Project 4: Regression Analysis and Define Your Own Task!

Large-Scale Data Mining: Models and Algorithms

Ananya Deepak Deoghare, Hariram Veeramani, Madhav Sankar Krishnakumar

005627628, 005528336, 405692669

ECE 219 Winter 2022

Assumptions:
Initial assumptions for the answers presented in the rest of the report

- For the Gas Turbine Emission Dataset, we primarily considered CO as our primary target variable, hence all our answers and reasoning are on the basis of the predictions related to the 'CO' target variable.
- After initial data inspection questions, we have standardized the features and target variable for all other models.
- We have tried numerical and one-hot embedding for 'year' feature and have found the latter to be better. Hence, we will use the latter for all the models, unless specified otherwise.
- The RMSE scores in the results are negative as scikit has implemented neg_rmse, which is nothing but the negative of the RMSE. This is done to ensure that higher the value, better the model. We have tried to display scores as positive RMSE in the report, though code might return negative.

Question 1

Standardize feature columns and prepare them for training
We employ standardization to reduce the mean of the features involved in dataset to zero and reduce the variance of the features to unit variance. We have standardized the feature columns and the target.

Standardization of the features in any dataset is inevitable for the proper utilization of many machine learning algorithms. This is because the features involved in many of the machine learning estimator algorithms such as the RBF Kernel in SVM , L1/L2 Regularization of linear models are centered around zero and also have their variance around the same order. This is needed because some of the features in our dataset may have higher variances compared to the variances of other features in our dataset and this odd value of high variance could dominate the objective function and mask the learning algorithm/estimator to learn from other features as correctly as expected.

It is worth mentioning the significance of the metrics associated with the diamond dataset:

- **Price** - price in US dollars
- **Carat** - Weight of Diamond
- **Cut** - Quality of the Diamond (Quality arranged in increasing order- Fair, Good, Very Good, Premium, Ideal)
- **Color** - J(worst) to D(best)
- **Clarity** - a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
- **X,Y,Z** - Dimensions measured in units of millimeters (mm).
- **Depth** - Total Depth Percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)
- **Table** - Width of Top of Diamond relative to Widest Point (43–95)

It is also worth elaborating the abbreviations related to the Gas Emission Measurement Dataset given by:

- AT - Ambient Temperature
- AP - Ambient Pressure
- AH - Ambient Humidity
- AFDP - Air Filter Differene Pressure
- GTEP - Gas Turbine Exhaust Pressure
- TIT - Turbine Inlet Temperature
- TAT - Turbine After Temperature
- CDP - Compressor Discharge Pressure
- TEY - Turbine Energy Yield

Question 2

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 each dataset, describe what this high correlation suggests.

Heatmap of the Pearson Correlation Matrix of Diamond dataset is displayed below:

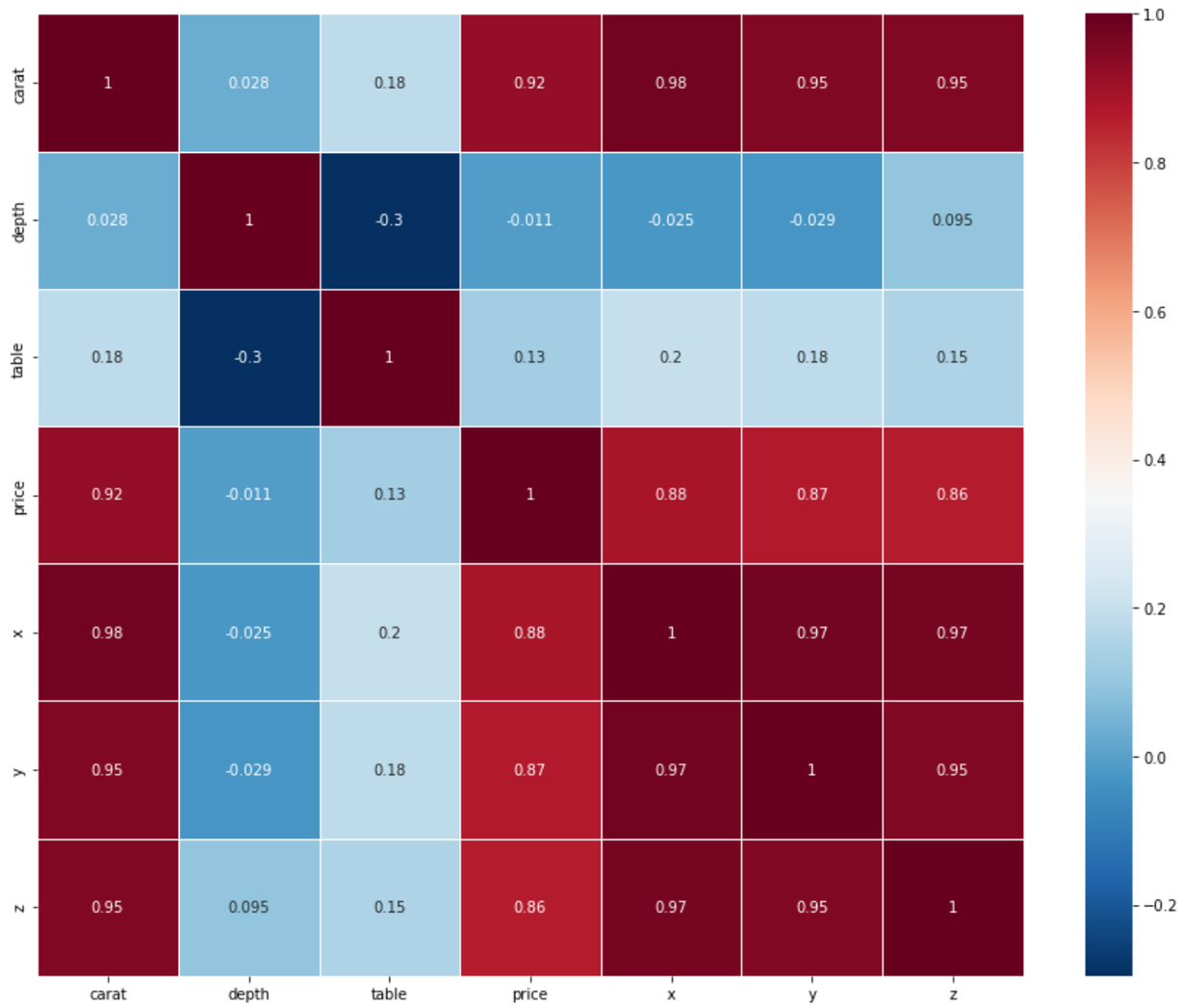


Figure 1: Pearson Correlation Matrix - Heatmap- Diamond Dataset

From the Heatmap, we could observe that the following features show a high degree of positive correlation with the target variable:

- carat - weight of the diamond
- x - length in mm
- y - width in mm
- z - depth in mm

Implication:

- The High correlation of price with carats makes sense because pure diamond is always more expensive than the comparatively less pure ones.

- Also, as the dimensions of the diamond scale up, the weight increases which implies an increased amount/quantity of diamond leading to a higher price.
- The low correlation of depth and table is understandable as these do not directly determine the prices.

Heatmap of the Pearson Correlation Matrix of Gas Emission dataset is displayed below:

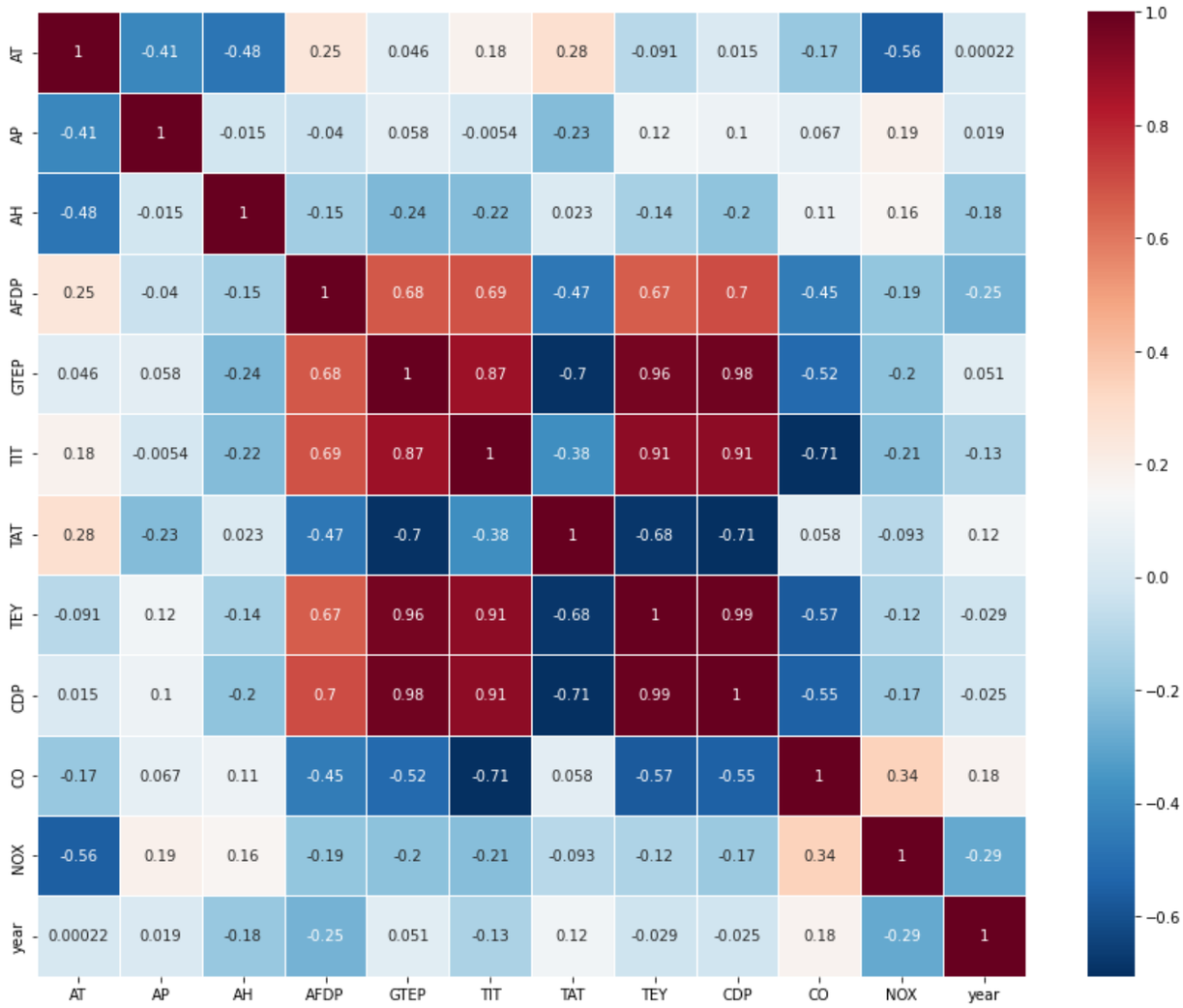


Figure 2: Pearson Correlation Matrix- Heatmap- Gas Emission Dataset

From the Heatmap, we could observe that the following features show a high degree of positive correlation with the target variable (CO):

