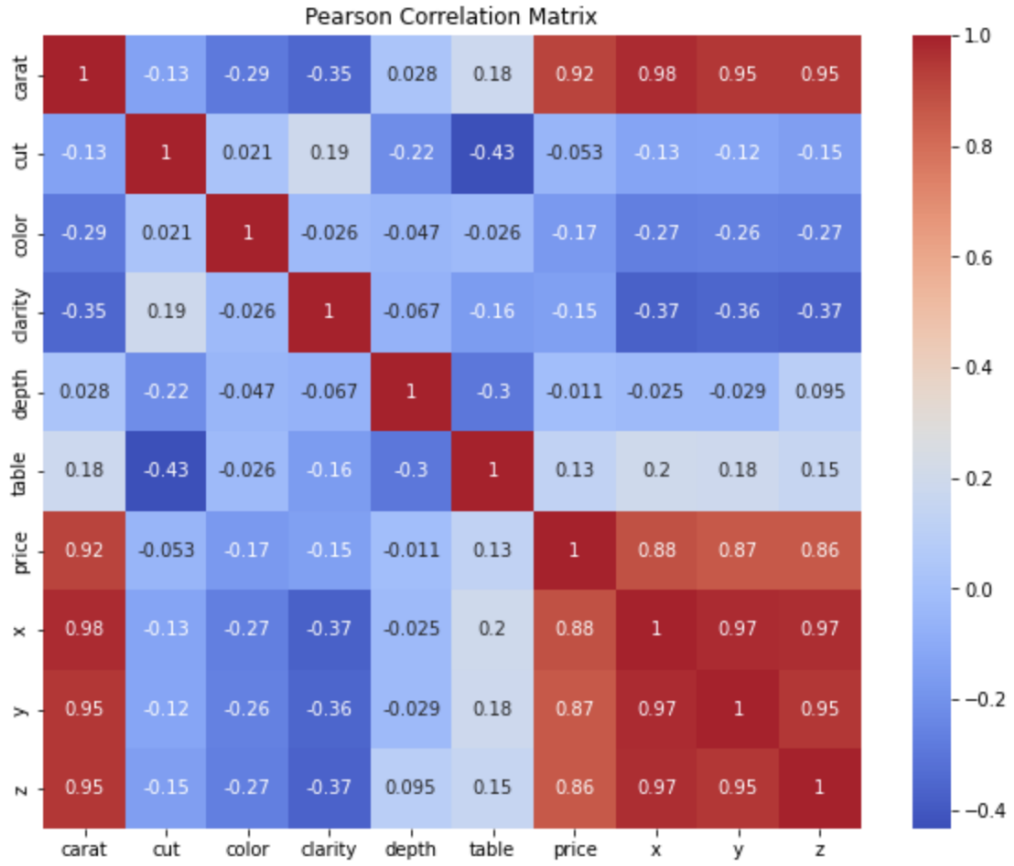
Question 1.1

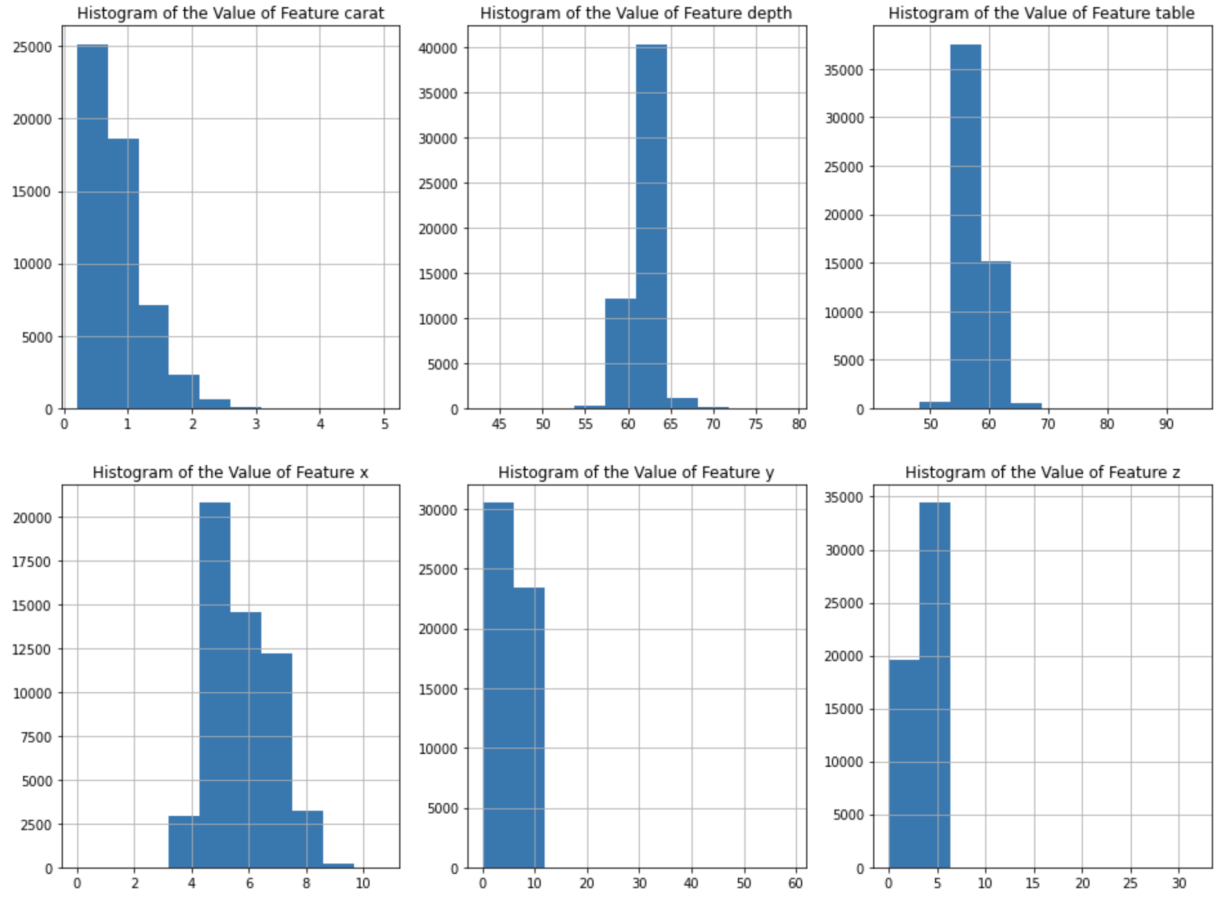


Carat has the highest absolute correlation with the target variable (price), and its correlation with price is 0.92.

And from the heatmap, we can find that the features 'carat', 'x', 'y' and 'z' have high positive correlations with the target variable 'price'. However, the other features ('table', 'depth', 'clarity', 'color', 'cut') have low absolute correlations with the target variable (among them, only 'table' has positive correlation with 'price'). This does not make sense, based on our knowledge, e.g., the correlative between the 'color' and 'price' should be positive (the better the color of the diamond, the higher the price), but the heatmap shows a negative correlation. This pattern suggests the dataset might be biased, diamond with low 'clarity', 'color' or 'cut' tends to have relatively larger 'carat' (or

'x', 'y', 'z'), while the features 'carat' (or 'x', 'y', 'z') have more impacts on the price than the 'clarity', 'color' or 'cut, so this kind of diamond still tend to have high price. And diamond with high 'clarity', 'color', or 'cut' tends to have smaller 'carat' (or 'x', 'y', 'z'), so the diamond would not have very high price. This explains the pattern here, which also fits the real life, i.e. the diamond with higher quality tends to be less and smaller. And this also suggests, we should not only take 'clarity', 'color', 'cut' as the predicting features, (in the dataset, the 'price' does not reflect the real impacts), but must take any or any combinations among 'carat', 'x', 'y', and 'z'.

