?

What degree of polynomial is best?
The best degree of polynomial is 2, as shown in 'poly_transform__degree'.

How did you find the optimal degree?
We find the optimal degree of polynomial by cross-validation. We test various degrees of polynomial, and for each degree we evaluate the model using cross-validation. Then we can find the best one.

What does a very high-order polynomial imply about the fit on the training data?
A polynomial of very high order (e.g. 6 or more) implies that this is an increasingly complex model. Thus the model may perfectly fit on the training data, but in the meantime it risks the overfitting problem and may perform badly on the testing data.

What about its performance on testing data?
A very high-order polynomial may overfit and has poor performance on testing data, despite its exceptionally good performance on the training data.

QUESTION 6.1 Adjust your network size (number of hidden neurons and depth), and weight decay as regularization. Find a good hyper-parameter set systematically (no more than 20 experiments in total).

```
mlpr = Pipeline([
    ('model', MLPRegressor(max_iter=1000, early_stopping=True)),
], memory=memory)

param_list = {
    "model__hidden_layer_sizes": [(30, 40), (40, 60)],
    "model__activation": ["relu"],
    "model__solver": ["adam"],
    "model__alpha": [0.0001, 0.001, 0.01],
}
```

We used the GridSearchCV function from the sklearn.model_selection library, fitting 3 folds for each of 6 candidates, totalling 18 fits (below 20 total experiments).
Best parameters for Neural Network:
Model Activation: 'relu',
Model Alpha: 0.01,
Model Hidden Layer Sizes: (40, 60),
Model Solver: 'adam'
Best score = -0.6205043512139231
Original RMSE = 0.7877209856376324

QUESTION 6.2 How does the performance generally compare with linear regression? Why?

The performance of MLP generally outperforms linear regression.

This is because MLP models can deal with complex and non-linear relationships better than linear regression models. MLPs have the capability to model intricate non-linear relationships through their multiple layers of neurons and activation functions. In contrast, linear regression models simply assume a linear relationship between the input features and the target, and thus are inclined to fail in non-linear and complex cases.

QUESTION 6.3 What activation function did you use for the output and why? You may use none.

We used the `relu` activation function.

We use `relu` because it is computationally efficient, and has shown to perform well in lots of previous works. `relu` can also help to promote sparse activations and to largely avoid the vanishing gradient problem, which are beneficial for our training process.

QUESTION 6.4 What is the risk of increasing the depth of the network too far?

Increasing the depth of the network too far increases the risk of overfitting. This is because when the network depth is too large, the model will memorize the training data instead of learning generalizable patterns, leading to poor performance on unseen data and causing overfitting.

QUESTION 7.1 Random forests have the following hyper-parameters:

– Maximum number of features;
– Number of trees;
– Depth of each tree;

Explain how these hyper-parameters affect the overall performance. Describe if and how each hyper-parameter results in a regularization effect during training.

1. Maximum number of features:

It controls the complexity of individual trees, promoting diversity and preventing overfitting by limiting the number of features considered for splitting at each node.

Thus, it acts as a form of regularization by discouraging the model from memorizing noise in the data and promoting better generalization to unseen examples.

2. Number of trees:

It increases model robustness and generalization by aggregating predictions from multiple trees, reducing variance and stabilizing performance.

Thus, a higher number of trees can act as a regularization technique by stabilizing the model's predictions and improving its ability to generalize.

3. Depth of each tree:

It controls the complexity of decision boundaries, preventing overfitting by limiting tree depth and encouraging better generalization to unseen data.

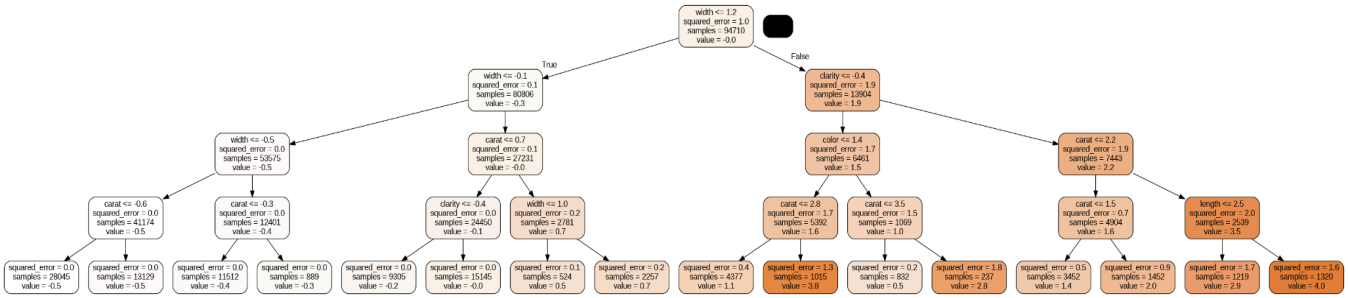
Thus, limiting the depth of each tree acts as a regularization, ensuring that the model focuses on learning the most salient patterns rather than memorizing noise in the training data.

QUESTION 7.2 How do random forests create a highly non-linear decision boundary despite the fact that all we do at each layer is apply a threshold on a feature?

Random forests create such a decision boundary due to their ensemble nature of multiple decision trees. Each tree learns different aspects of the data, capturing various features and relationships. When combined, the diverse predictions of these trees form complex, non-linear decision boundaries. This allows random forests to effectively model intricate patterns in the data.

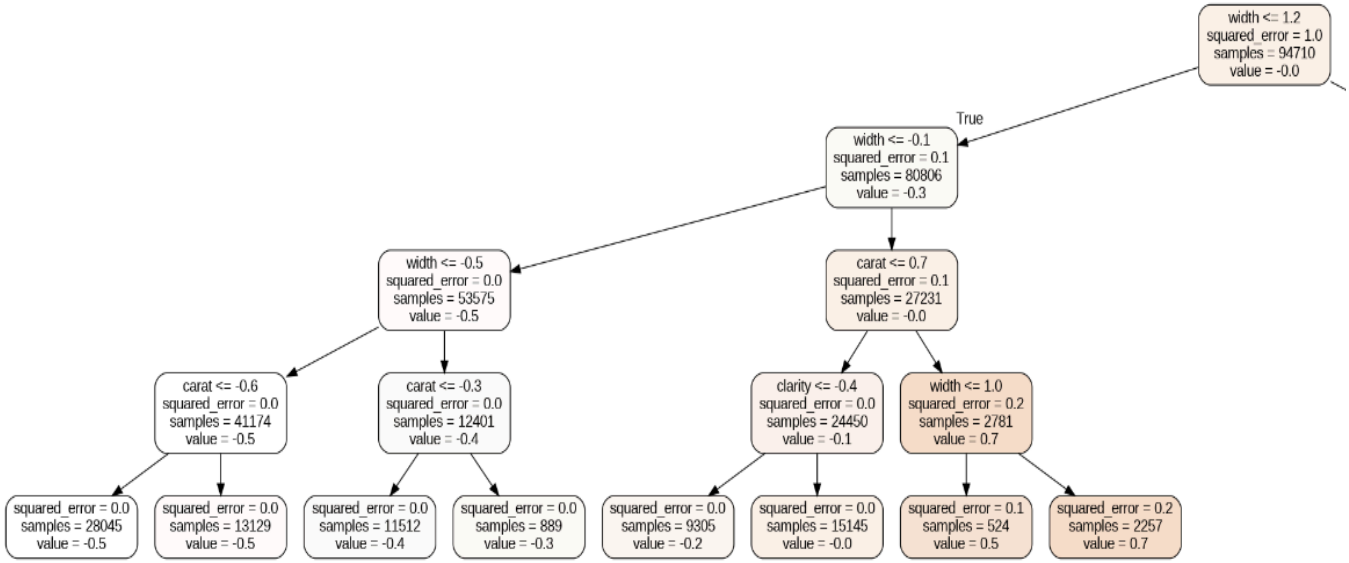
QUESTION 7.3 Randomly pick a tree in your random forest model (with maximum depth of 4) and plot its structure. Which feature is selected for branching at the root node? What can you infer about the importance of this feature as opposed to others? Do the important features correspond to what you got in part 3.3.1?