- TIT - Turbine Inlet Temperature
- TEY - Turbine Energy Yield
- CDP - Compressor Discharge Pressure

Implication:

- The strength of absolute magnitudes of correlation in the Gas Emission dataset is on the lower side when compared with that of the diamond dataset. This may be attributed to the fact that amount of correlation of gas emission has widespread, complex hidden dependencies distributed among so many factors and is therefore not as straightforward as the succinct features which do a good job in characterizing the diamonds.
- It is interesting to note that all the features which are closely correlated with the CO target variable in the Gas Emission Dataset are associated with Turbine measurements and related characterizations.

- We can also note that all the features demonstrate a negative correlation based relationship with the target variable of interest. This means that as the target variable's value increases, the other variable's value is bound to decrease.

- The Negative Correlation of TIT (Turbine Inlet Temperature) with the Gas Emissions could be interpreted in different ways. We know that increasing gas emissions would lead to the phenomenon of Global Warming. Though the Global warming, as such, is expected to elevate the environmental temperature, it is also said that the increased temperature causes the world's frozen ice reserves to melt and serve as reservoirs which then again decreases the Environmental Temperature and hence Turbine Inlet temperature due to the energy transfer in this melting process. Hence, it may be interpreted that increase in quantity of gas emissions might lead to decrease in temperature as long as we are left with enough Ice reserves to melt down and keep the temperature drowning.

- The feature 'Turbine Energy Yield' showing a similar negative correlation with green house gas emission measurements is also intuitive because of the exponential increase in the amount of vehicles causing emission (two wheelers, cars, other means of transport) so on, and the exponential increase in the field of Deep Learning Research and hardware accelerators - Compute Power(GPUs, CPUs, TPUs) which lead to considerable amount of green house emission. This increased amount of emission decreases the Turbine's efficiency or capacity to generate power which is referred to as the yield of the turbine for the input wind energy over a pre-defined period of time. Hence, in presence of undesired Gas Emissions, more input wind energy is required to generate an equivalent amount of useful power(Electric) than what would have been required in an otherwise normal system operating in an emission free environment.

- The Compress Discharge Pressure (also referred to as Head pressure) is the output pressure generated on a Turbine system. This is affected by various factors such as size, speed, cleanliness of the comprising components and size of the entire connecting pipeline. Provided this, the Negative Correlation of CDP with Gas Emission becomes intuitive because, increased amount of gas emission can lead to excessive pollution and spoil the cleanliness of the Turbine pipeline. This will in turn reduce the CDP Pressure generated during the functioning of the Turbines. This can also be visualized as one of the reasons for the reduced amount of Turbine Energy yield, reinforcing our previous assertion.

The following the are Pearson's correlations after converting categorical features and without standardizing the target variables.

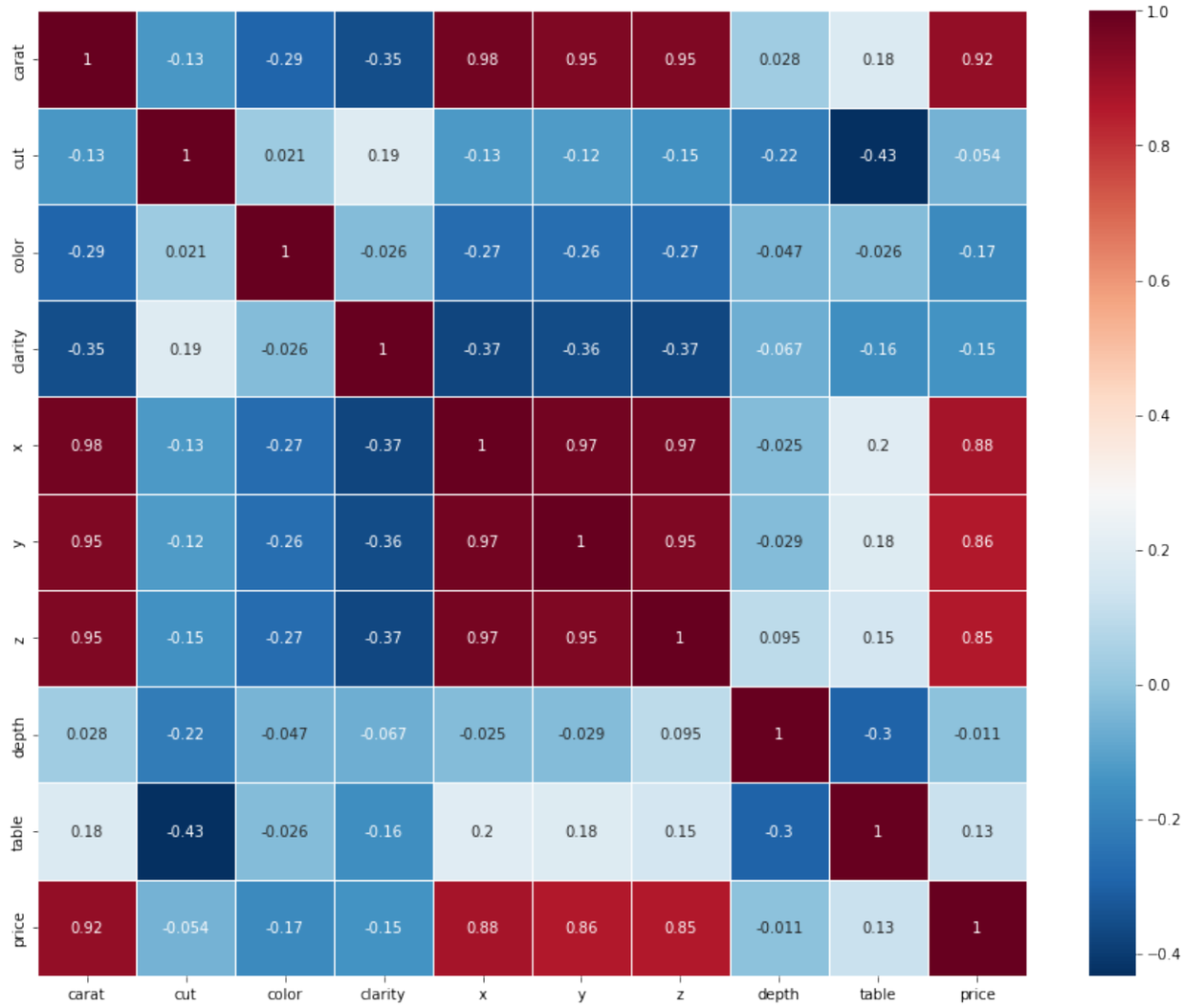


Figure 3: Pearson Correlation Matrix- Heatmap - Diamond Dataset

Features: Carat, x, y, z.