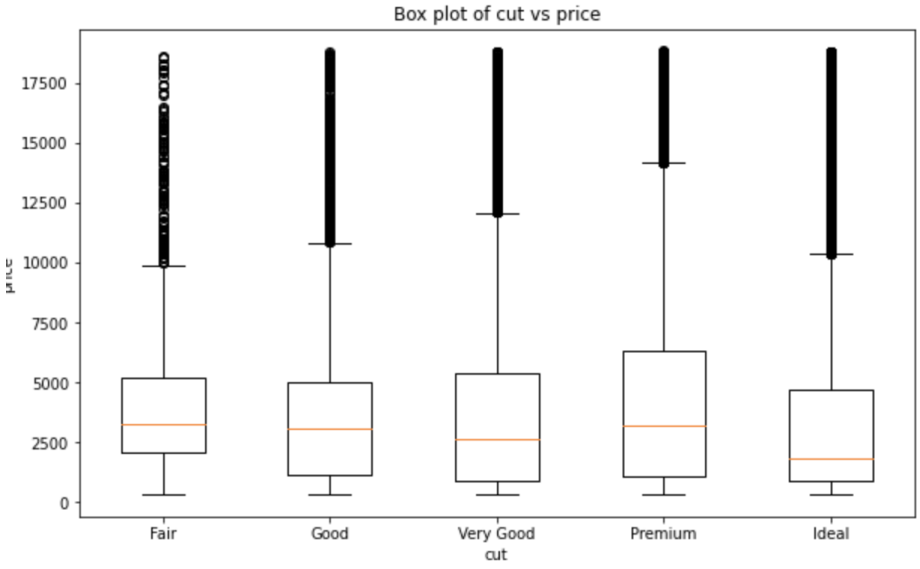
Question 1.2

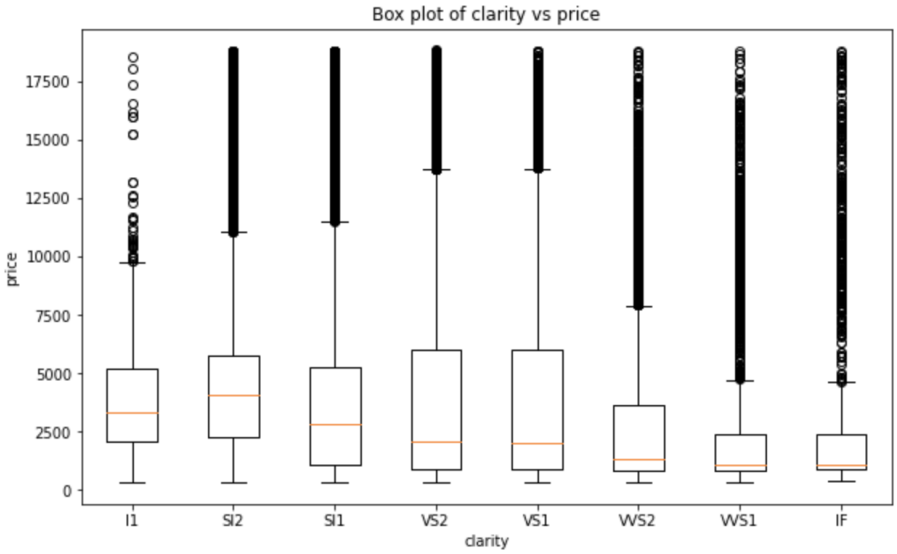


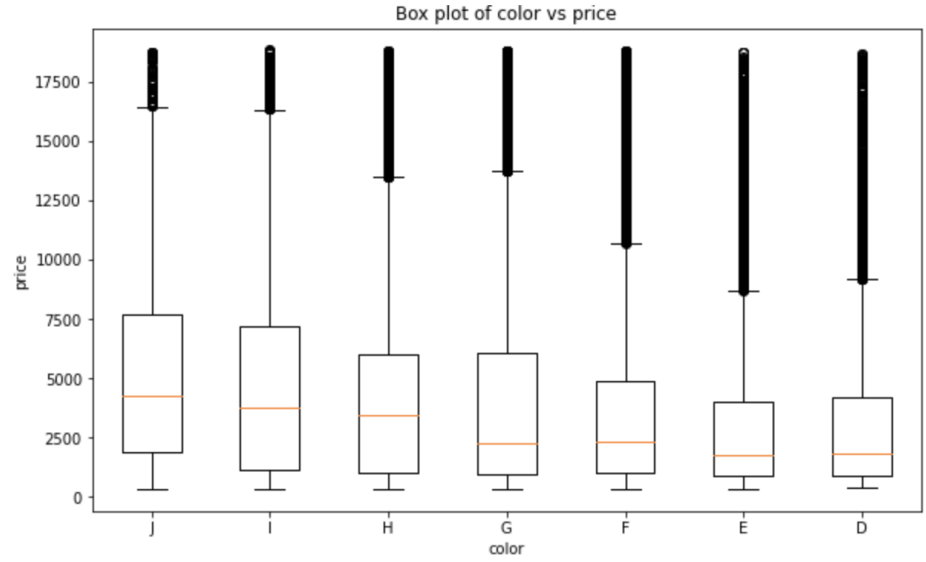
The preprocessing methods that can be used if the distribution of a feature has high skewness:

1. We can remove outliers to reduce the skewness of the distribution of a feature. However, we might remove important information from the data when using this method.
2. We can use log transformation to reduce the impact of outliers and to normalize the distribution of a feature. (There are also some other transformation methods available to reduce the skewness of the distribution of a feature).
3. We can also group the values of the feature into bins (a specified range) based on their values.