Figure 9: Plot of the structure of the randomly-selected tree in our random forest model

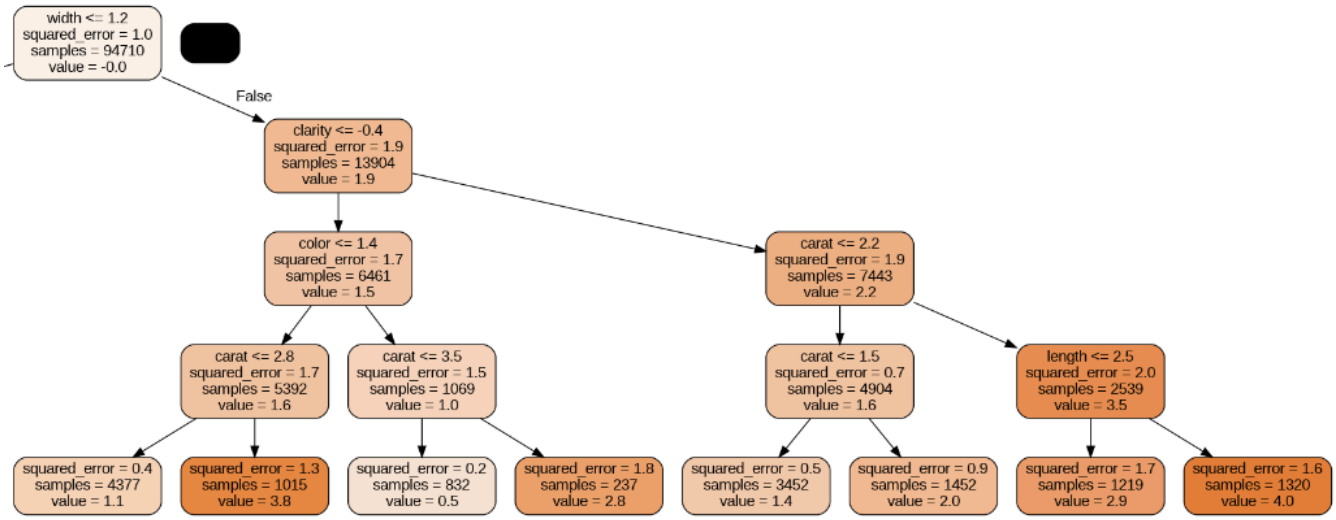


Because our plot is too wide to be legible in the letterbox format of this report, we have magnified the left and right branches of the tree plot from the root node below. This plot is the same as above.

(a) Left side of tree plot:



(b) Right side of tree plot:



Which feature is selected for branching at the root node?

The feature selected for branching at the root node is "width <= 0.5".

What can you infer about the importance of this feature as opposed to others?

The selection of "width" at the root node suggests that this feature has significant predictive power for the model. The root node is chosen because it provides the greatest reduction in variance for the target variable. Hence, width plays a crucial role in explaining the variability of price.

Do the important features correspond to what you got in part 3.3.1?

From the top 6 salient features computed in Question 5.1, "width" resulted as the second most salient feature, corroborating its importance. The fact that it appears as the root node feature in this tree of the random forest suggests that it is consistently significant across different models and techniques for feature selection and importance evaluation. Color (the fourth most salient feature) and Carat (the most salient feature) are also high on the chart, close to the root node. Thus, the important features in this plot do correspond to the results in part 3.3.1.

QUESTION 7.4 Measure "Out-of-Bag Error" (OOB). Explain what OOB error and R2 score means.

Table 5: Best Random Forest Model for Diamond Dataset

OOB score	0.9022
R^2 score	0.9034

"Out-of-Bag Error" (OOB) estimates model performance using data not included in bootstrap samples:

$$\text{OOB Error} = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i^{OOB})$$

Here n is the number of samples, y_i is the true label for sample i, and \hat{y_i^OOB} is the prediction made by the tree trained without including sample i. L denotes the loss function, such as mean squared error for regression.

R2 score measures how well the model fits the training data, with higher values indicating better fit.

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

Here SS_res is the residual sum of squares and SS_tot is the total sum of squares.

QUESTION 8.1 Read the documentation of LightGBM OR CatBoost and determine the important hyperparameters along with a search space for the tuning of these parameters (keep the search space small).

For LightGBM:

1. num_leaves: larger num_leaves value increases the model's capacity to learn, but also risks overfitting; Search space: [20, 100]
2. learning_rate: lower learning_rate generally require more iterations to converge; higher learning_rate may overfit; Search space: [0.01, 0.1]
3. max_depth: larger max_depth can lead to more complex models, but also risks overfitting; Search space: [3, 7]
4. min_child_samples: higher min_child_samples prevent the model from overfitting by ensuring that each leaf node represents a sufficient number of samples; Search space: [10, 30]
5. subsample: a value less than 1.0 introduces randomness into the training process and helps prevent overfitting; Search space: [0.6, 0.8]
6. colsample_bytree: a value less than 1.0 introduces randomness and helps prevent overfitting; Search space: [0.6, 0.8]
7. reg_alpha: larger value makes the model more conservative by penalizing large weights; Search space: [0.0, 0.5]
8. reg_lambda: larger value makes the model more conservative by penalizing large weights; Search space: [0.0, 0.5]

Here is the implementation of the 8 listed hyperparameters with associated search spaces:

```
param_lgb = {
    'model__num_leaves': Integer(20, 100),
    'model__learning_rate': Real(0.01, 0.1, prior='log-uniform'),
    'model__max_depth': Integer(3, 7),
    'model__min_child_samples': Integer(10, 30),
    'model__subsample': Real(0.6, 0.8),
    'model__colsample_bytree': Real(0.6, 0.8),
    'model__reg_alpha': Real(0.0, 0.5),
    'model__reg_lambda': Real(0.0, 0.5)
}
```

QUESTION 8.2 Apply Bayesian optimization using skopt.BayesSearchCV from scikit-optmize to find the ideal hyperparameter combination in your search space. Keep your search space small enough to finish running on a single Google Colab instance within 60 minutes. Report the best hyperparameter set found and the corresponding RMSE.

Fitting 10 folds for each of 1 candidates, totalling 10 fits

Best hyperparameter set:

- Number of leaves: 84
- Learning Rate: 0.087
- Maximum depth: 4
- Minimum Child Samples: 14
- Subsample: 0.707
- Column Sample By Tree: 0.747
- L1 Regularization (Alpha): 0.186
- L2 Regularization (Lambda): 0.230

Best score = -0.32106710899626095
RMSE = 0.2497

Full output of the best parameters for LightGBM:

```
[
        ('model__num_leaves', 84),
        ('model__learning_rate', 0.08697005357407943),
        ('model__max_depth', 4),
        ('model__min_child_samples', 14),
        ('model__subsample', 0.7069530320242271),
        ('model__colsample_bytree', 0.7468056035971137),
        ('model__reg_alpha', 0.18628988899162893),
        ('model__reg_lambda', 0.22951225707540285)
]
```

QUESTION 8.3 Qualitatively interpret the effect of the hyperparameters using the Bayesian optimization results: Which of them helps with performance? Which helps with regularization (shrinks the generalization gap)? Which affects the fitting efficiency?

Hyperparameters that helps with performance:

1. Larger 'model__num_leaves' allows the model to capture finer patterns and thus may improve performance.
2. Smaller 'model__learning_rate' allow for finer adjustments during training and thus may improve performance.
3. Larger 'model__max_depth' means deeper trees and can thus capture more complex patterns.
4. Larger 'model__min_child_samples' prevent splitting nodes with too few samples and thus may reduce overfitting and improve performance.
5. Adding 'model__subsample' introduces randomness, and thus prevents overfitting and thus improves performance.
6. Adding 'model__colsample_bytree' selects features for each tree, and thus prevents overfitting and thus improves performance.
7. Adding 'model__reg_alpha' and 'model__reg_lambda' penalize large coefficients and improve performance.