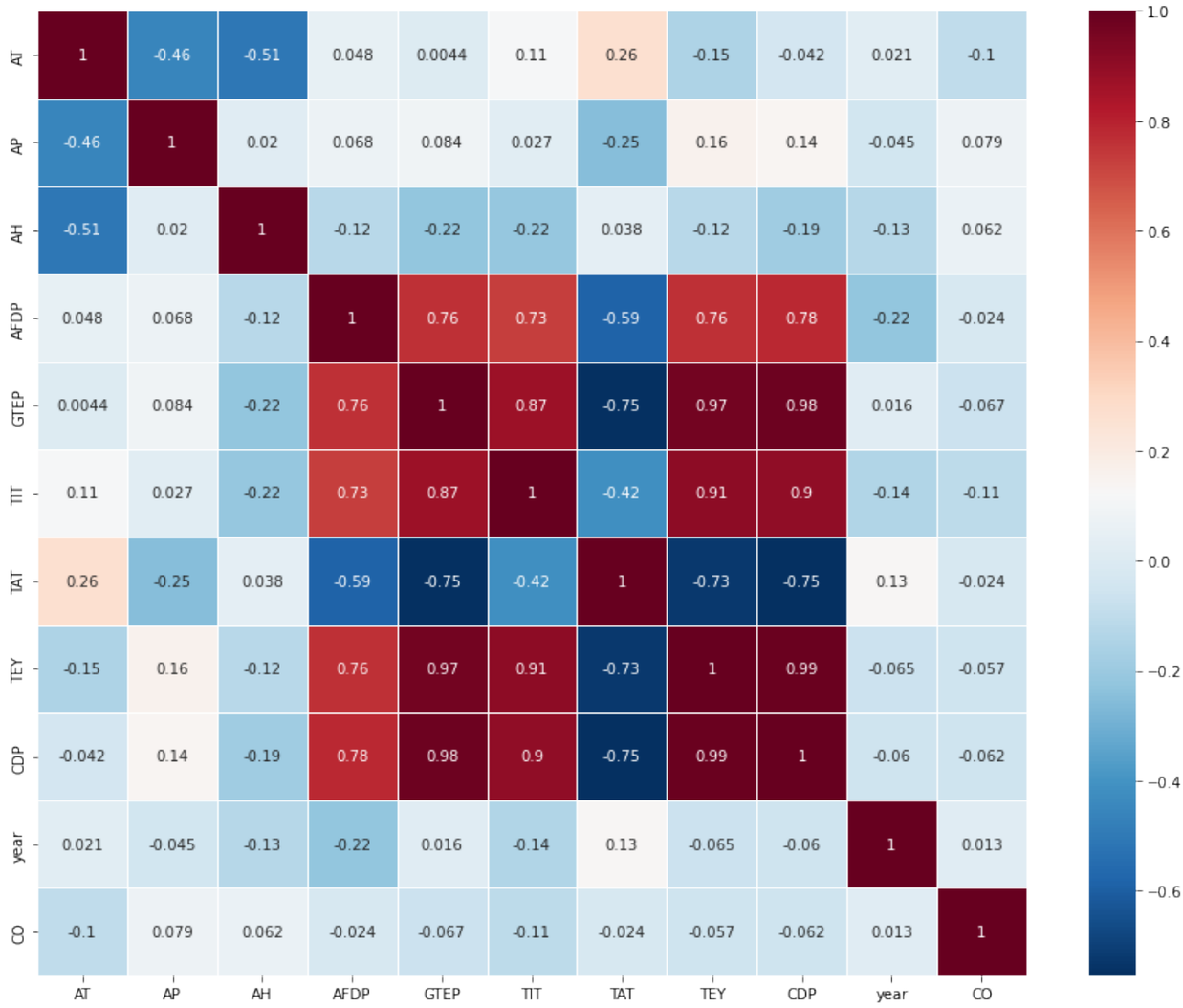


Figure 4: Pearson Correlation Matrix- Heatmap - Gas Emission Dataset

Features: TIT, AT.

Question 3

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

Histograms of Numerical Features of Diamond Dataset

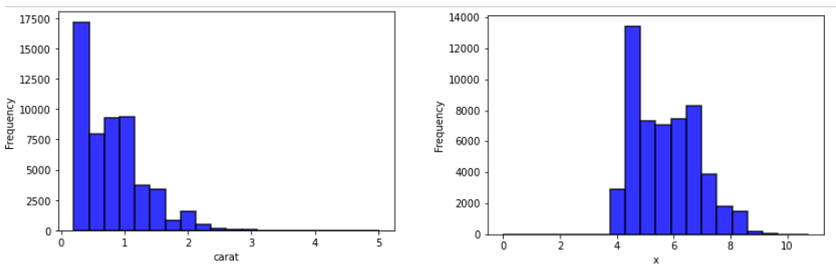


Figure 5: Histogram of Features- Carat, X(dimension)- Diamond Dataset

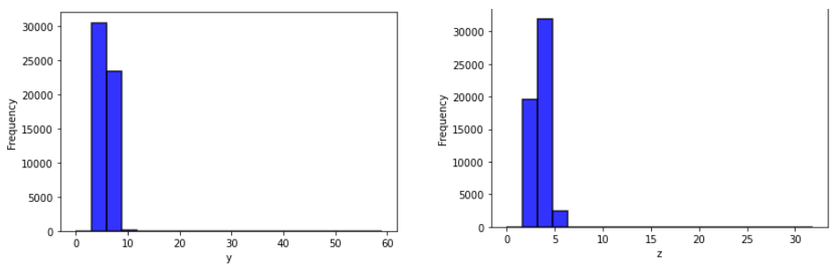


Figure 6: Histogram of Features- Y, Z Dimensions- Diamond Dataset

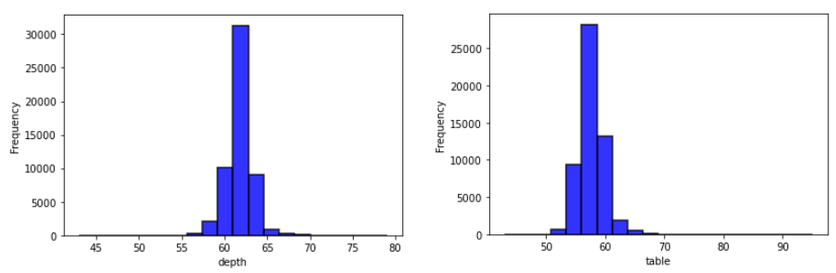


Figure 7: Histogram of Features- Depth, Table- Diamond Dataset

Observations from the Histograms corresponding to Diamond Dataset

- From the above mentioned set of histograms, we could observe that the histograms corresponding to the X dimension, Table and Depth Features of the Diamond dataset seem to be uniformly centered around a mean and seem to approximately resemble the non-zero mean Gaussian Distribution
- It is also interesting to note that Y and Z dimensions of diamonds are crafted more carefully by constraining their dimensional values to lie within a very narrow range while allowing variations in dimensions only along the X-plane (length of diamonds).
- The Y and Z dimensions hence exhibit more like a impulsive or dirac kind of distribution.

- This statistical variations can be attributed to the constraints associated with either manufacturing process pipeline or degrees of freedom associated with the instruments involved in the manufacturing process.
- The Numerical Feature- Carat table are more skewed towards the left direction.
- The Carat Feature's histogram being left skewed clearly implies that we could find more diamonds which belong to the lower end category (being less pure) when compared with highly pure diamonds. This fact might be dictated by the Diamond's role in the market established by the Supply, Demand Chain. That is, the purchasing power of people might be more strongly associated with diamonds of lower range (lower carats) than the higher range/high tier diamonds necessitating the manufacturing of increased quantities of diamonds taking on such carat values in the lower range.