Question 1.3

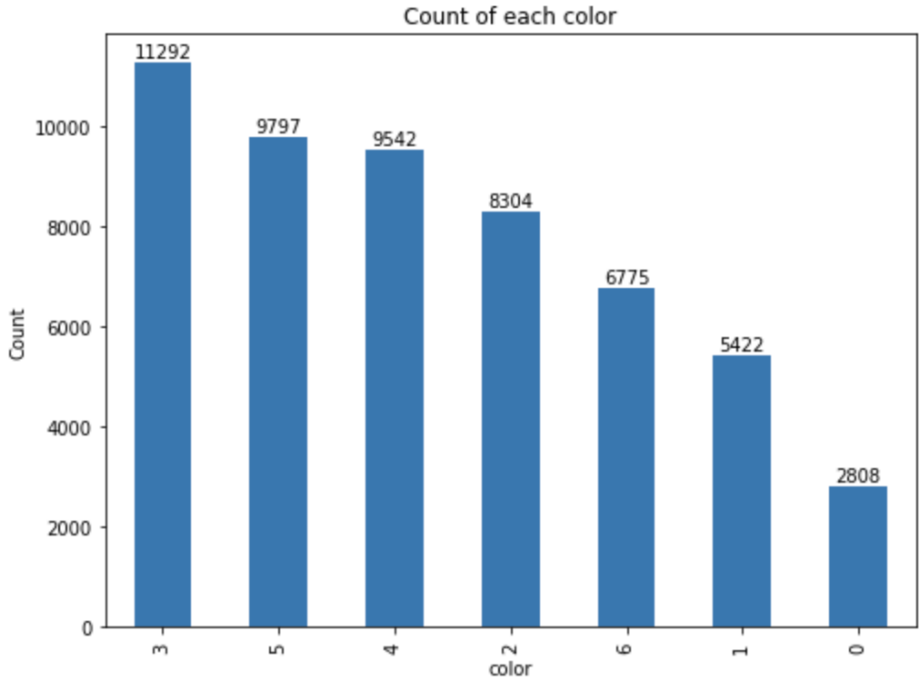


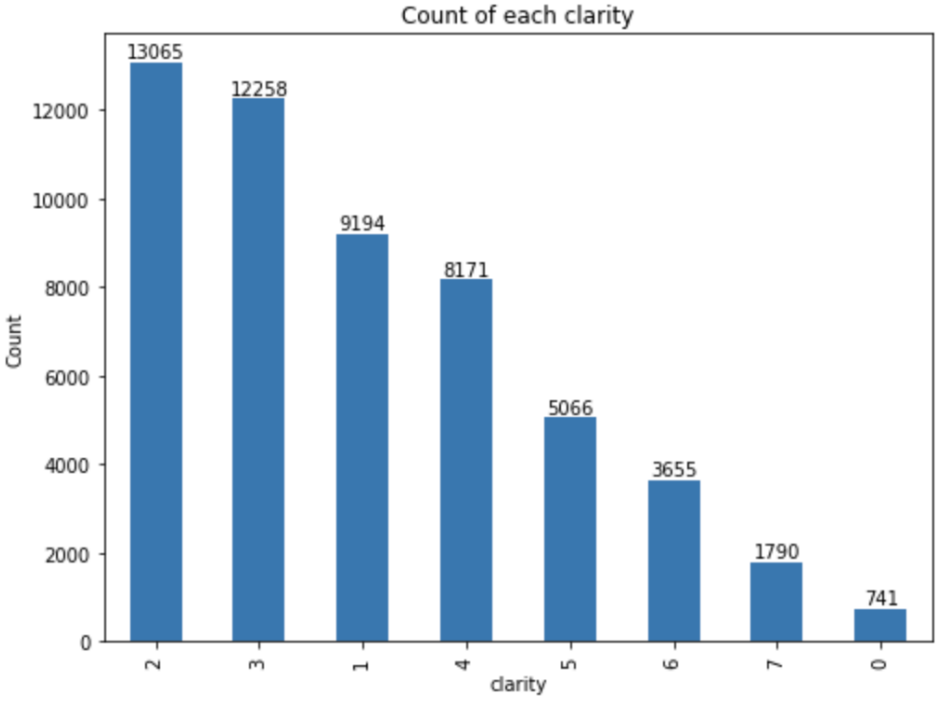


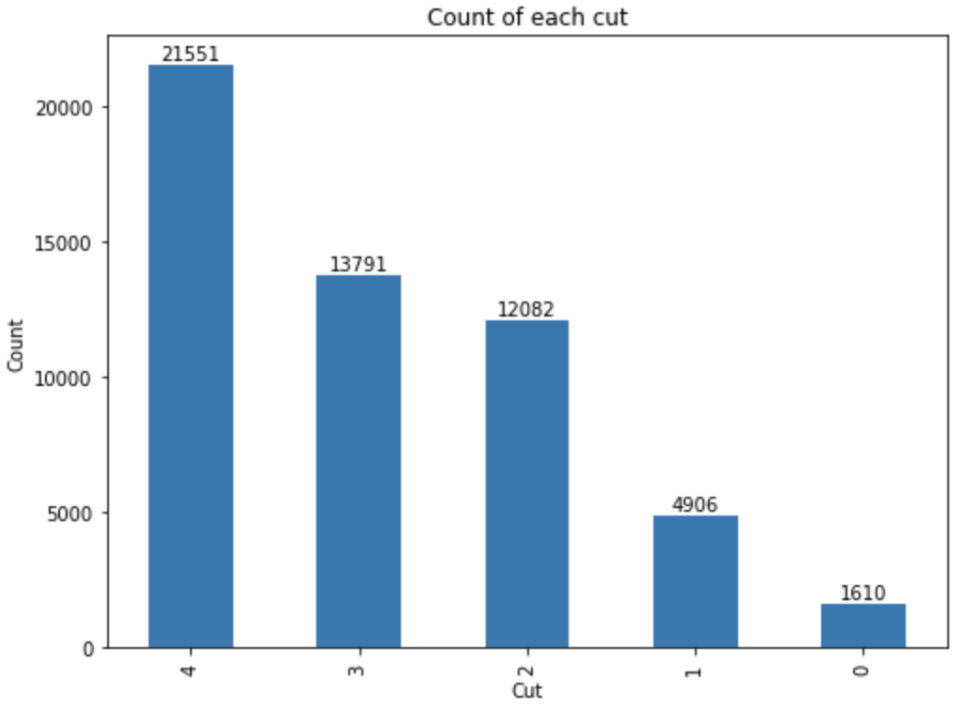


From the boxplots here, we can see that, in most cases, lower quality will lead to a higher mean price. For example, in the color vs price box plot, we can see that trend(D means the best quality yet it has the lowest mean price). We also find a lot of outliers in all box plots. That indicates that the data is skewed and not normally distributed. And since the data is not normally distributed, the mean value of data will not be a good representation of the data. This explains why the better quality leads to a lower mean price. Therefore, more data preprocessing techniques like standardization could be used. And this may also explain that the dataset is biased (high quality diamonds tend to have a smaller size/weight).

Question 1.4


Count of each color
11292
9797
9542
8304
6775
5422
2808
10000
8000
6000
4000
2000
0
Count
3
5
4
2
6
1
0
color


Count of each clarity
13065
12258
9194
8171
5066
3655
1790
741
12000
10000
8000
6000
4000
2000
0
Count
2
3
1
4
5
6
7
0
clarity



Question 2.1

The standardized data is shown below:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	-1.198168	0.981473	0.937163	-1.245215	-0.174092	-1.099672	-0.903594	-1.587837	-1.536196	-1.571129
1	-1.240361	0.085889	0.937163	-0.638095	-1.360738	1.585529	-0.904346	-1.641325	-1.658774	-1.741175
2	-1.198168	-1.705279	0.937163	0.576145	-3.385019	3.375663	-0.904095	-1.498691	-1.457395	-1.741175
3	-1.071587	0.085889	-1.414272	-0.030975	0.454133	0.242928	-0.901839	-1.364971	-1.317305	-1.287720
4	-1.029394	-1.705279	-2.002131	-1.245215	1.082358	0.242928	-0.901588	-1.240167	-1.212238	-1.117674

Question 2.2

By using feature selection methods such as 'mutual information regression' and computing F scores allows us to find the most relative and useful features and drop unrelated features. By doing so, we can improve the performance(reducing the RMSE) of our model by reducing overfitting.

For some models such as the linear regression model, we can see a significant improvement in terms of RMSE(lower RMSE). However, for those non-linear models

such as neural network models(activation layers introduce non-linearity), there may not be a significant drop in RMSE because those models can automatically select relevant features.

We find that features 'depth' and 'table' have the lowest MI w.r.t to the target(price).

Below is the MI of all features:

```
carat: 1.653
cut: 0.057
color: 0.138
clarity: 0.217
depth: 0.032
table: 0.035
x: 1.412
y: 1.423
z: 1.362
```

Question 3

(For linear regression and neural network models, we choose ‘carat’, ‘cut’, ‘color’, ‘clarity’, ‘x’, ‘y’, and ‘z’ as features. For polynomial regression and random forest, we choose ‘carat’, ‘clarity’, ‘x’, ‘y’, and ‘z’ as features.)

Linear Regression without Regularization:

Average RMSE error for training set: `0.305721173659`
Average RMSE error for testing set: `0.30643155`

lasso regression:

Average RMSE error for training set: 0.38109265
Average RMSE error for testing set: 0.38103165

Ridge regression:

Average RMSE error for training set: 0.305724245559
Average RMSE error for testing set: 0.30638856

Polynomial Regression (linear regression w/o regularization):

With Degree of 1:
Average RMSE error for training set: `0.335352057073`
Average RMSE error for testing set: `0.33665765`

With Degree of 2:
Average RMSE error for training set: 0.275594024800
Average RMSE error for testing set: 0.45453605
With Degree of 3:
Average RMSE error for training set: 0.261947452036
Average RMSE error for testing set: 3.29305735

With Degree of 4:
Average RMSE error for training set: 0.335352057073
Average RMSE error for testing set: 0.33665765

With Degree of 5:
Average RMSE error for training set: 0.254526929579
Average RMSE error for testing set: 348.58220441

With Degree of 6:
Average RMSE error for training set: 0.245570057711
Average RMSE error for testing set: 280656967.62288600

Polynomial Regression (Lasso Regression):

With Degree of 1:
Average RMSE error for training set: 0.38109265
Average RMSE error for testing set: 0.38103165