Hyperparameters that helps with regularization (shrinks the generalization gap):

1. Limiting 'model__num_leaves' can control tree complexity and prevent overfitting.
2. Smaller 'model__learning_rate' can act as regularization by slowing down the learning process and preventing overfitting.
3. Limiting 'model__max_depth' can control tree complexity and prevent overfitting.
4. Using 'model__min_child_samples' can act as regularization by preventing small leaf nodes.
5. Adding 'model__subsample' can act as regularization by reducing dependence on specific samples.
6. Adding 'model__colsample_bytree' can act as regularization by reducing dependence on specific features.
7. Using 'model__reg_alpha' and 'model__reg_lambda' means using L1 and L2 regularization, which avoids overfitting by adding additional penalty terms.

Hyperparameters that affects the fitting efficiency:

1. Larger 'model__num_leaves' will increase the computational resources and training time needed, and thus may potentially decrease fitting efficiency.
2. Larger 'model__learning_rate' leads to longer time to converge and decreases the fitting efficiency.
3. Larger 'model__max_depth' means deeper trees and more computational resources as well as training time needed, and thus may potentially decrease fitting efficiency.
4. Larger 'model__min_child_samples' may lead to more efficient training and fewer splits, thus potentially reducing computational overhead and increasing fitting efficiency.
5. Larger 'model__subsample' may lead to more efficient training as fewer samples are used for each tree, thus potentially reducing computational overhead and increasing fitting efficiency.
6. Adding 'model__colsample_bytree' may lead to smaller fractions of features used for each tree, and may improve efficiency by reducing the complexity of individual trees and the computational cost associated with feature selection.
7. Using 'model__reg_alpha' and 'model__reg_lambda', or using L1 and L2 regularization, may slightly increase the computational overhead and potentially affect the overall fitting efficiency, but only slightly.

QUESTION 9.1 Download the training tweet data3 . The data consists of 6 text files, each one containing tweet data from one hashtag as indicated in the filenames. Report the following statistics for each hashtag, i.e. each file has:

- Average number of tweets per hour
- Average number of followers of users posting the tweets per tweet (to make it simple, we average over the number of tweets; if a users posted twice, we count the user and the user's followers twice as well)
- Average number of retweets per tweet

Statistics for *tweets_#gohawks.txt*:

Avg. Number of Tweets Per Hour	292.48785062173687
Avg. Number of Followers of Users Posting the Tweets Per Tweet	2217.9237355281984
Avg. Number of Retweets Per Tweet	2.0132093991319877

Statistics for *tweets_#gopatriots.txt*:

Avg. Number of Tweets Per Hour	40.954698006061946
Avg. Number of Followers of Users Posting the Tweets Per Tweet	1427.2526051635405
Avg. Number of Retweets Per Tweet	1.4081919101697078

Statistics for *tweets_#nfl.txt*:

Avg. Number of Tweets Per Hour	397.0213901819841
Avg. Number of Followers of Users Posting the Tweets Per Tweet	4662.37544523693
Avg. Number of Retweets Per Tweet	1.5344602655543254

Statistics for *tweets_#patriots.txt*:

Avg. Number of Tweets Per Hour	750.8942646068899
Avg. Number of Followers of Users Posting the Tweets Per Tweet	3280.4635616550277
Avg. Number of Retweets Per Tweet	1.7852871288476946

Statistics for *tweets_#sb49.txt*:

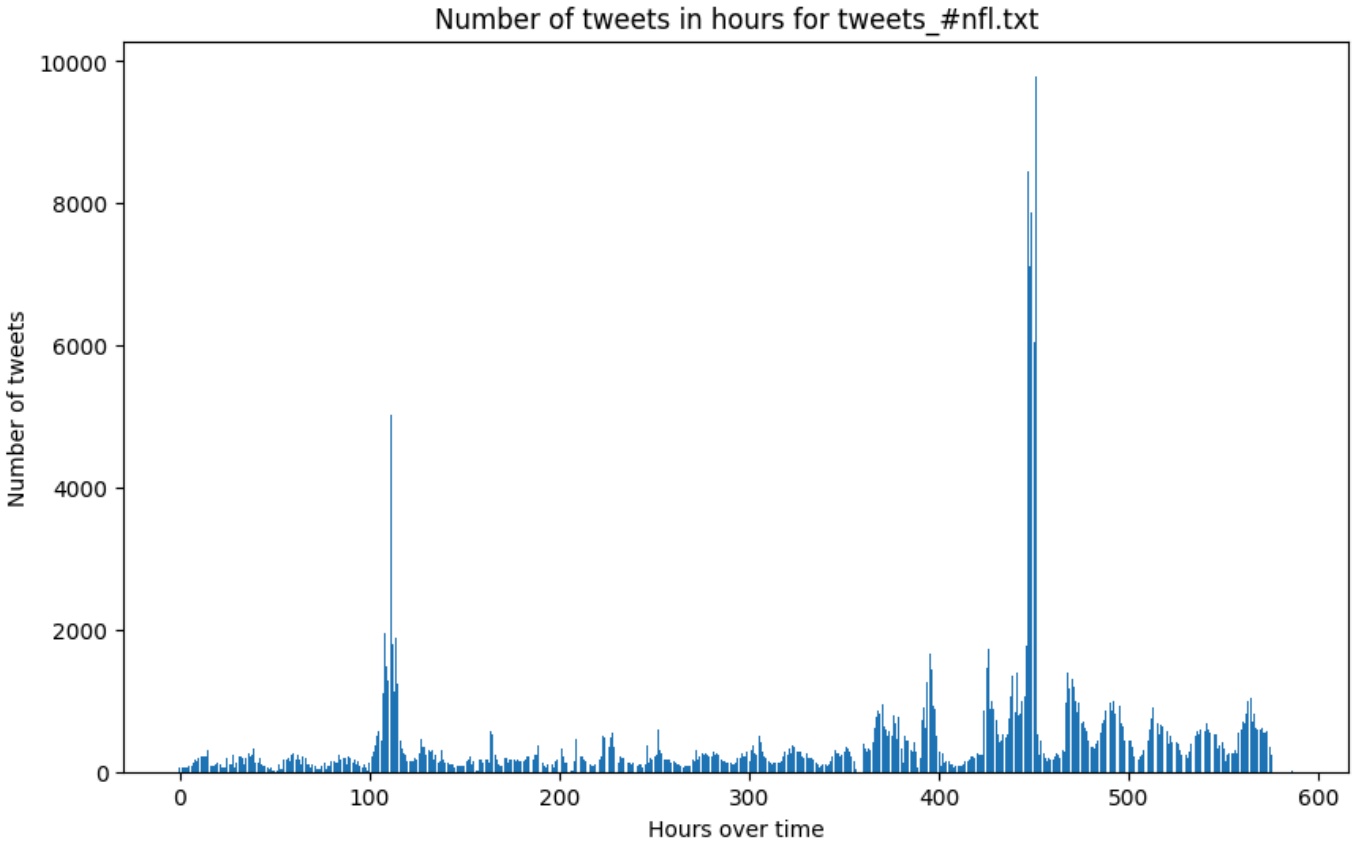
Avg. Number of Tweets Per Hour	1276.8570598680474
Avg. Number of Followers of Users Posting the Tweets Per Tweet	10374.160292019487
Avg. Number of Retweets Per Tweet	2.52713444111402

Statistics for *tweets_#superbowl.txt*:

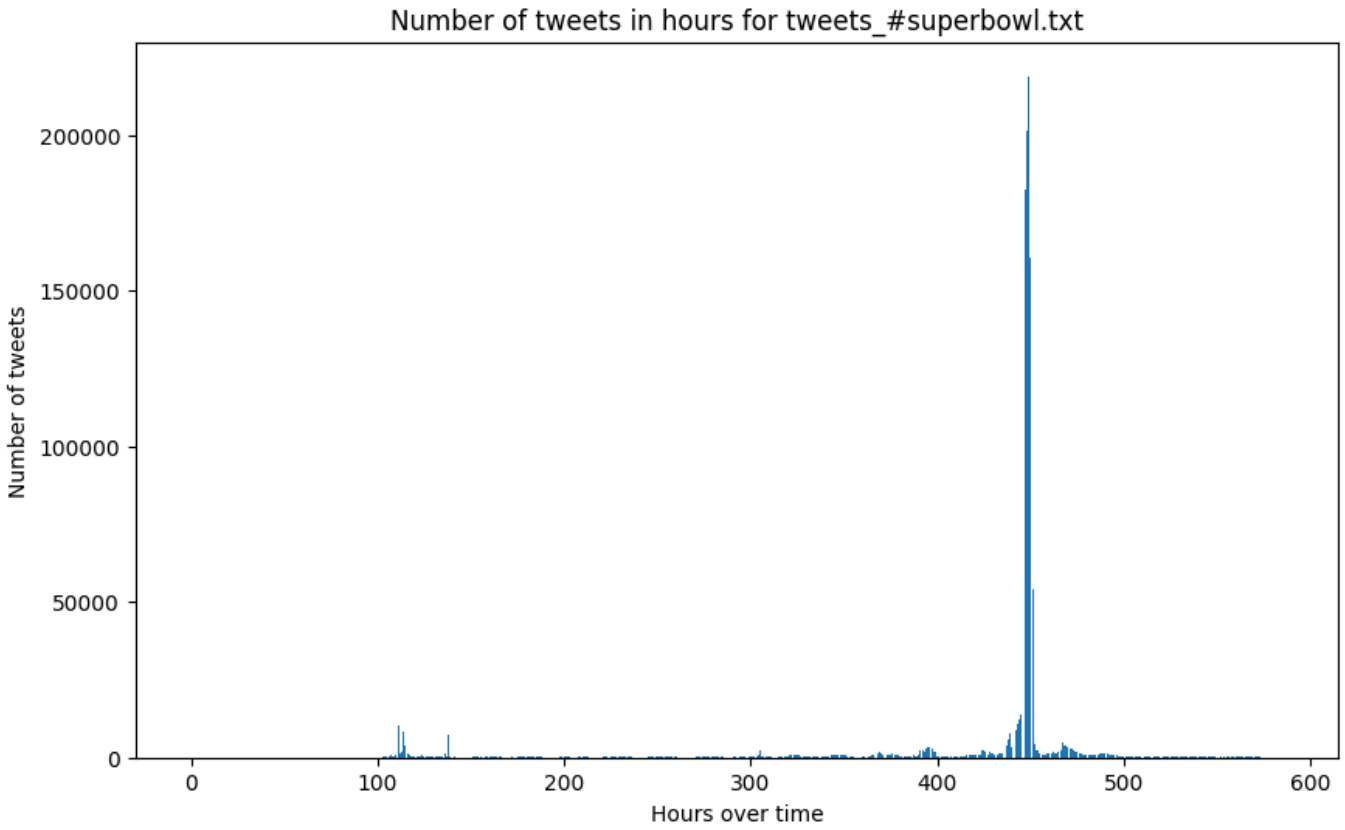
Avg. Number of Tweets Per Hour	2072.11840170408
Avg. Number of Followers of Users Posting the Tweets Per Tweet	8814.96799424623
Avg. Number of Retweets Per Tweet	2.3911895819207736

QUESTION 9.2 Plot "number of tweets in hour" over time for #SuperBowl and #NFL (a bar plot with 1-hour bins). The tweets are stored in separate files for different hashtags and files are named as tweet [#hashtag].txt.

Figure 10: Number of tweets in hour over time



(a) tweets_#nfl.txt



(b) tweets_#superbowl.txt

QUESTION 10 Follow the steps outlined below:

- Describe your task.
- Explore the data and any metadata (you can even incorporate additional datasets if you choose).
- Describe the feature engineering process. Implement it with reason: Why are you extracting features this way - why not in any other way?
- Generate baselines for your final ML model.
- A thorough evaluation is necessary.
- Be creative in your task design - use things you have learned in other classes too if you are excited about them!

Twitter Dataset Overview

The Twitter dataset is divided into six text files, each corresponding to a different hashtag, providing insights into various aspects of the event.
Depending on the different hashtag, the dataset comprises approximately 40 to 2100 tweets per hour on average, with the average number of followers per user posting tweets ranging from 1400 to 10400, and the average number of retweets per tweet ranging from 1.40 to 2.53. The detailed statistics can be found in the Answer to Question 9. From the variance in statistics, it's clear that each of the six dataset should be analyzed independently, and then the results from different datasets can be compared.
The dataset comes in the format of one tweet in each line in JSON string, and sorted with respect to the time sequence. This makes the dataset easier to process and analyze.
In general, the Twitter dataset offers a comprehensive view of Twitter activity surrounding the event, including discussions, reactions, and engagements from users with diverse follower counts.

Our Task Description

Our first task involves analyzing Twitter conversations during Super Bowl 49 through word cloud visualizations. By examining the frequency of terms across different datasets, we aim to identify prevalent topics and discussions. Additionally, we plan to present the most frequent tweets for each hashtag-specific dataset to offer quantitative insights into the most engaging content during the event.

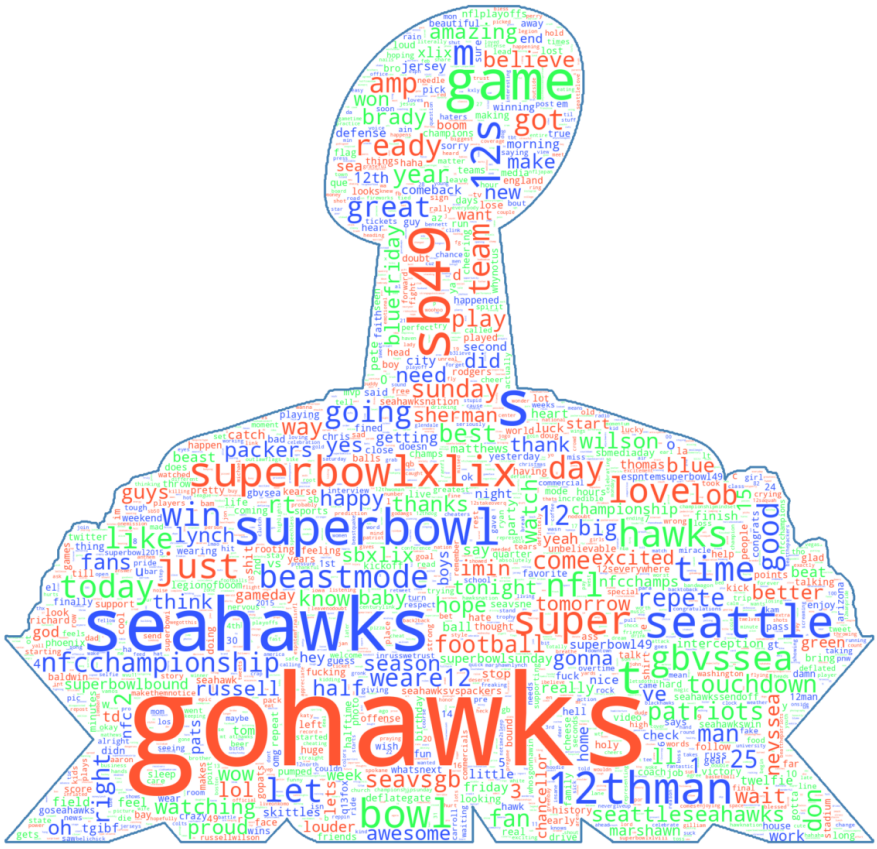
For our second task, we will train a model to predict retweet numbers using a neural network approach. We plan to experiment with varying numbers of hidden units in the network, ranging from 50 to 600, to determine the optimal configuration. Our objective is to minimize the root mean square error (RMSE) in the test set while effectively predicting retweet counts based on tweet characteristics.