Histograms of Numerical Features of Gas Turbine Emission Dataset

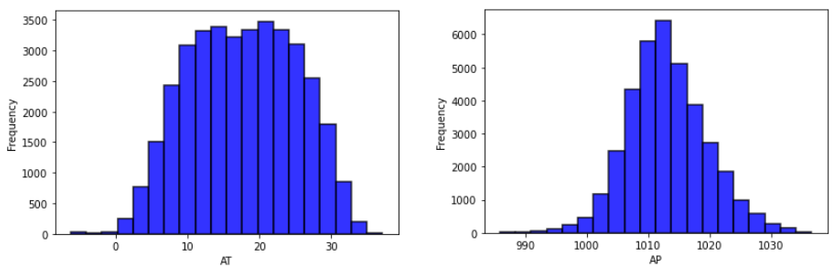


Figure 8: Histogram of Numerical Features- AT, AP- Gas Emission Dataset

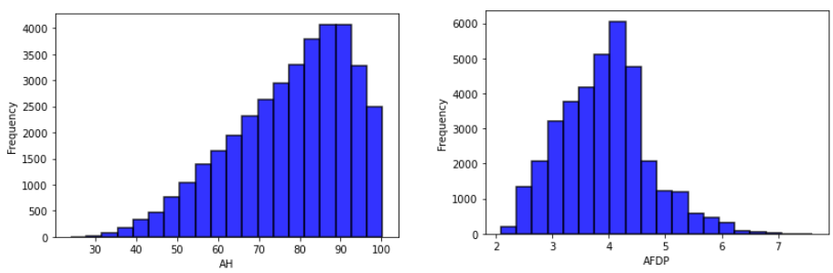


Figure 9: Histogram of Numerical Features- AH, AFDP- Gas Emission Dataset

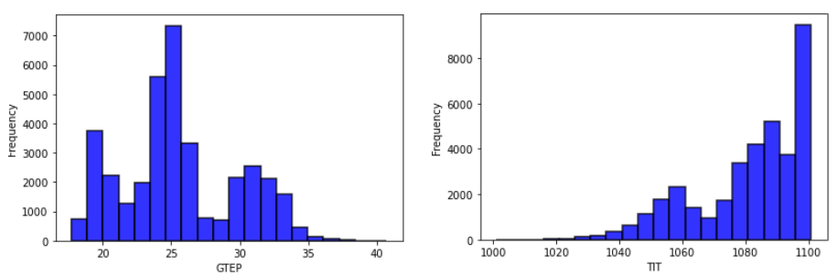


Figure 10: Histogram of Numerical Features- GTEP, TIT- Gas Emission Dataset

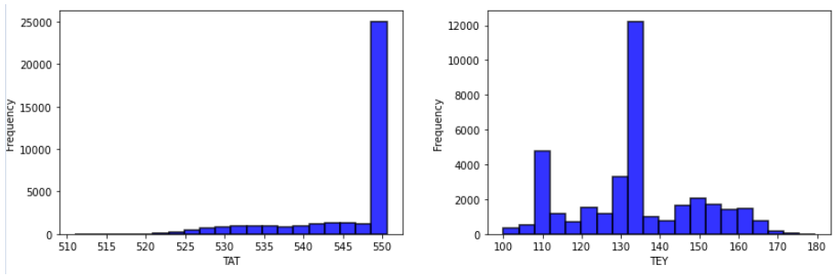


Figure 11: Histogram of Numerical Features- TAT,TEY - Gas Emission Dataset

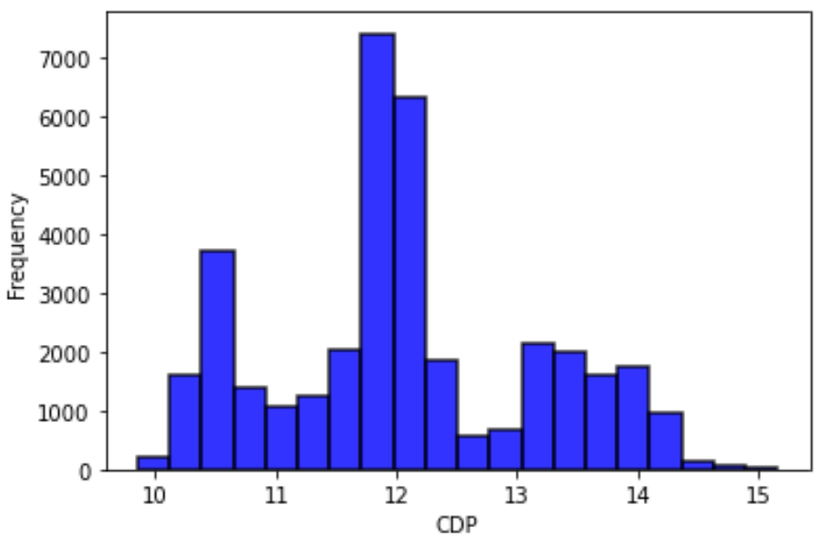


Figure 12: Histogram of Numerical Feature- CDP- Gas Emission Dataset

Observations from the Histograms corresponding to Gas Emission Dataset

- We Observe that the first two numerical features- Ambient Temperature and Ambient Pressure are almost symmetric and exhibit a distribution which is close to non-zero mean Gaussian Distribution.
- The remaining four features- Ambient Humidity, AFDP, GTEP, TIT and TEY exhibit skewness in their distributions. The AH(Ambient Humidity) and TIT (Turbine Inlet Temperature) Take on right skewed distributions, while the AFDP feature is skewed more towards the left.
- The TAT feature is clearly inferred to be taking on closely spaced values within a very narrow dynamic range.
- Two of these features are positively skewed while the other two features are negatively skewed

Such skewed features may not be correctly interpreted by the ML models since most of the ML models are designed with the inherent assumption that target component of the input features have a normal statistical distribution.

While common estimation algorithms like Least Squares regression and Mean square require the mean to point to central tendency in the skewed case- First order moment- mean might not be a good representative of the distribution of the features

Solution

- As suggested in the first step, we can standardize the data to mean 0 and variance 1. This will help remove some skewness.
- We can perform different type of transformations on the skewed features depending on the nature of the skew- Positively skewed vs Negatively skewed
- If the features are positively skewed, we can utilize transformations in the increasing order of strengths such as:
  - Square Root Transformation
  - Cube Root Transformation
  - Natural Log Transformation(Except 0 inputs)
  - Log Base 10 Transformation
  - Inverse Transformation
  - Removing and mitigating outlier candidates
- What these transformations essentially do is that they act on different values present in the skewed segment and offer shrinking and compressing transformations on the data points depending on the magnitude which essentially makes the features look more centered around a defined value and thereby reduces skewness.
- if the features are negatively skewed(left tail), we can incorporate the following transformations in the increasing order of correction capability:
  - Feature reflection - Square Root Transformation
  - Feature reflection - Natural Log Transformation
  - Feature reflection - Log base 10 Transformation
- Depending upon the degree of skewness present in the features, we can utilize use any of the schemes in the order of their transformation or correction capability.

Question 4

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

Box plot

- The Box plot is an efficient and compact way to capture the information present in the distribution of various feature datasets.
- The Coloured middle box represents the interquartile range which consists of datapoints lying from the 25th percentile to the 75th percentile in the dataset.
- The Extremes represent the Maximum and Minimum quantities which are mathematically determined by the interquartile range along with the right and left boundaries of the interquartile range respectively.

Figures 11 and 12 show the boxplot of Categorical features vs target variable in the Diamond Dataset.

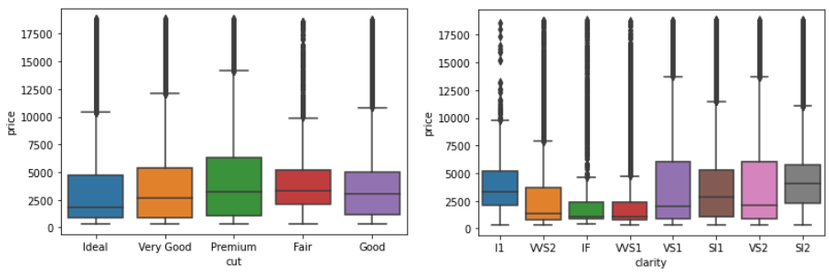


Figure 13: Boxplot of Diamond Price Vs Cut and Clarity

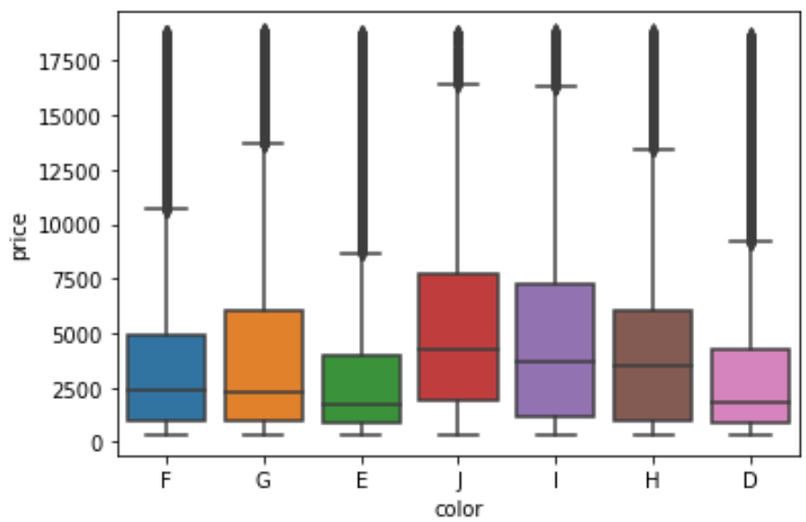


Figure 14: Boxplot of Diamond Price Vs Color

Observations for Diamond Dataset:

- In the Cut feature, the median of premium cut is the highest. This is surprising considering the fact that Ideal is the best cut. This could be related to other factors playing a role. The premium cut might have had higher carat or clarity, for instance.

- Even in case of clarity, we see that IL diamonds have much higher prices than IF based diamonds, while in reality we might expect more of a converse relationship. Same is the case for color. Therefore, all of cut, color and clarity may not be salient features.
- In the Cut vs Price Box Plot (Figure 11- Left), First we observe that Premium cut type diamonds have the least amount of outliers, with a very distinguishing minimum and maximum segments making it more predictable in addition to this category accounting for the majority in the dataset.
- Very Good and Good cateogry of Diamonds have a relatively higher number of outliers with the prices overlapping with the lower range of prices of the premium category.
- We also observe that Ideal, Premium and Fair Cut Diamonds look very skewed about the median, whereas the Very Good and Good Cut Diamonds are almost symmetric about their median.
- From the Clarity vs Price plot (Figure 11- Right), we can clearly point that diamonds with 'VS1' and 'VS2' clarity are more easily predicatable by the model, we can also conclude that 'l1', 'WS2', 'IF/WS1' clarity categories relate to clear distinguishing set of prices.
- 'IF' and 'WS1' categories, in addition to having the largest number of outliers, have their prices skewed almost entirely to the right of the median Price.
- The diamonds with 'S12' clarity have almost their entire price range overlapping with the price range of (VS1) but with a comparatively increased number of outliers, also they are symmetric about their median price value.
- Whereas, the 'VS1', 'VS2' clarity categories are almost indistinguishable closely resembling each other in all metrics such as Median, Minimum, Maximum.
- In case of color, we see that colours J and I have very low outliers and this is not the case for the rest.
- We also observe that G, E and D are highly skewed, whereas I, J and H are fairly uniform about the median.

Figure 13 shows the boxplot of Categorical features vs target variable in the GT Emission Dataset:

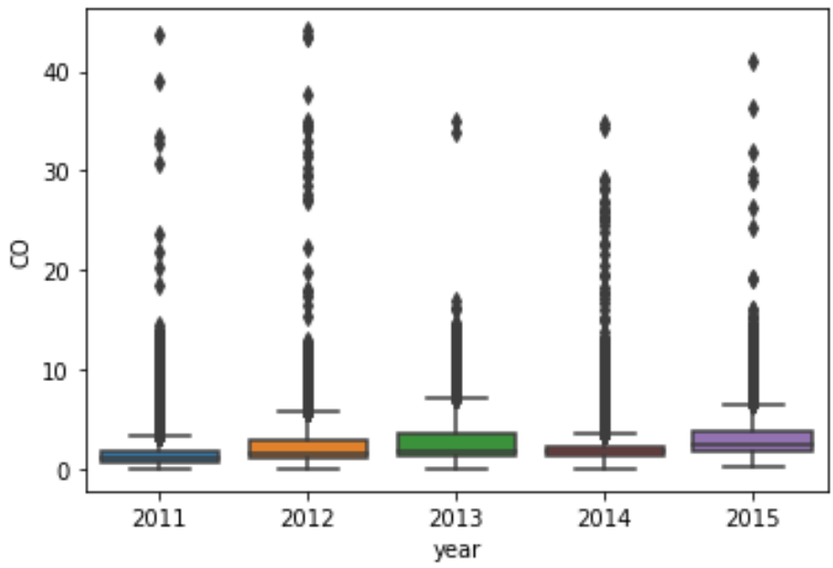


Figure 15: Boxplot of CO Emission Vs Year

Observations for Gas Emission Dataset:

- This Box plot clearly shows how year of 2013 stands apart from the other years, which has the fewest number of outliers comparably while also having contributed to the maximal value of emission with maximal dynamic range of emission values, clearly making it the most predictable one.
- Overall, we see that the years of 2012 and 2015, have distributions which remotely resemble 2013 but with a far higher number of outliers and a compressed predictable region.
- The years of 2011 and 2014 could potentially be the least predictable ones among all the years with very high number of outliers.
- Overall, almost all years have a compressed predictable region and a lot of outliers. Therefore, none of the years may be salient features for prediction of prices. Also, the median prices of 2014, is not higher than, say 2013. Therefore, using year as a numerical feature may not be the right idea. Using them as one-hot representation would be a better option.