With Degree of 2:
Average RMSE error for training set: 0.37826211
Average RMSE error for testing set: 0.37976554
With Degree of 3:
Average RMSE error for training set: 0.37428226
Average RMSE error for testing set: 1.18037188

With Degree of 4:
Average RMSE error for training set: 0.35497330
Average RMSE error for testing set: 2.27938330

With Degree of 5:
Average RMSE error for training set: 0.35566231
Average RMSE error for testing set: 7.46482009

With Degree of 6:
Average RMSE error for training set: 0.35504569
Average RMSE error for testing set: 17.86048476

Polynomial Regression (Ridge Regression):

With Degree of 1:
Average RMSE error for training set: 0.33535208
Average RMSE error for testing set: 0.33665151

With Degree of 2:
Average RMSE error for training set: 0.27559863
Average RMSE error for testing set: 0.44476629
With Degree of 3:
Average RMSE error for training set: 0.26224964
Average RMSE error for testing set: 5.76968797

With Degree of 4:
Average RMSE error for training set: 0.25514220
Average RMSE error for testing set: 249.07537081

With Degree of 5:
Average RMSE error for training set: 0.25062402
Average RMSE error for testing set: 9718.92123274

With Degree of 6:
Average RMSE error for training set: 0.24783831
Average RMSE error for testing set: 164622.09044429

Random Forest:
Average RMSE error for training set: 0.282143078864174
Average RMSE error for testing set: 0.28308853181178417
Average OOB score: 0.9195686138270596

From the source code of the _set_oob_score in RandomForestRegressor (from the given link), we can find OOB score is calculated by taking the average R2 score across all outputs from decision trees. This means that the OOB score is a measure of how well the random forest model is able to predict the target values for new data, and the R2 score is a measure of how well the model fits the training data for each output.

Question 4.1

(a) ordinary least squares (linear regression without regularization): It has no regularization effect.
(b) Lasso: applies a L1 regularization. It is a penalty term proportional to the absolute value of the coefficients added to the objective function.The first order derivative of the penalty term w1, w2, ... will be 1 if the parameter is non zero and 0 if the parameter is 0. As a result, Lasso encourages sparsity in the learned parameter set. That means some parameters will be set to zero.
(c) Ridge regression: a L2 regularization. It is a penalty term proportional to the squared coefficients added to the objective function. The derivative of those squared parameters will be themselves. Because the penalty term is differentiable, there may be a solution that minimizes the objective function without setting any coefficients exactly to zero. Therefore, ridge regression encourages smaller values of the coefficients but does not explicitly encourage any coefficients to be exactly zero.

Question 4.2

Best Lasso regression alpha and corresponding RMSE:
Best alpha: 0.0006614740641230146
Best training RMSE: 0.3057930932191351
testing RMSE: 0.30770741

Best ridge regression alpha and corresponding RMSE:
Best alpha: 34.55107294592218
Best training RMSE: 0.3059634521525212
testing RMSE: 0.30760941

The best regularization scheme is ridge regression, and the optimal penalty parameter is alpha = 34.551. We get this answer by selecting 40 numbers among 1e-4 to 1e4 as alpha and we choose the alpha that has the lowest RMSE.

Question 4.3

Ridge regression RMSE with unscaled data:
Best alpha: 5.2233450742668435
Best training RMSE: 1220.8294111899638
testing RMSE: 1227.14856110

We used the unscaled data to train our ridge regression model and the RMSE we get is 1227.15. This is much larger than the RMSE we get from the ridge regression model that uses scaled data. Therefore, we believe that feature standardization plays an important role in improving the model performance.

Question 4.4

The p-value measures the statistical significance of the relationship between one of the features and the target variable(here, it is price). The p-value is used to test the null hypothesis that the coefficient of a feature is equal to zero(when the coefficient is 0, the feature has no effect on the final predicted result, so no relationship between that feature and the target). We can infer the most significant features by setting a significance level for the p-value. If the p value of that feature is less than that significance level, the null hypothesis is rejected and we believe there is a relationship between that feature and the target.

Question 5.1

When the degree is 1, we can find the feature 'carat' has the highest absolute coefficient value, which means with the degree of 1, 'carat' is the most salient feature.

When the degree is 2, we can find the features 'carat', 'y', 'x y', and 'y z' have relatively high absolute coefficient values, which means with the degree of 2, 'carat', 'y', 'x y' and 'y z' are salient features.

But as the degree increases, the features of higher degree might affect the target variable, but the effect would not be too large (they can become highly correlated with each other), so we do not consider them as the most salient features.

And as the degree increases, from the RMSE scores for training set and testing set, we can find the model tends to be more overfitting. So, we do not consider these features as the most ones

Question 5.2

For the polynomial regression model without regularization, the best degree of polynomial is 1.

For the polynomial regression model with Lasso, the best degree of polynomial is 2.

For the polynomial regression model with Ridge, the best degree of polynomial is 1.

I found the optimal degree by comparing the average training and testing RMSE of the models, the ones I picked almost have the lowest RMSE either in the training dataset or the testing dataset.

A very high-order polynomial implies the model would fit on the training data very well, but it might not be too good for the generalized data (overfitting), so its performance on testing data would be not good and sometimes very bad. From the results above, we can find the average RMSE for training data is slightly decreasing as the degree increases, but the average RMSE for testing data is dramatically increasing as the degree increases (e.g. when the degree is 1 with linear regression w/o regularization, the average RMSE for training and testing data are both about 0.34, but when the degree is 6, the RMSE for training data is 0.25, and the RMSE for testing data is 280656967.62).

Question 6.1

The good hyper-parameter set we find is listed below:
Best parameters: {'alpha': 0.05, 'hidden_layer_sizes': (50, 100)}
The corresponding train RMSE: 0.13477331610402857
The corresponding test RMSE: 0.13620513090531988

Question 6.2

The performance of the Neural network model is much better than the performance of linear regression in terms of RMSE. The Neural network model has an RMSE of 0.14 while the linear regression has an RMSE of 0.305. This is because the neural network model introduces non-linearity by adding an activation layer while linear regression assumes the relation between features and target is always linear. In real world problems, the relationship between the feature and the target is not linear, which means that a neural network may be better able to capture patterns in the data.

Also, a neural network model can automatically find and select the most important features while we can only manually select the features when using a linear regression model. The input features for linear regression need to be transformed and selected manually before being used in the model. Neural networks, on the other hand, can learn the relevant features automatically from the raw data.

Question 6.3

The MLPRegressor class in scikit-learn uses the ReLU activation layer by default. It is a very popular activation layer used in deep learning and it has many advantages. It is

computationally efficient to compute because it's a simple max(0,x) function. It can help to avoid the vanishing gradient problem because it does not saturate for positive input values.

Question 6.4

1. Increasing the depth of the network will lead to higher complexity and that can lead to overfitting problems. That will lead to a poorer performance on the testing set.
2. Increased depth will lead to more parameters. That also means more calculations are needed to compute all the parameters in the network. Therefore, the iteration needed to converge will increase and the time taken to train the model will increase.
3. Increasing depth of network may also lead to vanishing gradients problem. Since all the parameters are computed at the back propagation, the gradient close to the end will be calculated at the first and the gradient will be smaller and smaller as the gradient back propagates and finally becomes 0 if the depth is large enough. If the gradient of a layer is zero, the gradient of the layers before that layer will also be 0 due to the mechanics of back propagation(dz/da * dl/dz = 0 if dl/dz = 0). Vanishing gradients problem will make the network learn ineffectively because no gradient can be learned(no more learning) when the gradient is 0.

Question 7.1

- Maximum number of features: increasing the value of the hyperparameter max_features (maximum number of features) can lead to a more complex model with more variance, but this can also lead to overfitting.
  Setting a suitable number of features can avoid the model from relying too much on individual features that are not representative for the whole dataset, and thus can lead to regularization and prevent the model from overfitting the training data.
- Number of trees: increasing the value of the hyperparameter n_estimators (number of trees) usually leads to better performance, but may cause the training time to increase a lot.
  Increasing the number of trees can lead to regularization by reducing overfitting (more trees means that the model's predictions will be averaged over many different trees, and thus reduce the impact of individual trees that might be overfitting the data).
- Depth of each tree: increasing the value of the hyperparameter max_depth (depth of each tree) can also lead to a more complex model with more variance,

and cause overfitting.
Setting a limitation to the depth of each tree can lead to regularization by preventing the model from being too complex and overfitting the training data.