Part 1 - Word Cloud and lexical dispersion plots for each hashtag:

In this section, we employed word cloud visualizations on the training tweet data. By analyzing mentions within the dataset, we were able to pinpoint subjects that were frequently cited across different conversations. We then compiled and presented the most frequent associated terms (or the words that appeared most frequently within tweets containing the given hashtag), alongside their respective counts, for each of the six hashtag-specific datasets included in this project. This not only provided a quantitative basis for our analysis but also offered insights into the specific content that resonated most with specific segments of users during the event. Through this methodology, the terms appearing with higher frequency were accorded greater prominence, meaning they were larger in the word cloud visualizations. This direct correlation underscores the utility of the word cloud format in succinctly conveying the relative importance or popular terms within the given tweet data generated during Super Bowl 49.

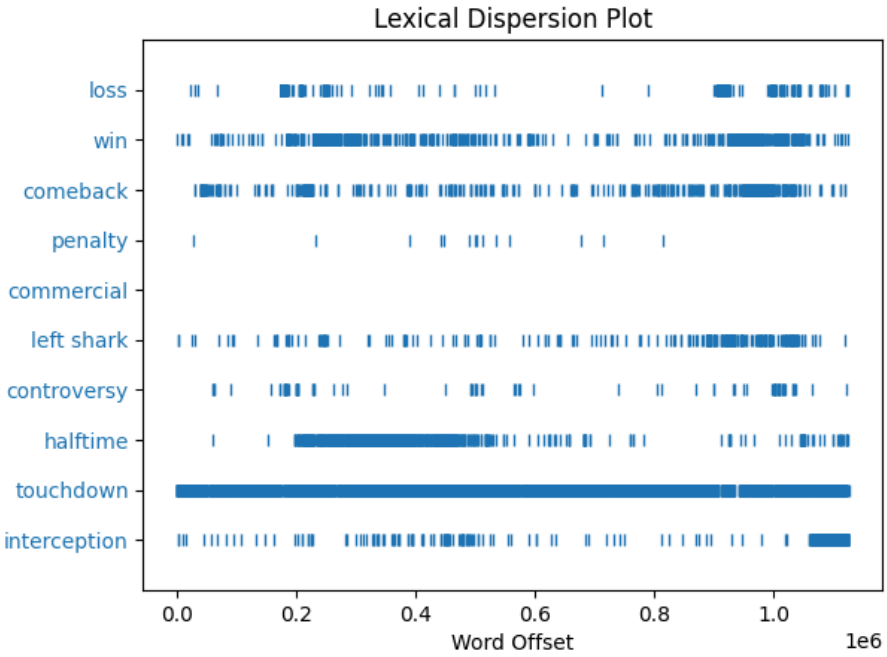
To further our exploration of the training tweet data, we included a lexical dispersion plot for each file. These plots serve as visual representations that map the occurrences of selected topics across all selected tweets related to Super Bowl 49. The topics chosen for analysis are indicative of key events and themes that resonated during the game: 'interception', 'touchdown', 'halftime', 'controversy', 'left shark', 'penalty', 'comeback', 'win', and 'loss'.

Results for *tweets_#gohawks* (1/6)

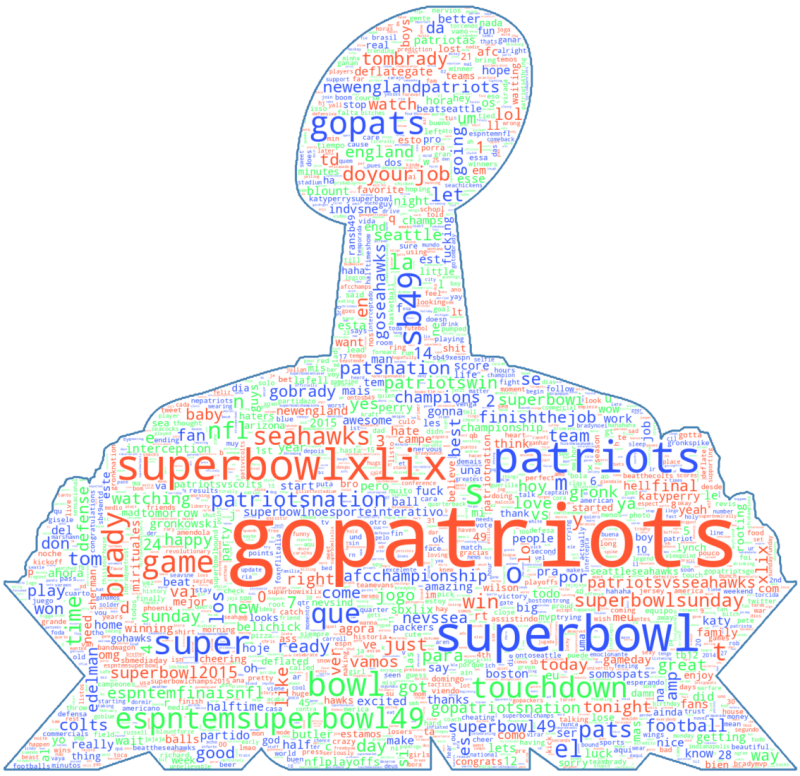


Top Terms:

'gohawks'	171263
'seahawks'	27006
'sb49'	15935
'game'	15011
'superbowl'	12111
'seattle'	9937
'12thman'	9485
'superbowlxlix'	8524

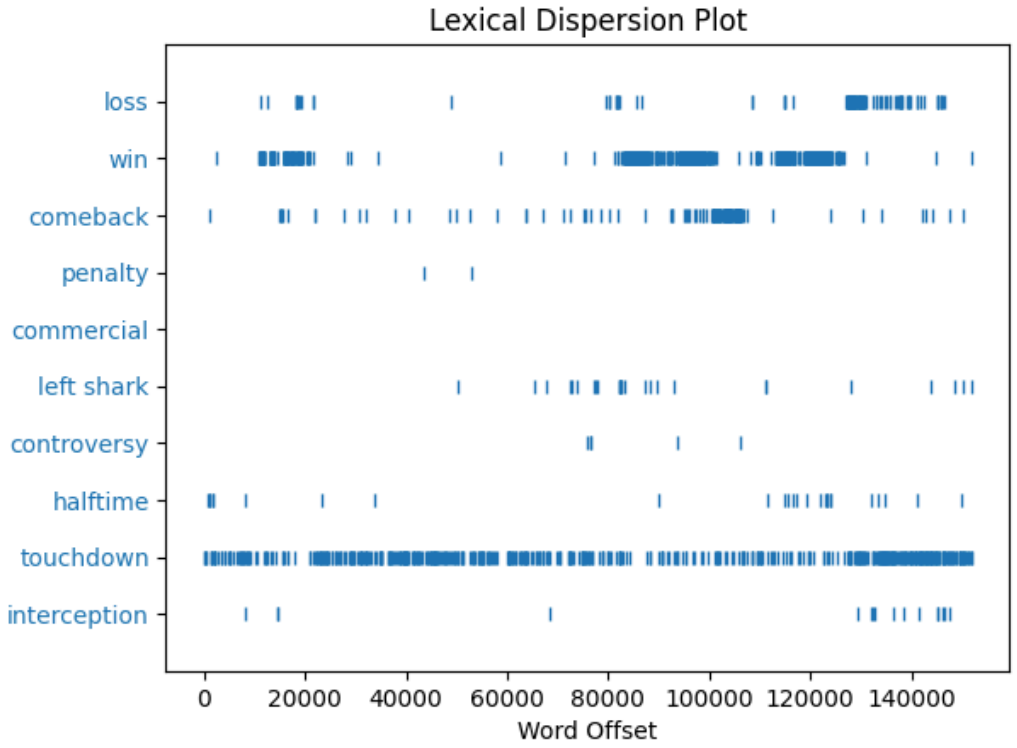


Results for *tweets_#gopatriots* (2/6)