Question 5

For the Diamonds dataset, plot the counts by color, cut and clarity.
Following figures show the plot of the counts of different colors, cut and clarity in the Diamond dataset.

Observation

- We can see a uniform, non-increasing pattern in the counts vs color plot which denotes that higher production of diamonds based on certain colors.
- Similarly, in Cuts vs counts plot, we note that the the ideal cut diamond accounts for the majority in the dataset.
- This is followed by Premium Cut category of diamonds which are then followed by Good Cut, Very Good Cut and Fairly Cut Diamonds.
- The Fact that 'ideal cut' accounts for the most more than premium and that 'good cut' accounts for a larger part than very good cut category of diamonds unveil that it is a better market trade-off to have more of Good to Moderate quality of Cut Diamonds at a Moderate price than more expensive priced Premium Diamonds.

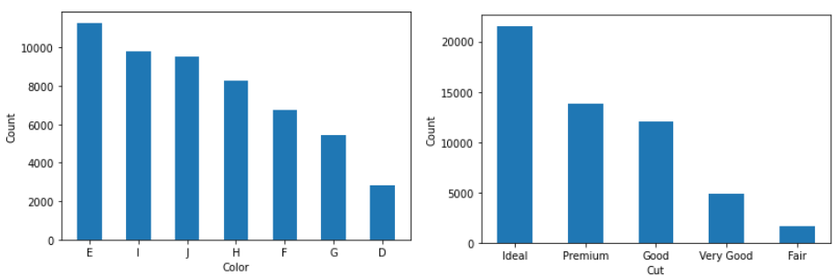


Figure 16: Plots showing Color vs Count on left, Cut vs Count on Right

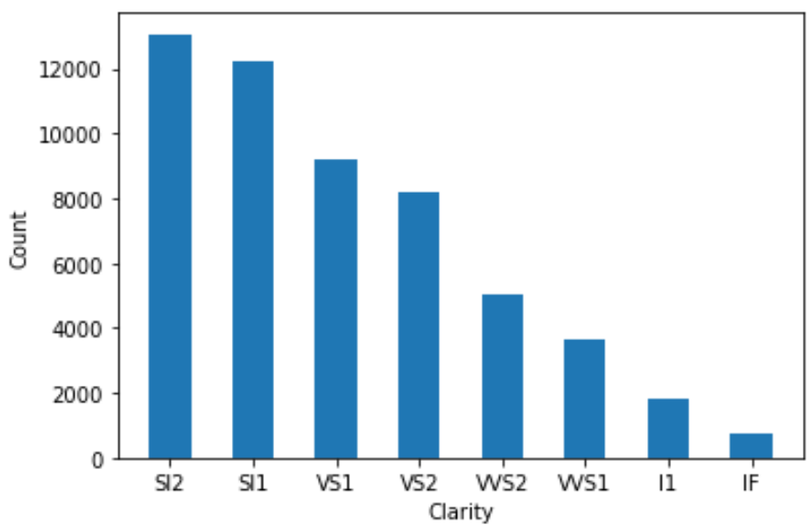


Figure 17: Plot showing Clarity cs Count

Question 6

For the Gas Emission dataset, plot the yearly trends for each feature and compare them. The data points don't have timestamps but you may assume the indices are times.

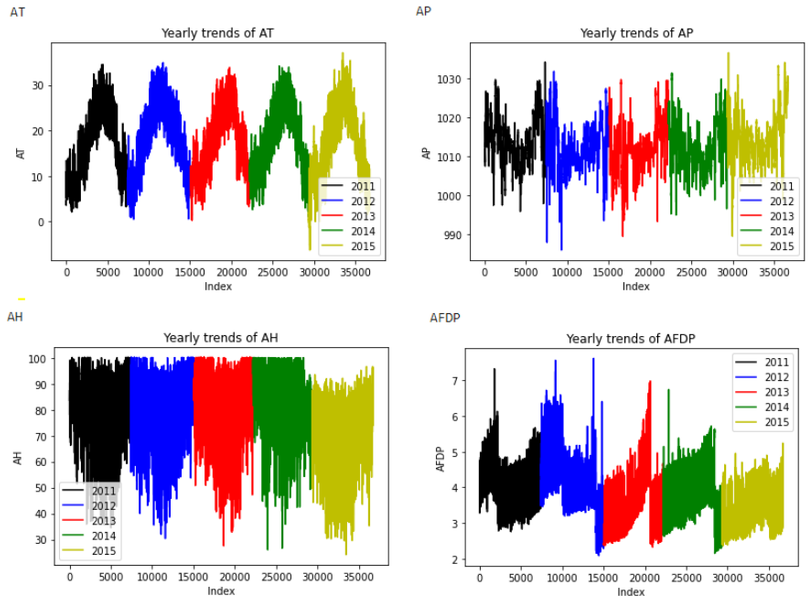


Figure 18: Plots showing: Yearly Trends of AT, AP, AH, AFDP

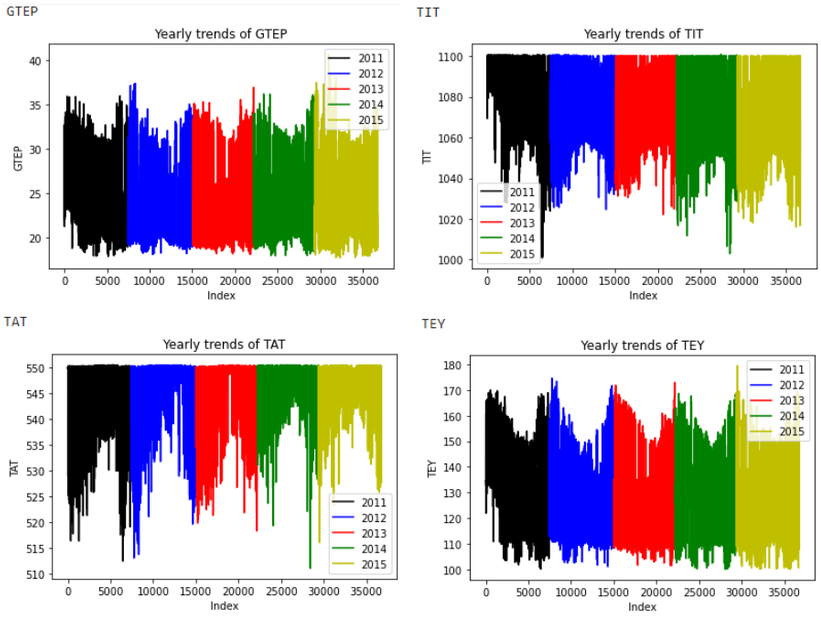


Figure 19: Plots showing Yearly Trends of GTEP,TIT,TAT,TEY

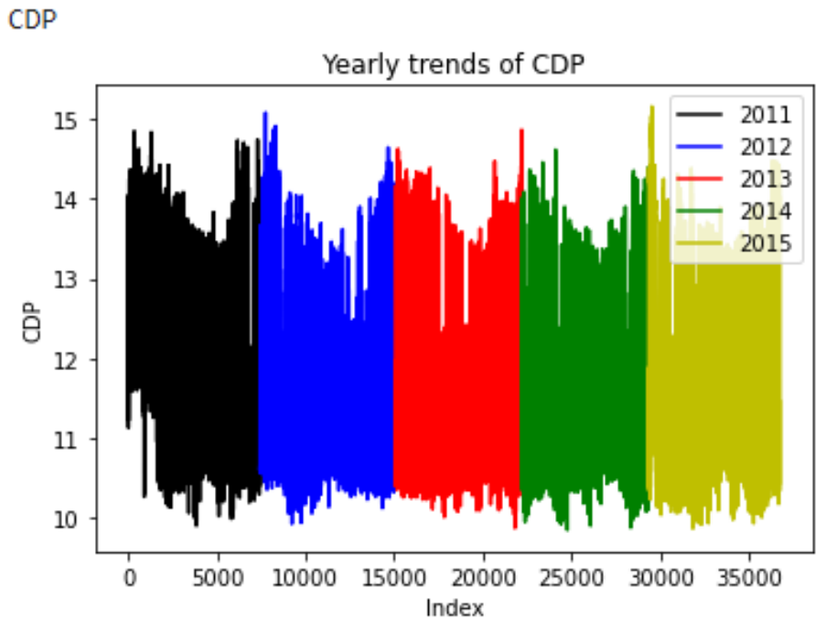


Figure 20: Yearly Trends of CDP

Observations from yearwise trends of the Gas Emission Dataset:

- By looking at the individual gas emisson trends across years, we can observe some common patterns.
- The Gas sensor measurements such as GTEP, TEY and CDP show very similar variations for the trends within an individual year, as well as the trends across all the years, though all of them span different range of values across the distinct individual years.
- On the contrary, we can observe the inverse of the previous trends,for the gas measurements associated with set comprising of AH, TIT and TAT. To be more precise, while the previous set of emission measurements had a behaviour, wherein the emission magnitude decreased towards the middle of every individual year and then increased back again to the higher emission values towards the end of the year, the current set of gas measurements consisting of AH, TIT and TAT exhibited higher values around the starting and endpoints of individual years and had assumed lower values around the midpoint of each years.
- This intuitively highlights the fact that while the emission of GT gases as a whole is required to be capped at the minimum value possible, the constituent gas measurements which characterize the holistic environmental impacts due to gas emission do not necessarily follow that the minimalastic objective. That is, emission of some constituent gases at a higher level is tolerable provided the other set of gases assume a proportionate decrease in their respective emission.