Question 7.2

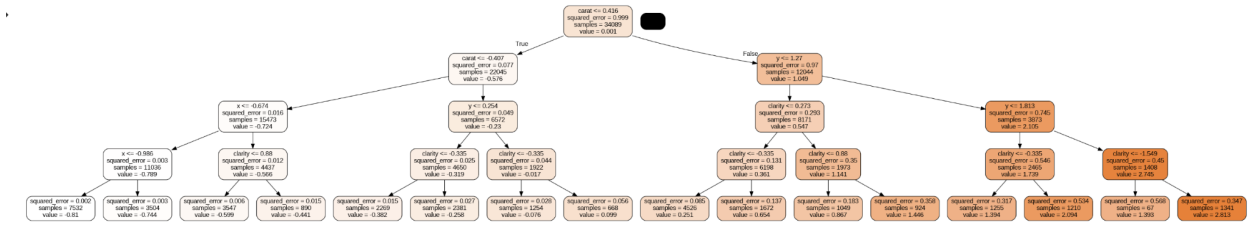
In a random forest, each decision tree splits the data into smaller regions based on the thresholds applied to each feature. Though an individual decision tree might not be able to form complex decision boundaries, the ensemble of trees (each trained on different subsets of data and features) is able to create a highly non-linear decision boundary with its mechanisms - random feature selection and bootstrap aggregation.

With random feature selection, each decision tree in the ensemble uses a different subset of features to make decisions (lead to different splits and different decision boundaries).

With bootstrap aggregation, each decision tree in the ensemble is trained on a different subset of the training data (leading them to different decision boundaries).

In these ways, the ensemble can find complex interactions between features and thus leads to highly non-linear decision boundaries.

Question 7.3



(sorry for the unclear figure)

The feature 'carat' is selected for branching at the root node. We can infer that this feature is one of the most important features for predicting the response variable ('price'). And this is because the root node of a decision tree is where making the first split, so the feature chosen for this split is the one that provides the most information gain.

And yes, the important feature corresponds to what we got in part 3.3.1.

Question 7.4

The OOB score: `0.9194478956684811`

From the source code of the _set_oob_score in RandomForestRegressor, we can find OOB score is calculated by taking the average R2 score across all outputs from decision trees. This means that the OOB score is a measure of how well the random forest model is able to predict the target values for new data, and the R2 score is a measure of how well the model fits the training data for each output.
And from here, we have the OOB score approximately equals to 0.92, which is significantly higher than the accuracies from above models, this is a good sign for random forest, indicating a better performance.
OOB score works similarly to R2 score, both of them indicate a better performance (a good fit of the model to the data) when their scores are larger.

Question 8.1

Here is the list of the important hyperparameters along with a search space for the tuning of these parameters for LightGBM:

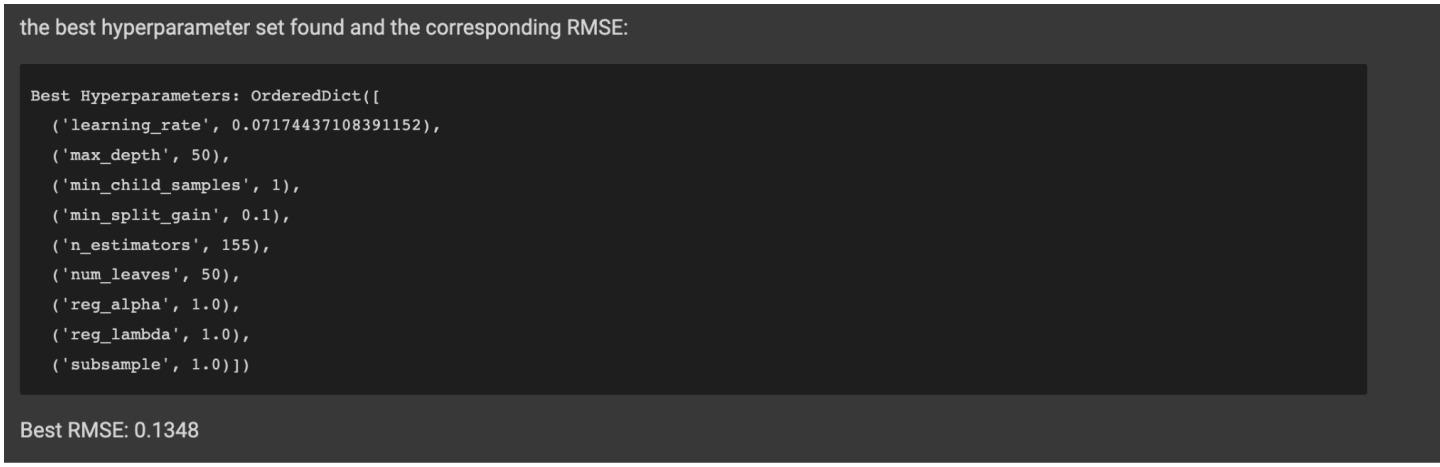
'learning_rate': Real(1e-4, 1e-1, 'log-uniform'),
'num_leaves': Integer(10, 50),
'max_depth': Integer(5, 50),
'min_child_samples': Integer(1, 20),
'subsample': Real(0.1, 1.0, 'uniform'),
'min_split_gain': Real(0.1, 1.0, 'uniform'),
'reg_alpha': Real(0, 1.0, 'uniform'),
'reg_lambda': Real(0, 1.0, 'uniform'),
'n_estimators': Integer(50, 200)

Here is the list of the important hyperparameters along with a search space for the tuning of these parameters for LightGBM:

```
'learning_rate': Real(1e-4, 1e-1, 'log-uniform'),
'num_leaves': Integer(10, 50),
'max_depth': Integer(5, 50),
'min_child_samples': Integer(1, 20),
'subsample': Real(0.1, 1.0, 'uniform'),
'min_split_gain': Real(0.1, 1.0, 'uniform'),
'reg_alpha': Real(0, 1.0, 'uniform'),
'reg_lambda': Real(0, 1.0, 'uniform'),
'n_estimators': Integer(50, 200)
```

Question 8.2

Best test set RMSE: 0.1348
Best Hyperparameters:
('learning_rate', 0.07174437108391152),
('max_depth', 50),
('min_child_samples', 1),
('min_split_gain', 0.1),
('n_estimators', 155),
('num_leaves', 50),
('reg_alpha', 1.0),
('reg_lambda', 1.0),
('subsample', 1.0)



Question 8.3

The hyperparameters that help improve performance are: learning rate, n_estimators (number of trees), and max_depth (maximum depth of each tree).

- learning_rate: a suitable learning rate can help improve the performance of a model by controlling the speed and stability of the optimization process (avoid vanishing/exploding gradient).
- n_estimators: a suitable n_estimators can help to make the model more robust to noise (more decision trees makes the model more complex, and the error caused by an individual tree would be smoothed by the other trees).
- max_depth: a suitable max_depth can make the model more complex (is able to capture more potential patterns) and perform better on generalized data.

The hyperparameters that help regularization are: learning_rate, max_depth, reg_alpha and reg_lambda.

- learning_rate: a suitable learning rate slows down the optimization process and reduces the risk of overfitting by making smaller updates to the model parameters. Thus, it can help regularization.
- max_depth: a suitable max_depth can avoid the model fits the training data too well and performs badly on other unseen data.
- reg_alpha and reg_lambda: reg_alpha and reg_lambda control the regularization applied to the model. These hyperparameters prevent overfitting by adding a penalty to the loss function for large parameter values.

The hyperparameters that affect the fitting efficiency are: learning_rate, min_split_gain.