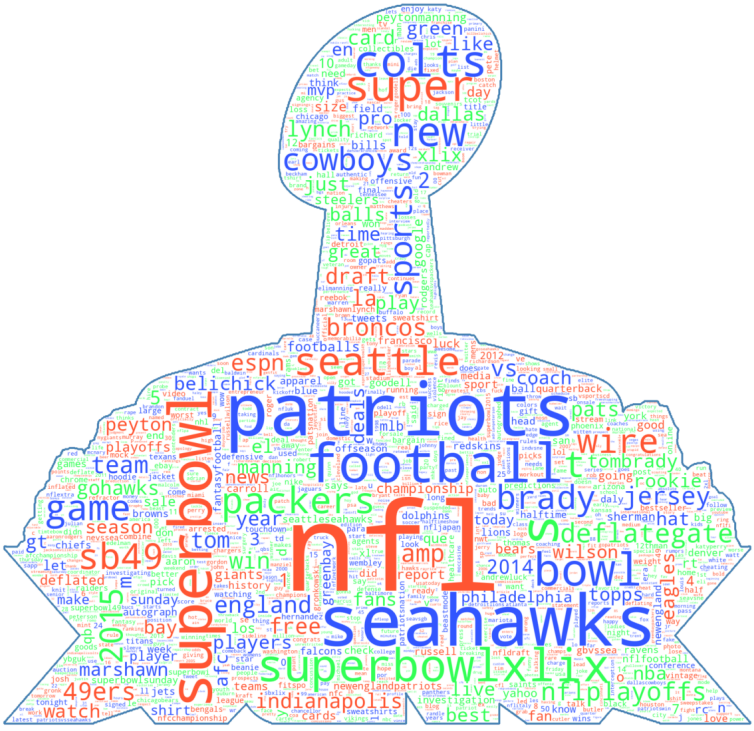


Top Terms:

'gopatriots'	23581
'superbowl'	3685
'patriots'	2769
'superbowlxlix'	2662
'super'	1543
'bowl'	1512
'gopats'	1463
'sb49'	1423

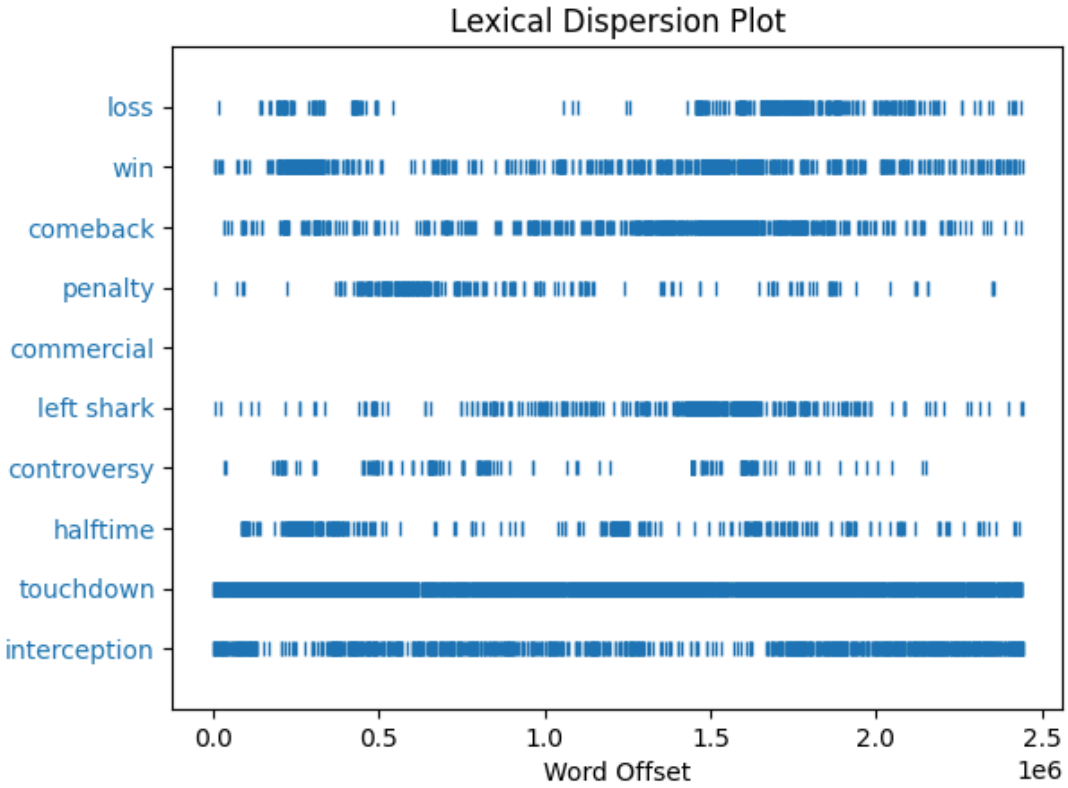


Results for *tweets_#nfl* (3/6)



Top Terms:

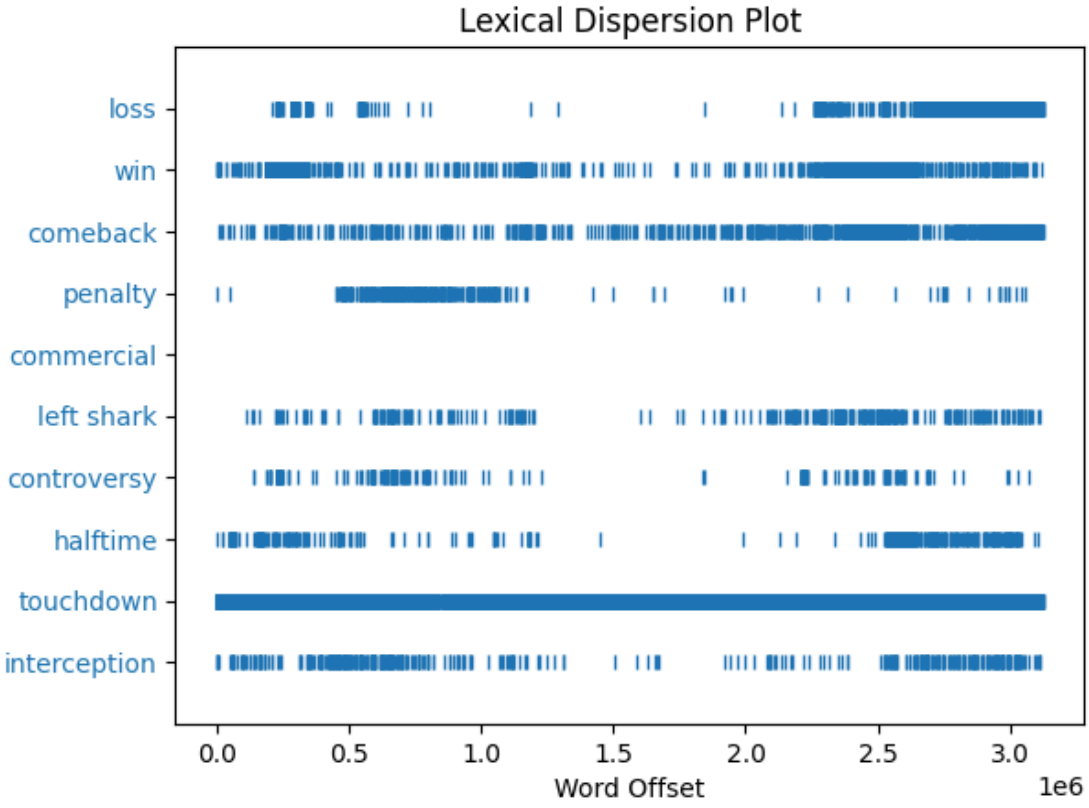
'nfl'	262661
'patriots'	50814
'seahawks'	47533
'superbowl'	32519
'football'	26250
'superbowlxlix'	18109
'bowl'	18093
'colts'	17668