Question 7

You may use these functions to select most important features. How does this step affect the performance of your models in terms of test RMSE? Briefly describe your reasoning.

- Here, we are asked to Perform Feature selection and understand the impact of Feature Selection on the Model's Performance.
- Feature Selection is performed by invoking the SelectKBest function() utility to select the best 'k' number of features in the range of the 1 to N- Max number of features in the dataset.
- We measure the performance of the model using the Negative Test Root Mean Square Metric denoted by (RMSE) and the desired model should ideally make this RMSE value as much positive as possible.
- We analyze the impact of Feature selection using both the given datasets.

Mutual Information - Background and Significance

- MI which stands for Mutual Information captures the dependency between two distinct features.
- MI is essentially, arrived at by using Expectation operator on the pointwise version of the Mutual Information between two variables.
- The Mutual Information so obtained would be very close to zero, when the underlying two features are independent and would be more positive when the input features have some kind of interdependencies.

F- Score - Background and Significance

- F-Score is used to calculate an estimate of the linear dependency between the Entropy of two random variables.
- F-scores are obtained by performing F-tests which are linear regression tests (univariate).
- The purpose of F-score is to evaluate the improvement in the model upon the addition of new variables into the system under consideration.

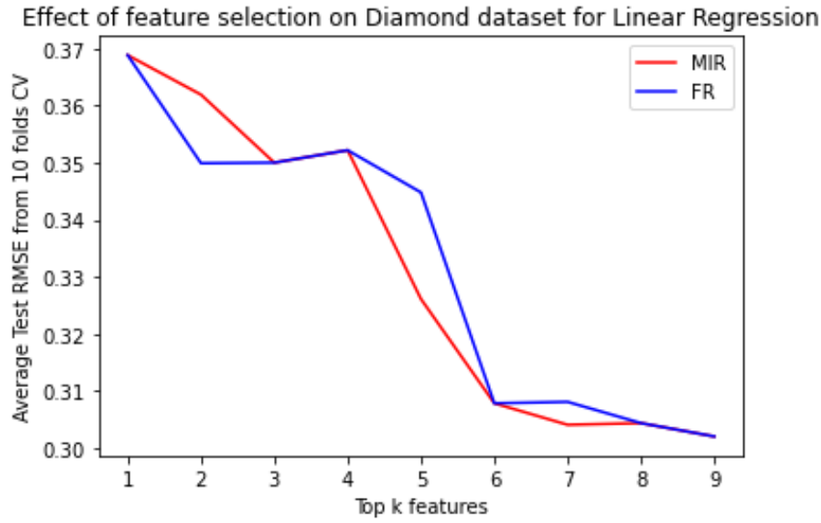


Figure 21: Effect of Feature Selection on Diamond Dataset- Linear Regression

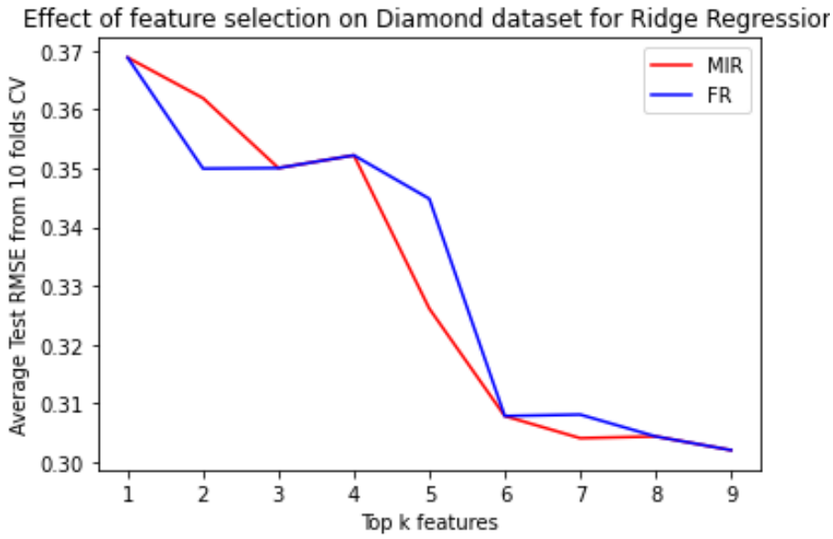


Figure 22: Effect of Feature Selection on Diamond Dataset- Ridge Regression

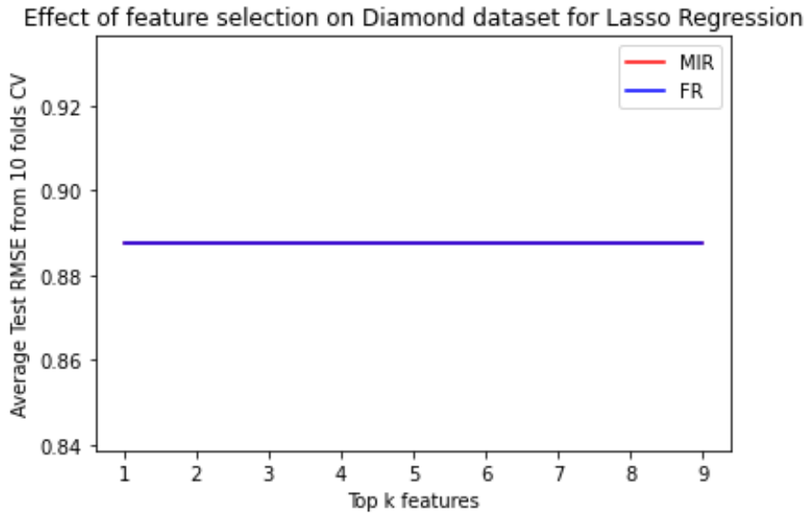


Figure 23: Effect of Feature Selection on Diamond Dataset- Lasso Regression

Effect of Feature Selection on Diamond Dataset

- Analyzing the feature selection plots obtained by using the three different regression techniques, we can draw some common patterns.
- First thing is we can see that as we increase the number of features being considered, the RMSE value clearly drops and becomes more optimum.
- As we keep increasing number of features from 0 to 6, the RMSE value decreases drastically, whereas once the number of features, K crosses the value of 6, the incremental amount of RMSE improvement for every additional feature is very low, this indicates the saturation behaviour,
- This saturation in RMSE might also mean that we are approaching the overfitting condition.
- This may also indicate that this problem can be solved efficiently with a finite number of feature dimensions around K=6,7 or very close to these values.
- We observe that very similar Feature selection vs RMSE characteristics for both Linear Regression and Ridge Regtession.
- Whereas for the Lasso Regression, we observe that the RMSE metric clearly remains constant and independent, clinging onto its value irrespective of the number of features being considered to make the target prediction.
- This might indicate that incorporating Lasso Regression in such problems may not offer improved benefits as we identify and integrate new features.
- We can also observe that the F-score based Regression tries to converge to a minimal RMSE value at a faster rate at many of the instants as opposed to the MIR metric which takes time to converge to the minimal RMSE