- learning_rate: a suitable learning rate can reduce the number of iterations required to reach convergence, and thus improve the fitting efficiency.
- min_split_gain: a suitable minimum split gain can reduce the complexity of the trees by reducing the number of splits and nodes required to capture the potential patterns in the data, and thus improve the fitting efficiency.

Question 9.1:

For hashtag #nfl:

```
The Average Number of Tweets Per Hour is: 397.0213901819841
The Average Number of Followers of Users Posting the Tweets Per Tweet is: 4662.37544523693
The Average Number of Retweets Per Tweet is: 1.5344602655543254
```

For hashtag #gohawks:

```
The Average Number of Tweets Per Hour is: 292.48785062173687
The Average Number of Followers of Users Posting the Tweets Per Tweet is: 2217.9237355281984
The Average Number of Retweets Per Tweet is: 2.0132093991319877
```

For hashtag #gopatriots:

```
The Average Number of Tweets Per Hour is: 40.95469800606194
The Average Number of Followers of Users Posting the Tweets Per Tweet is: 1427.2526051635405
The Average Number of Retweets Per Tweet is: 1.4081919101697078
```

For hashtag #patriots:

```
The Average Number of Tweets Per Hour is: 750.89426460689
The Average Number of Followers of Users Posting the Tweets Per Tweet is: 3280.4635616550277
The Average Number of Retweets Per Tweet is: 1.7852871288476946
```

For hashtag #sb49:

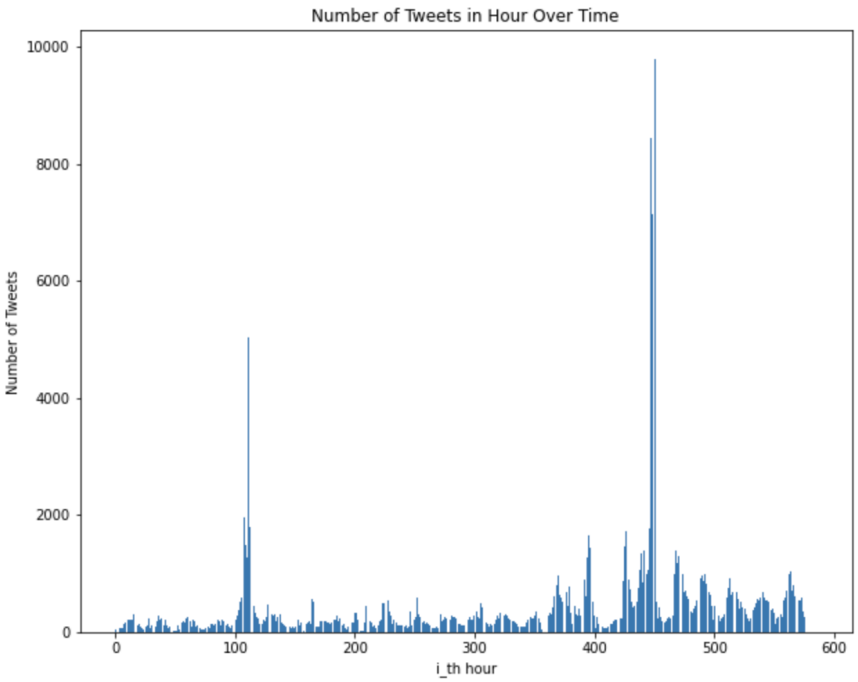
```
The Average Number of Tweets Per Hour is: 1276.8570598680474
The Average Number of Followers of Users Posting the Tweets Per Tweet is: 10374.160292019487
The Average Number of Retweets Per Tweet is: 2.52713444111402
```

For hashtag #superbowl:

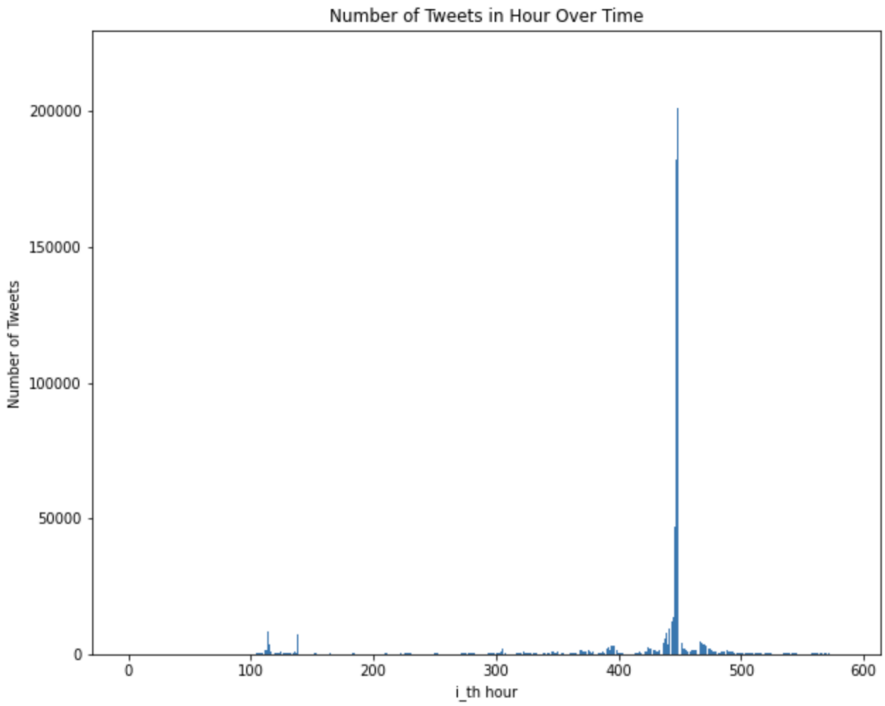
```
The Average Number of Tweets Per Hour is: 2072.11840170408
The Average Number of Followers of Users Posting the Tweets Per Tweet is: 8814.96799424623
The Average Number of Retweets Per Tweet is: 2.3911895819207736
```

Question 9.2:

“Number of tweets in hour” over time for #NFL



“Number of tweets in hour” over time for #SuperBowl



Question 10:

Task: Given a specific tweet posted during the game session, we can predict the team which is ahead on the scorecard.
Dataset: ECE219_tweet_data/tweets_#superbowl.txt

1. Data Selection:

In our designed task, our goal is to predict the winning team based on a given tweet. The first thing we will do is to select the tweets within the game session period.

1. First, we will create a pandas dataframe for the superbowl hashtag and convert 'citation_date' to datetime format based on LA time.
2. Then we will do a range selection to choose only tweets between Feb 1 15:00 to 19:10 local time in LA.
3. The last step will be doing a random sampling(50,000rows) on the data, because our original data is too large (400,000rows) which will be problematic for later training.

	citation_date	tweet	num_retweets	num_followers
1407	2015-02-01 15:00:01-08:00	It's almost time #SuperBowl	1	1307
1401	2015-02-01 15:00:01-08:00	#bedford #superbowl fans join us tommorrow mo...	2	3644
1403	2015-02-01 15:00:01-08:00	Who are YOU rooting for? #SuperBowl http://t.c...	1	43
1404	2015-02-01 15:00:01-08:00	Finally, it's time! .. #SuperBowl #SEAvsNE	1	127
1405	2015-02-01 15:00:01-08:00	The PERFECT news for #SuperBowl Sunday: http:/...	1	1456
1406	2015-02-01 15:00:01-08:00	#Superbowl starting soon. Should be a good one...	1	145
1408	2015-02-01 15:00:01-08:00	Bandwagon #SuperBowl	1	690
1400	2015-02-01 15:00:01-08:00	“@mforden2: A #SuperBowl ad that will chill yo...	4	18126
1409	2015-02-01 15:00:01-08:00	#SuperBowl Sunday means tons of parties with g...	1	73
1411	2015-02-01 15:00:01-08:00	Have a "SUPER" Day!\n#SuperBowl #SuperDooper h...	1	40
1412	2015-02-01 15:00:01-08:00	At Doctors Express we treat sprains to broken ...	1	330
1413	2015-02-01 15:00:01-08:00	Ball Deflaters vs. Ball Grabbers... NEEEXXXXT....	1	67