Results for *tweets_#patriots* (4/6)

Top Terms:

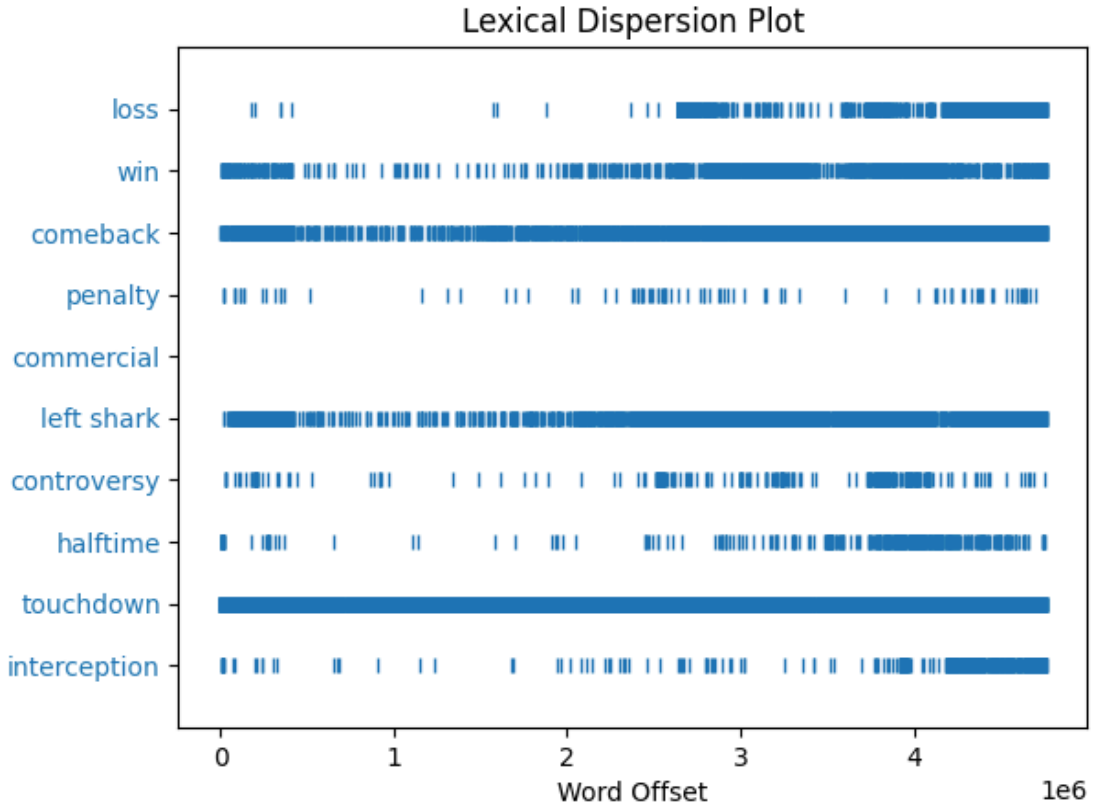
'patriots'	267968
'sb49'	187849
'patriotswin'	179573
'got'	168507
'winning'	166352
'superbowl'	60223
'seahawks'	50220
'nfl'	39638



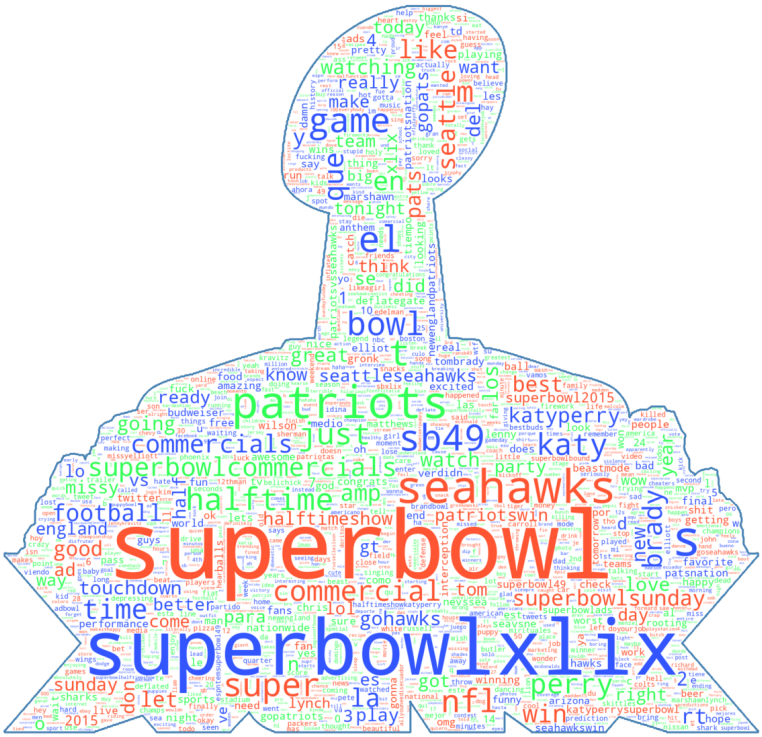
Results for *tweets_#sb49* (5/6)

Top Terms:

'sb49'	742175
'got'	367153
'winning'	363160
'seahawkswin'	195618
'patriotswin'	167265
'superbowl'	51845
'seahawks'	37744
'patriots'	31837



Results for *tweets_#superbowl* (6/6)



Top Terms:

'superbowl'	749704
'superbowlxlix'	468806
'seahawks'	121821
'patriots'	116917
'sb49'	70440
'game'	61952
'super'	56730

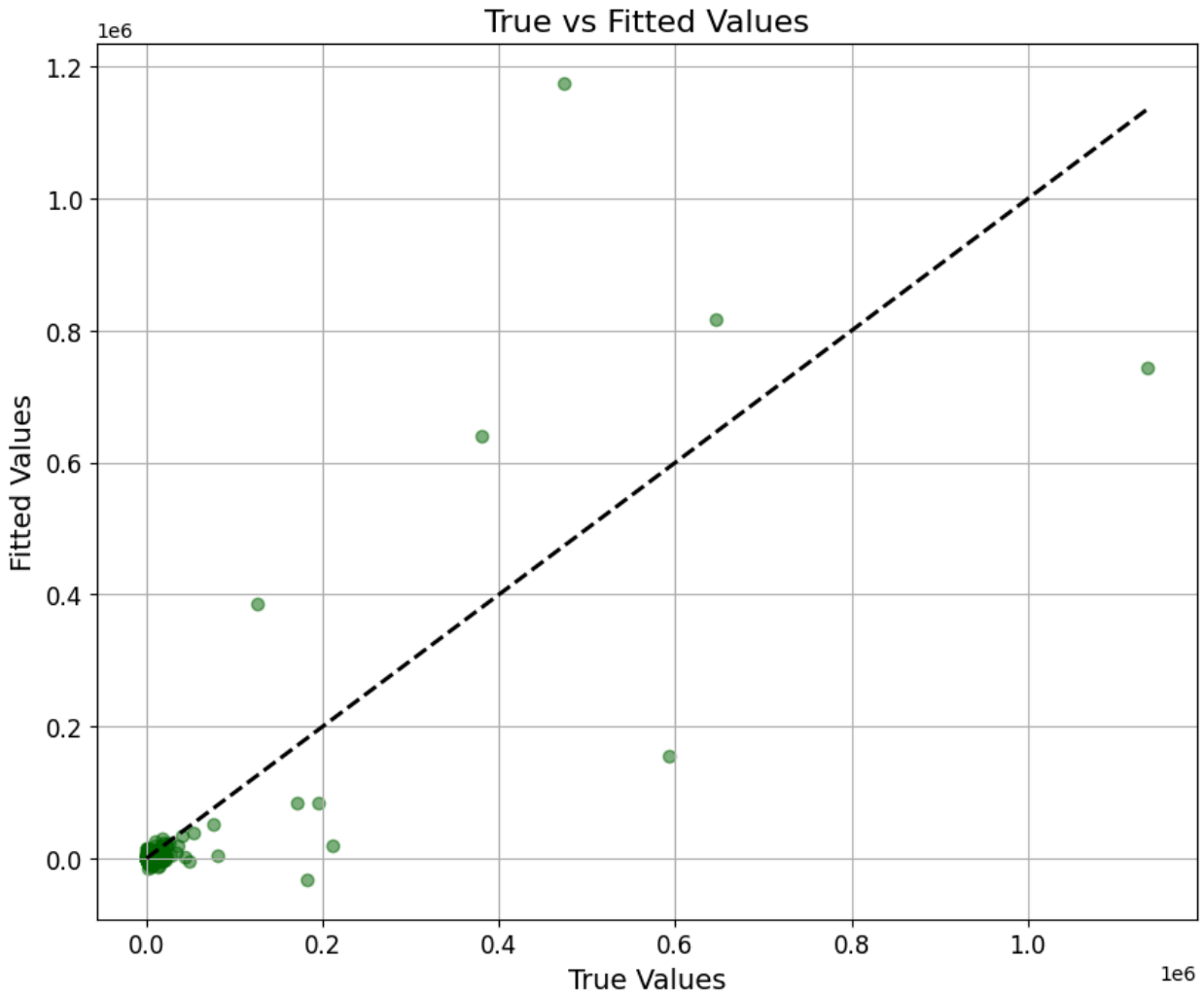
Part 2 - ML Model for Retweet Number Prediction