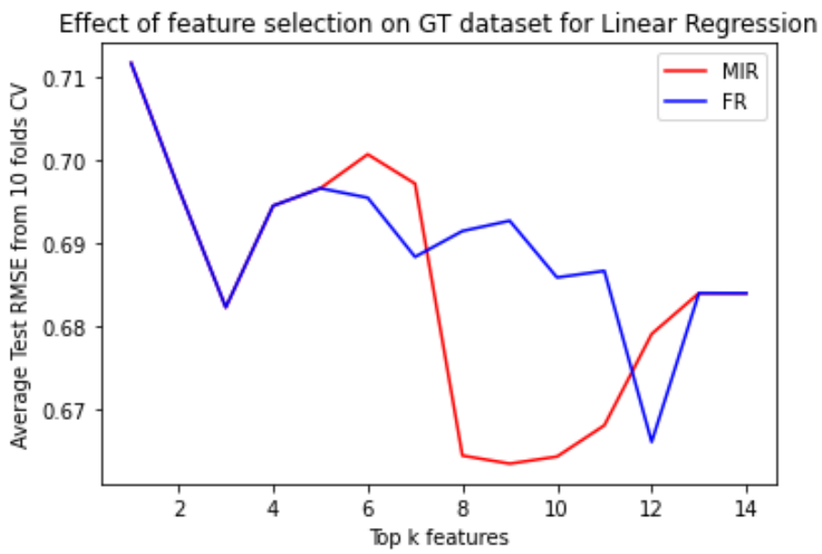


Figure 24: Effect of Feature Selection on GT Dataset- Linear Regression

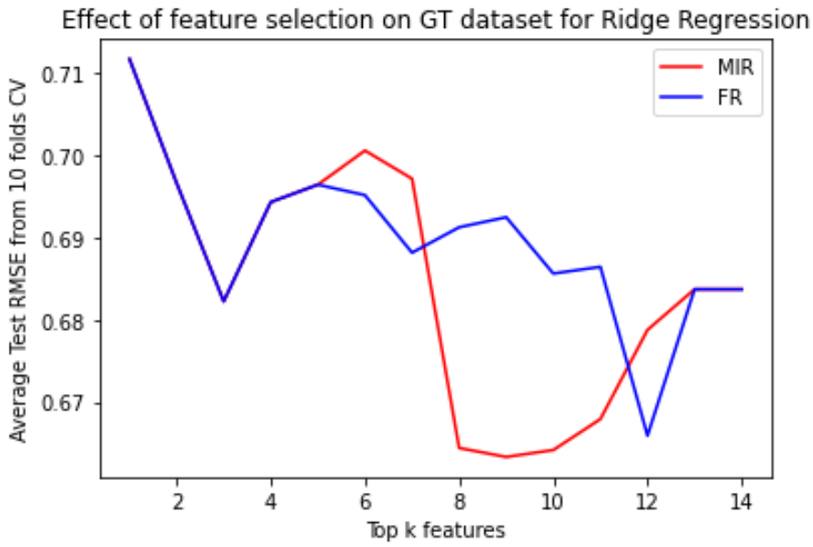


Figure 25: Effect of Feature Selection on GT Dataset- Ridge Regression

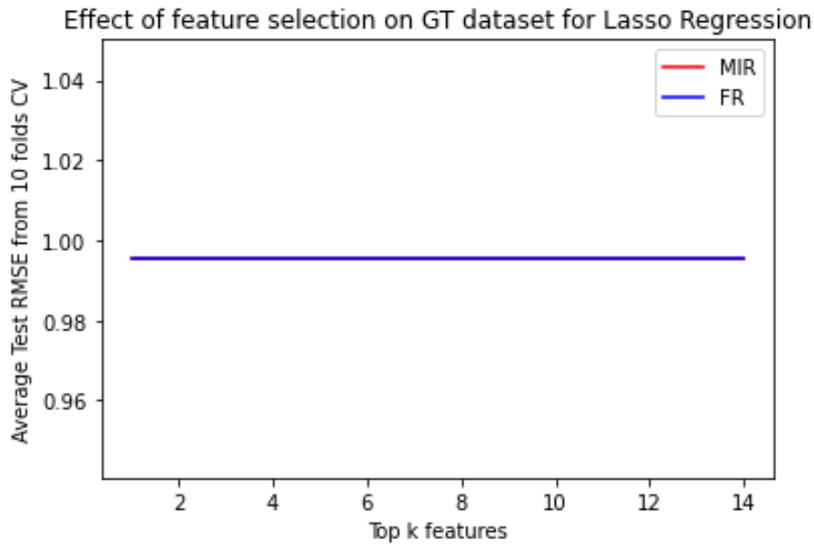


Figure 26: Effect of Feature Selection on GT Dataset- Lasso Regression

- Similar to Diamond dataset, by analyzing the feature selection plots obtained by using the three different regression techniques, we can draw some common patterns here too in addition to some differences.
- First thing is we can see that as we increase the number of features being considered, the RMSE value drops to a value less than 0.67 @K=8 features, and then RMSE remains constant for K=9,10 and then starts to bounce back to higher values and becomes more saturated(when extrapolated probably).
- As we keep increasing number of features from 0 to 8, the RMSE value decreases drastically, whereas once the number of features- K crosses the value of 6, the incremental amount of RMSE improvement for every additional feature is very low, this indicates the saturation behaviour,
- The Sudden decrease in RMSE and then further increase beyond number of features, K=8, makes us suspect that some level of dimensionality very close to the neighbourhood of K might be the optimum polynomial dimension to represent, model and solve this problem with maximum capacity.

- We observe that very similar Feature selection vs RMSE characteristics for both Linear Regression and Ridge Regression.
- Whereas for the Lasso Regression, we observe that the RMSE metric clearly remains constant and independent, clinging onto its value irrespective of the number of features being considered to make the target prediction.
- This might indicate that incorporating Lasso Regression in such problems may not offer improved benefits as we identify and integrate new features.
- We can also observe that the F-score based Regression tries to converge to a minimal RMSE value at a faster rate at many of the instants as opposed to the MIR metric which takes time to converge to the minimal RMSE

Question 8

Explain how each regularization scheme affects the learned hypotheses.
The objective functions for ordinary least squares, Lasso and Ridge can be formulated respectively as,

$$\text{OLS} : \min_{\beta} \|Y - X\beta\|_2^2$$
$$\text{Lasso} : \min_{\beta} \|Y - X\beta\|_2^2 + \lambda\|\beta\|_1$$
$$\text{Ridge} : \min_{\beta} \|Y - X\beta\|_2^2 + \lambda\|\beta\|_2$$

where $\| \ \|_1$, $\| \ \|_2$ represent L1 and L2 norm of the vector respectively, and λ is the penalty term for regularization.
Ridge regression is also known as linear regression with L2 or Tikhonov regularization, whereas Lasso regression is known as linear regression with L1 regularization. Regularization promotes generalization by resulting in simpler models that are less prone to overfitting. It seeks to enhance generalizability by boosting estimator bias while decreasing variance, resulting in a simpler model that works well on unseen test data but may have a lower total RMSE.

- As the regularization strength (λ) grows for both types of regularization, more weights in the learnt model are set to 0. This is because the L1 and L2 regularizations are designed to minimize the weights. Thus, in order to keep the regularization section of the least squares equation from getting too big, the optimization program will try to minimize the weights; ideally, the weights should be close to zero. However, when models get simpler, they begin to lose the most critical weights needed to promote the salient characteristics to contribute to the final estimation. As a result, very high regularization strength levels cause the test accuracy to plummet substantially (underfitting). Finite levels of regularization strength, on the other hand, can assist make the model more generalizable on previously unknown test data, preventing overfitting. Regularization also improves model stability by lowering variance and sensitivity to outliers while raising bias.

- Lasso regularization is appropriate for feature selection and the building of basic and sparse linear regression models in which features with 0 weights can be eliminated. Because the subgradient of $\| \ w \ \|$ is equal to the same sign as **w**, L1 regularization promotes all sorts of weights to decrease to 0, regardless of the magnitude of w. For comparable test accuracies, L1 regularization is more likely to zero out coefficients than L2 regularization because it assumes priors on weights drawn from an isotropic Laplace distribution (linear descent), which has a significantly smaller Q factor than the Gaussian distribution (exponential descent). As a result, only a subset of characteristics are active in the learnt hypothesis.

- Ridge regularization, which implies priors on weights drawn from a unit Gaussian distribution, is appropriate for lowering the impacts of collinear features, which can lead to higher variance (and hence model instability and susceptibility to outliers). This is due to the fact that the subgradient of $\| \ w \ \|^2$ is affected by not just the sign of w but

also the magnitude of w, which may be thought of as scaling the variance of weights, minimizing the model's reliance on a few features and encouraging dispersed contribution from all features. When compared to L1 regularization, this results in regression models with diffuse weights. Thus, ridge regularization encourages the inclusion of all characteristics in the learnt hypothesis in order to avoid overfitting.

Specifically, here we see the following impacts of different Regression Algorithms(OLS, Ridge, Lasso) on the Two Datasets:

For the Diamond Dataset:

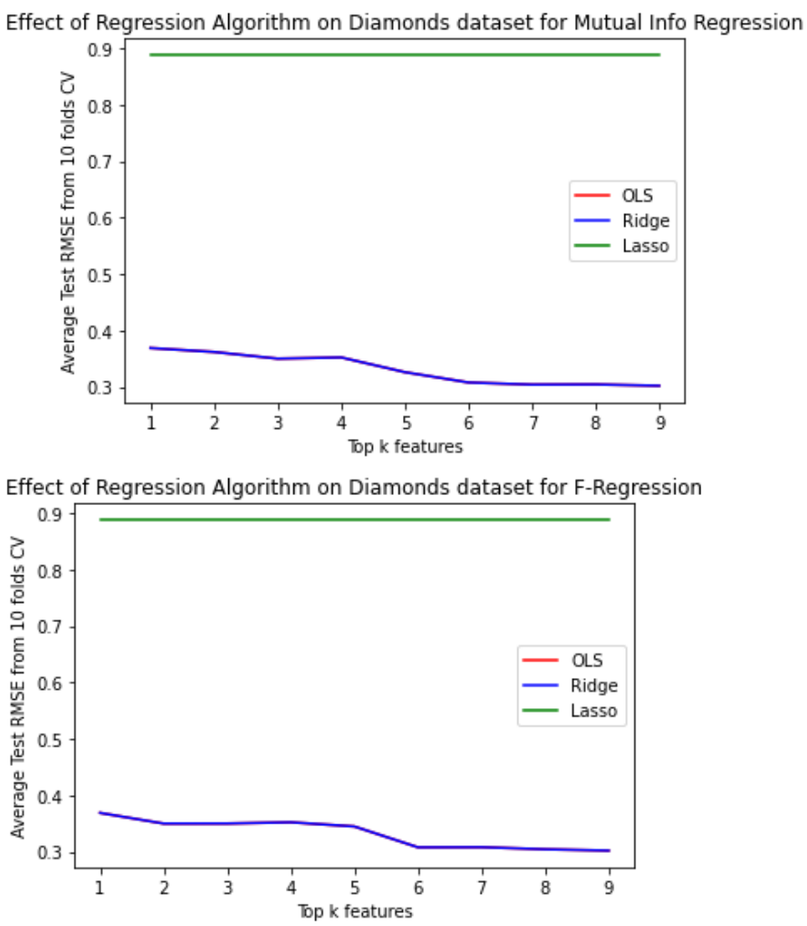


Figure 27: Effect of Regression Algorithm on Diamond dataset for Mutual Info Regression; Effect of Regreesion Algorithm on Diamond dataset for F-Regression

For the GT Dataset:

Figure 28: Effect of Regression Algorithm on GT dataset for Mutual Info Regression; Effect of Regreesion Algorithm on GT dataset for F-Regression

Question 9

Report your choice of the best regularization scheme along with the optimal penalty parameter and briefly explain how it can be computed.
For this question, we test the performance in terms of train and test RMSE of linear regression, lasso regression and ridge regression on top 10 selected features. To compute the optimal penalty parameter for each regularization scheme as well as test general model performance, we perform grid search with 10-fold cross validation (GridSearchCV()). We test over the range of λ [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]. We run for both mutual info regression and f-regression, with best k features, where ke ranges from 1 to total number of features.Full results are appended at the end.
For Diamond Dataset:

Best RMSE metrics are obtained using the Linear Regression

Best RMSE using Standardisation and Linear Regression: 0.300928763 (num_features or k = 6)
Best RMSE without using Standardisation and Linear Regression: 0.300928763 (num_features or k = 6)

test_score	train_score	Standardize	Model
0.300928763	0.3125287816	TRUE	LinearRegression
0.300928763	0.3125287816	FALSE	LinearRegression
0.300928765	0.3125287816	TRUE	Ridge(alpha=0.0001)
0.3009287724	0.3125287816	FALSE	Ridge(alpha=0.0001)
0.3010687817	0.312531704	TRUE	Lasso(alpha=0.0001)
0.3011460647	0.3125336074	FALSE	Lasso(alpha=0.0001)

For GT Dataset:

Best RMSE metrics are obtained using Lasso Regression incorporating Standardisation and one hot encoding representation for the 'year' feature:

Best RMSE using Standardisation and Lasso Regression: 0.6495247607 Lasso(alpha=0.01) (num_features or k = 12)
Best RMSE without using Standardisation and Lasso Regression: 0.6501135353 Lasso(alpha=0.01) (num_features or k = 10)

test_score	train_score	Standardize	Model
0.6495247607	0.6373622583	TRUE	Lasso(alpha=0.01)
0.6501135353	0.6347775254	FALSE	Lasso(alpha=0.01)
0.6543662606	0.6341046523	FALSE	Ridge(alpha=1000)
0.6551903549	0.6376917393	TRUE	Ridge(alpha=1000)
0.6642276887	0.6322371873	TRUE	LinearRegression
0.6642276887	0.6322371873	FALSE	LinearRegression

Question 10

Does feature scaling play any role (in the cases with and without regularization)? Justify your answer.