2. Time-Score Pairing:

Now that we have all the data available in the dataframe. We need to label the tweet for later model training. In our case, we label the tweet as 0 when there is a draw or no action at all, e.g: halftime show. We label it a 1 when the New England Patriots are winning and 2 when Seattle

Seahawks. We will be pairing the winning team one by one based on the information provided by http://superadsdb.com/2015/timeline.
Here is a random sample selection from the dataframe:

citation_date	tweet	score_lead
2015-02-01 17:29:18-08:00	Yep twas short 4 @LennyKravitz at the #SuperBo...	0
2015-02-01 19:08:45-08:00	Well that was one heck of a game! Couldn't bel...	1
2015-02-01 15:21:43-08:00	Go Patrihawks! #SuperBowl	0
2015-02-01 18:19:14-08:00	Black God knows the struggle! Haha I love it! ...	2
2015-02-01 16:41:48-08:00	Goats are always relevant! #superbowlcommercials	0
2015-02-01 17:35:36-08:00	Matthews on fire #SuperBowlXLIX #SEAvsNE	0

3. Text Cleaning:

With the tweets and their corresponding labels now stored in the DataFrame, we can begin the process of basic cleaning. This involves removing unnecessary elements such as punctuation, special notations, and emojis. Additionally, since many of the hashtags are likely to be duplicated across tweets, we have chosen to remove them entirely.
Here is an example of tweet vs clean_tweet:

tweet	clean_tweet	score_lead
Yep twas short 4 @LennyKravitz at the #SuperBo...	yep twas short 4 lennykravitz at the still che...	0
Well that was one heck of a game! Couldn't bel...	well that was one heck of a couldnt believe th...	1
Go Patrihawks! #SuperBowl	go patrihawks	0
Black God knows the struggle! Haha I love it! ...	black god knows the struggle haha i love it	2
Goats are always relevant! #superbowlcommercials	goats are always relevant	0
Matthews on fire #SuperBowlXLIX #SEAvsNE	matthews on fire seavsne	0
Well that just ruined my adrenaline buzz. #sup...	well that just ruined my adrenaline buzz fight	1

4. Lemmatization:

To improve our model training, we will perform lemmatization on our clean text. We have chosen lemmatization over stemming because it allows us to preserve the original vocabulary of the text while obtaining the essential base of each word.

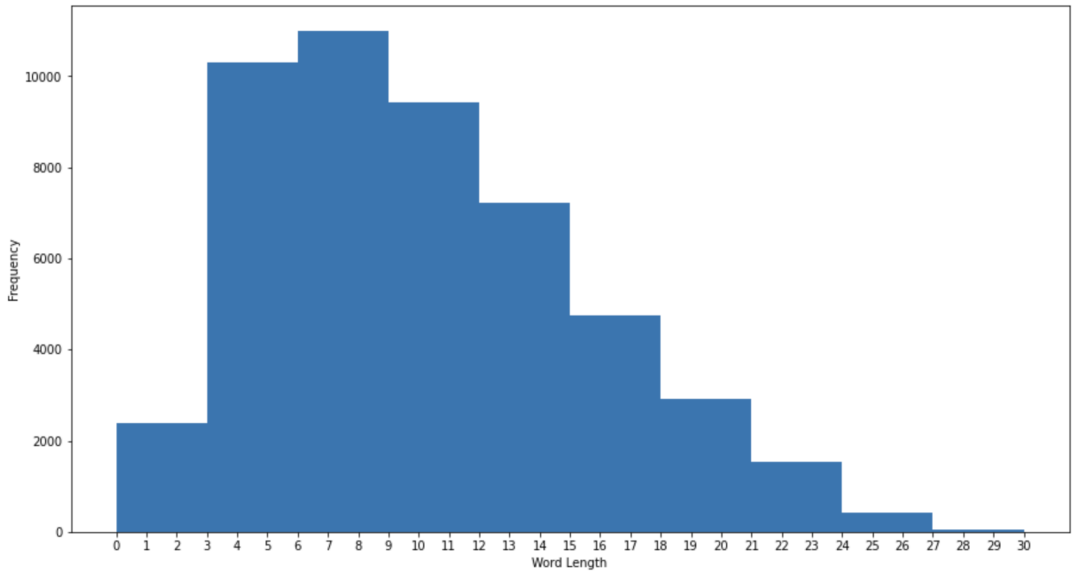
Here is an example of clean_tweet vs lemmatized_tweet:

clean_tweet	lemmatized_tweet
yep twas short 4 lennykravitz at the still che...	yep twas short 4 lennykravitz at the still che...
well that was one heck of a couldnt believe th...	well that be one heck of a couldnt believe tha...
go patrihawks	go patrihawks
black god knows the struggle haha i love it	black god know the struggle haha i love it
goats are always relevant	goat be always relevant
matthews on fire seavsne	matthew on fire seavsne
well that just ruined my adrenaline buzz fight	well that just ruin my adrenaline buzz fight

5. Text Exploration:

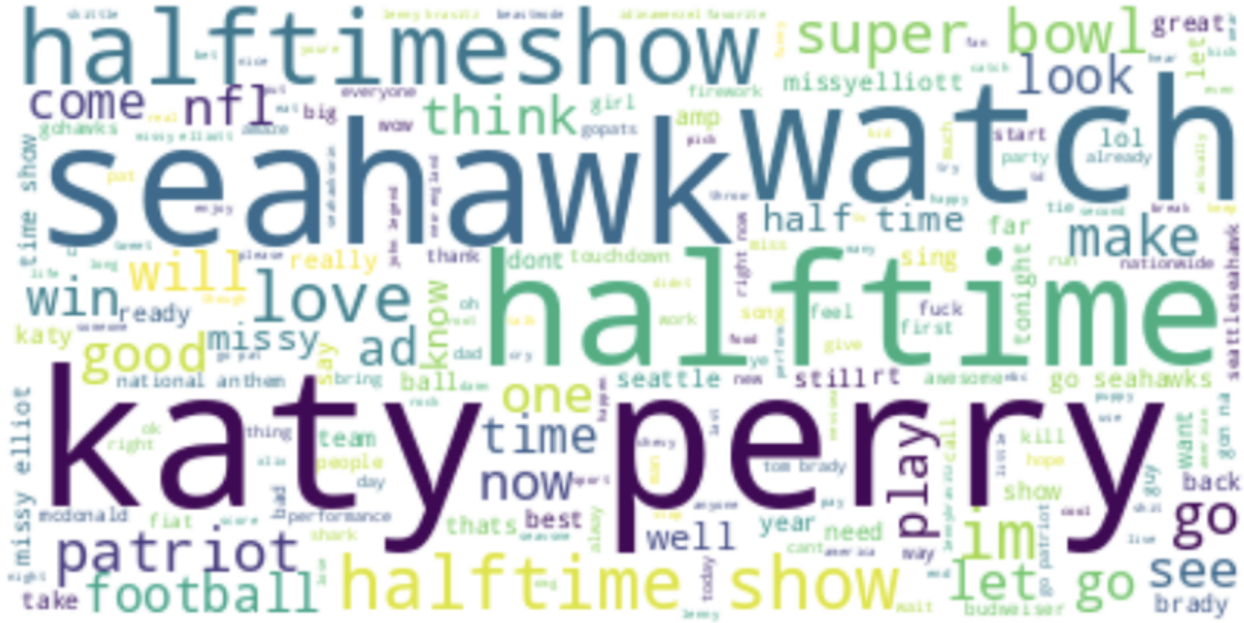
Now that we have the clean tweets, it's time to perform some exploratory data analysis (EDA) to extract useful information from them. First, we will create a histogram of text lengths. Then, we will use a word cloud to visualize the most frequent words used when a specific team wins. We will compare these results to our expectations.

1. The text length frequency of the 50,000 tweets: We observe that the majority of tweet lengths fall between 3-14 words, which could potentially limit the context available for later inference..