Using a Neural Network:

We varied the number of hidden units using the variable

```
hidden_units = [50, 100, 200, 300, 500, 600]
```

Results:
min RMSE in testset= 37293.82491504081
hidden layer sizes= 500 50 300
train_MSE= 1938963562.5349667
test_MSE= 153632405.97472566



Using Gradient Boosting:

Hyperparameters:

```
param_grid = {
    'max_depth': [10, 50],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [1, 4],
    'min_samples_split': [2, 10],
    'n_estimators': [100, 400]
}
```

Results:

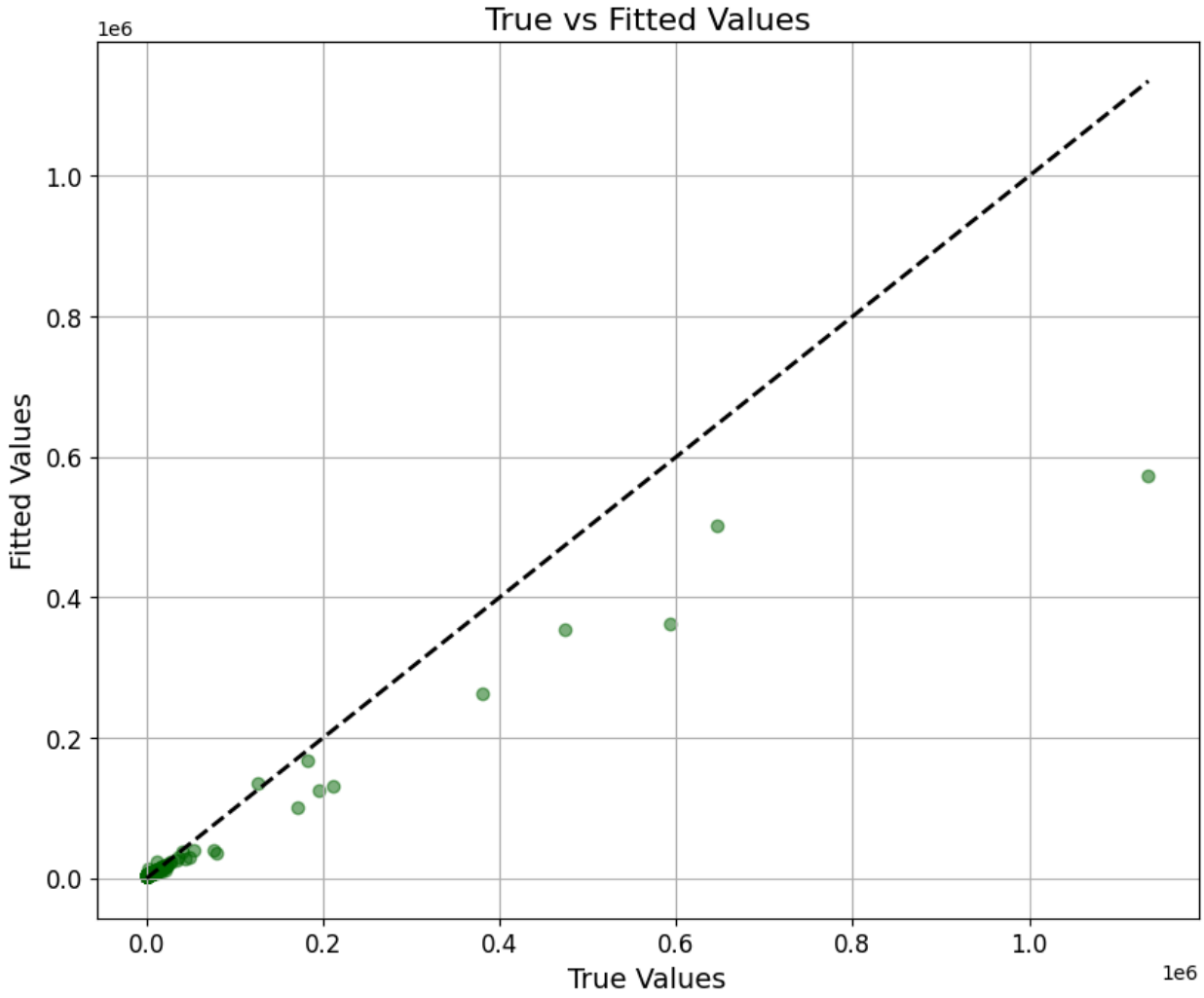
min RMSE in testset = 53851.448443186455

parameters:

max_depth= 10 max_features= sqrt min_samples_leaf= 4 min_samples_split= 2 n_estimators= 100

train_MSE= 758978450.1592393

test_MSE= 565809203.3566786



Evaluation of Part 2 Results:

Neural Network Model:

A neural network was utilized to predict retweet counts, varying hidden units [50, 100, 200, 300, 500, 600]. The minimum RMSE achieved on the test set was 37,293.825, with the optimal hidden layer sizes being [500, 50, 300]. The training MSE was high, at approximately 1,938,963,562.535, indicating potential overfitting as it is significantly larger than the test MSE. The test MSE was 153,632,405.975, which is notably lower than the training MSE. The generated plot showed points clustered around the diagonal, indicating an intermediate-strength linear relationship.

Gradient Boosting Model:

Gradient boosting was applied with a specified set of hyperparameters. The minimum RMSE on the test set was higher than the neural network at 53,851.448. The chosen parameters for the best result were: max_depth=10, max_features='sqrt', min_samples_leaf=4, min_samples_split=2, and n_estimators=100. The training MSE was 758,978,450.159, which is smaller than the neural network's, implying that the model may generalize better. The test MSE was 565,809,203.357, again higher than that of the neural network but closer to its training MSE, suggesting less overfitting compared to the neural network. The generated plot showed a wider spread around the diagonal line than in the neural network's plot.

Overall Evaluation:

The neural network achieved a lower minimum RMSE on the test set compared to the gradient boosting model, suggesting it was more precise for this particular task. However, the substantial difference between training and testing MSE for the neural network model may imply overfitting, where the model learned the training data too well and may not perform as consistently on unseen data. Gradient boosting showed less of a discrepancy between training and testing MSE, suggesting a better generalization, although with a higher RMSE indicating less precise predictions. The time of day, along with the number of tweets, total retweets, follower counts, and maximum follower number, were significant predictors in both models, emphasizing their importance in retweet prediction. Both models presented viable strategies for predicting retweets, with the neural network providing a more precise fit at the risk of overfitting. In contrast, gradient boosting offers a more general solution at the expense of precision.