2. WordCloud for each label:

- For label = 0 (draw or no action at all): As expected, we can see that the most frequently occurring words are "Katy Perry" and "halftime," indicating that these tweets were posted during periods of draw or non-action.



- For label = 1 (New England Patriots is winning): This matches our expectation as well, as we can see the biggest words are patriot and seahawk(patriot bigger than seahawks), which means these tweets were posted between most of the NE winning period.

- For label = 2 (Seattle Seahawks is winning): This matches our expectation as well, as we can see the biggest words are seahawk and patriot (seahawks bigger than patriot), which means these tweets were posted between most of the SEA winning period.

6. TF-IDF:

To prepare our tweet texts for processing in our subsequent machine learning models and to ensure that the information contained in the texts does not interfere with the analysis, we will apply TF-IDF on the tweet texts. Finally, we have the feature dimensions of 26848.

```
print(X_train_tfidf.shape)
print(X_test_tfidf.shape)
```

```
(40000, 26848)
(10000, 26848)
```

7. Dimensionality Reduction:

Since we have too many dimensions ~26848 which will drastically increase our computational complexity and bring much unnecessary information, we will do LSI on our tf-idf matrix to ensure we get just about the information we need. In the following models, we will be using different parameters for different models.

8. Model Selection:

Within each of our actual training models, we have to deal with the imbalanced data inside our dataset. We choose to use oversampling in our case to ensure that we get a valid and meaningful accuracy score.

```
sb_df['score_lead'].value_counts()
```

```
0    28444
1    12238
2     9318
Name: score_lead, dtype: int64
```

Base Model - Decision Tree:

When we do the decision tree after oversampling the minority, there is not so much difference when we switch to different n_components. We have about an accuracy score of 0.51 and f1 score of 0.51 as well.

```
Decision Tree for multi stats when n_components: 100:
accuracy score:  0.5109
recall score:  0.5109
precision score:  0.512954469961944
f1 score:  0.511895448170657
----------------------------------------
Decision Tree for multi stats when n_components: 300:
accuracy score:  0.517
recall score:  0.517
precision score:  0.5196853072427499
f1 score:  0.5183070894660288
----------------------------------------
Decision Tree for multi stats when n_components: 1000:
accuracy score:  0.5039
recall score:  0.5039
precision score:  0.5066163912122135
f1 score:  0.5052231783898107
----------------------------------------
Decision Tree for multi stats when n_components: 2000:
accuracy score:  0.5027
recall score:  0.5027
precision score:  0.5044260820594492
f1 score:  0.5035095889439971
```

Second Model - SVM:

When we do the decision tree after oversampling the minority, the best n_components is 2000. We have an accuracy score about 0.6 and f1 score of 0.6 as well.

```
SVM for multi stats when n_components: 100:
accuracy score:  0.5068
recall score:  0.5068
precision score:  0.567052763138968 2
f1 score:  0.5251435343182018
----------------------------------------
SVM for multi stats when n_components: 300:
accuracy score:  0.5626
recall score:  0.5626
precision score:  0.6056322741054907
f1 score:  0.5764996817841699
----------------------------------------
SVM for multi stats when n_components: 1000:
accuracy score:  0.5835
recall score:  0.5835
precision score:  0.6217793958588897
f1 score:  0.595693379469902
----------------------------------------
SVM for multi stats when n_components: 2000:
accuracy score:  0.5928
recall score:  0.5928
precision score:  0.6276773397426676
f1 score:  0.6038711403070359
```

Third Model - CNN

The CNN model is suitable for classifying both text(Conv1D) and image data(Conv2D).

The best validation accuracy we get is 0.6215:

```
Epoch 1/30
69/69 [==============================] - 58s 831ms/step - loss: 1.1499 - accuracy: 0.3959 - val_loss: 1.1490 - val_ac
curacy: 0.3570
Epoch 2/30
69/69 [==============================] - 58s 834ms/step - loss: 1.1328 - accuracy: 0.4879 - val_loss: 1.1367 - val_ac
curacy: 0.4070
Epoch 3/30
69/69 [==============================] - 57s 831ms/step - loss: 1.1096 - accuracy: 0.5430 - val_loss: 1.1115 - val_ac
curacy: 0.4802
Epoch 4/30
69/69 [==============================] - 57s 821ms/step - loss: 1.0771 - accuracy: 0.5835 - val_loss: 1.0767 - val_ac
curacy: 0.5262
Epoch 5/30
69/69 [==============================] - 57s 828ms/step - loss: 1.0394 - accuracy: 0.6109 - val_loss: 1.0458 - val_ac
curacy: 0.5450
Epoch 6/30
69/69 [==============================] - 57s 826ms/step - loss: 0.9961 - accuracy: 0.6381 - val_loss: 1.0145 - val_ac
curacy: 0.5632
Epoch 7/30
69/69 [==============================] - 57s 825ms/step - loss: 0.9511 - accuracy: 0.6623 - val_loss: 0.9823 - val_ac
curacy: 0.5855
Epoch 8/30
69/69 [==============================] - 57s 827ms/step - loss: 0.9036 - accuracy: 0.6834 - val_loss: 0.9559 - val_ac
curacy: 0.5987
Epoch 9/30
69/69 [==============================] - 56s 818ms/step - loss: 0.8563 - accuracy: 0.7043 - val_loss: 0.9320 - val_ac
curacy: 0.6114
Epoch 10/30
69/69 [==============================] - 58s 842ms/step - loss: 0.8089 - accuracy: 0.7229 - val_loss: 0.9155 - val_ac
curacy: 0.6159
Epoch 11/30
69/69 [==============================] - 59s 850ms/step - loss: 0.7661 - accuracy: 0.7382 - val_loss: 0.9048 - val_ac
curacy: 0.6189
Epoch 12/30
69/69 [==============================] - 57s 831ms/step - loss: 0.7264 - accuracy: 0.7526 - val_loss: 0.8958 - val_ac
curacy: 0.6211
Epoch 13/30
69/69 [==============================] - 59s 863ms/step - loss: 0.6902 - accuracy: 0.7671 - val_loss: 0.8949 - val_ac
curacy: 0.6178
Epoch 14/30
69/69 [==============================] - 62s 892ms/step - loss: 0.6591 - accuracy: 0.7789 - val_loss: 0.8930 - val_ac
curacy: 0.6187
Epoch 15/30
69/69 [==============================] - 57s 823ms/step - loss: 0.6304 - accuracy: 0.7904 - val_loss: 0.9024 - val_ac
curacy: 0.6114
Epoch 16/30
69/69 [==============================] - 56s 809ms/step - loss: 0.6055 - accuracy: 0.7996 - val_loss: 0.8937 - val_ac
curacy: 0.6215
```

9. Limitation and Summary:

We achieved an accuracy and f1 score of about 0.62, which is not perfect compared to most ML models. Despite trying different tweet cleaning methods and dimensionality reduction techniques, we were unable to achieve any discernible improvement. After careful consideration, we have identified several limitations that may be contributing to this.

Firstly, our reliance on website information for real time winning team pairing may lead to discrepancies since the website itself is not easy to read. We might have some tiny mismatch in our dataframe. This could result in inaccuracies in the data we collected. Furthermore, it is worth noting that some Twitter users may delay posting their reactions until after some time has passed since the game. This delay could potentially result in their tweets not accurately reflecting the real-time situation.

Secondly, the limited length of tweet text - which we also further reduced by removing hashtags. This will cause less information getting processed through the model training period, making it harder for the model to learn, as most tweets are only between 3 to 14 words.

Lastly, despite the tweet length, the tweets themselves may not provide useful information. Many of them are purely emotional, making it difficult even for humans to determine whether NE or SEA is winning. This makes it challenging for the model to extract the essential idea from the tweet.

Despite these challenges, we were able to achieve a relatively decent accuracy score of about 60%, which outperforms random guessing(3 classes for 33%). However, to further improve the model's performance, we may need to consider including hashtags that could contain more information and stronger personal preferences. Nonetheless, this approach does not align with our original goal and is less challenging for us.