Observation

- In the case of Linear Regression, the presence or absence of standardisation techniques in the processing pipeline did not cause any decrease or change the minimum or Best RMSE error observed so far. With this, we can conclude that that standardization may not be necessary in case if we stick to just Linear Regression.
- In the Case of Lasso or Ridge Regressions, we observed that the usage of feature scaling or standardization techniques has resulted in improvement in minimum attainable RMSE metric. This has been the case with Diamond Dataset. For Gas Emission, it can improve or increase RMSE.
- Hence we conclude that the feature scaling might be helpful depending on the type of the regularization scheme that we are implementing to build the model.

For Diamond Dataset, we observe that though the training error is same, test error improves with Standardization in case of Ridge and Lasso Regularization. For linear regression, it makes no difference.

test_score	train_score	Standardize	Model
0.300928763	0.3125287816	TRUE	LinearRegression
0.300928763	0.3125287816	FALSE	LinearRegression
0.300928765	0.3125287816	TRUE	Ridge
0.3009287724	0.3125287816	FALSE	Ridge
0.3010687817	0.312531704	TRUE	Lasso
0.3011460647	0.3125336074	FALSE	Lasso

For Gas Emission Dataset, we observe that though the training error and test error can increase or decrease for different hyperparameters, in case of Ridge and Lasso Regularization. For linear regression, it makes no difference.

test_score	train_score	Standardize	Model
0.6495247607	0.6373622583	TRUE	Lasso
0.6512118343	0.6347775254	FALSE	Lasso
0.6543662606	0.6341046523	FALSE	Ridge
0.6551903549	0.6376917393	TRUE	Ridge
0.6642276887	0.6322371873	TRUE	LinearRegression
0.6642276887	0.6322371873	FALSE	LinearRegression

Question 11

Some linear regression packages return p-values for different features2. What is the meaning of them and how can you infer the most significant features?
In the linear regression model, P-values assess the likelihood of feature coefficients being equal to zero. As a result, if the p-value for a given feature is extremely close to zero, we may be certain that the feature is important in the linear model.
The p-value for each feature tests the null hypothesis that the feature has no connection with the target variable at the population level (probability of the feature's weights being 0). A characteristic with a high p-value (> 0.05) implies that there is insufficient evidence to establish a significant relationship between that feature and changes in the target-variable. A feature with a low p-value (0.05), on the other hand, indicates that the probability of the coefficients of that particular feature being 0 is low, indicating that the feature is well-suited as a salient feature, contributing to changes in the target variable and ultimately rejecting the null hypothesis for that particular predictor (statistically significant feature). As a result, the most significant characteristics will have a low p-value.
We use python package **statsmodels** to compute p-values for all features conveniently.
The p values corresponding to the diamond dataset is given by:

```
const       1.000000e+00
carat       0.000000e+00
cut         1.113990e-98
color       0.000000e+00
clarity     0.000000e+00
x          3.098309e-136
y           3.501855e-02
z           4.158692e-01
depth       4.663890e-62
table       1.130952e-19
```

This highlights that the most significant features are carat, color and clarity, followed by x, cut, depth and table. The parameters y and z are of relatively lesser importance.

The p values corresponding to the Gas Emission dataset is given by:

```
const         6.900833e-01
AT            8.288866e-57
AP            1.780828e-02
AH            9.493852e-01
AFDP          7.389446e-01
GTEP          4.698295e-01
TIT           4.044448e-02
TAT          3.310755e-126
TEY          1.223186e-137
CDP           5.535595e-11
year_2011     5.733834e-01
year_2012     5.733834e-01
year_2013     5.733834e-01
```

```
year_2014       5.733834e-01
year_2015       5.733834e-01
```

This highlights that the most significant features are TEY, TAT, AT and CDP. All the other parameters are of relatively lesser importance.

Question 12

Look up for the most salient features and interpret them
The results found in this question relate to the best polynomial regressors found via 10-fold grid-search cross-validation for each dataset. To find the most salient features, we find the coefficients with the highest absolute values in the polynomial model.

Most important Salient features are found and listed below for the two datasets:

Diamond dataset

Top 5 Salient features (Diamond): ['carat color clarity', '$carat^2$ clarity', '$carat^3$', 'carat clarity y', 'z']

- We observe that the best degree among the compound terms enforcing the Polynomial regression is 3.
- We can observe that carat is the most important of the original features. Therefore, it repeatedly takes multiple forms in the polynomial regression. This is supported by the heatmap in Question 2.
- The relationship between price and carat could be seen as something along the form of $y = \alpha x + \beta x^2 + \gamma x^3$. This makes sense as more carat, higher the price of the diamonds.
- That is the most salient features of the Diamond dataset comprise of different mathematical combinations starting from the monomial type to squared to cross multiplication of different features or same individual features.
- We can also witness how the top 5 salient features that offer the best performance are embedded and revealed on top of the Pearson Correlation Matrix Heatmap, where we understood that the parameters such as carat, y and z dimensions offer the highest correlation with the Target variable under consideration - (Price of the diamond).
- The product of carat, color and clarity seems to be the most important feature.

Gas Emission Dataset

Top 5 Salient features (GT): ['TAT', 'TIT TAT', 'TIT', 'TEY', 'TAT TEY']

- We observe that the best degree among the compound terms enforcing the Polynomial regression is 2.
- This basically means that a regularization carried on with the proper combination of the below mentioned numerical features or regularization utilizing the individual numerical features might be more applicable
- We can observe that TAT is the most important of the original features. Therefore, it repeatedly takes multiple forms in the polynomial regression.
- TAT, TIT and TEY are the most important features. We can see that the Heatmap had predicted that TIT and TEY would be important features as well.
- Product of the inlet and after temperatures, TIT and TAT seems to be a very important feature.

Question 13

What degree of polynomial is best? What does a very high-order polynomial imply about the fit on the training data? How do you choose this parameter?

Impact of Polynomial Degrees on the two datasets are mentioned below. We use Ridge Regularization.

Impact of Polynomial Degree on the Diamond Dataset

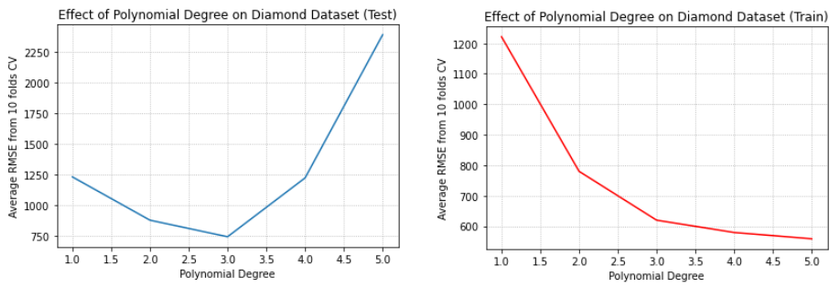


Figure 29: Impact of Polynomial Degree on the Diamond Dataset

Rank	mean_test_score	mean_train_score	degree	alpha
1	0.1856840557	0.1823719903	3	1000
2	0.2154015408	0.1701323558	3	100
3	0.2195150664	0.2233169958	2	1000
4	0.2358374886	0.2048298835	2	100
5	0.2477170499	0.1983588139	2	10

- From the plots, we can clearly observe that our model attains the least minimum value of RMSE around the polynomial degree of value 3.0.
- From the test results plot, We can also observe that the RMSE value starts improving as the polynomial degree increases and approaches close to 3.0 and after reaching 3, we see that it starts degrading almost linearly for the polynomial regression based regularizations for Polynomial degrees greater than 3.
- This might mean that Polynomial regularizations having the highest order of terms/degree greater than 3 inflict serious/undesired penalties on the model during the training and backpropagation, which heavily decreases the model's capacity to strike back and learn useful content.
- It might also mean that any multiplicative combination of more than 3 input arbitrary features from the dataset is not a fair way of regularizing the learning behaviour of the model during overshoot or undershoot.
- This might be inferred by visualizing how the multiplicative combination of three unique features degrades their individual absolute values and hence the resulting penalty term that might be generated during the process might not be more meaningful or related to the significance of the individual or even the combination of the input features.
- On the contrary, upon looking at the train results sweep of Polynomial Degree vs Average RMSE, the RMSE value keeps on decreasing. But this is expected because, the

model is continuously operated on the known/previously encountered data throughout the process, hence its error seems to decrease continuously with the increase in the Polynomial Degree of Regression

Impact of Polynomial Degree on the GT Dataset

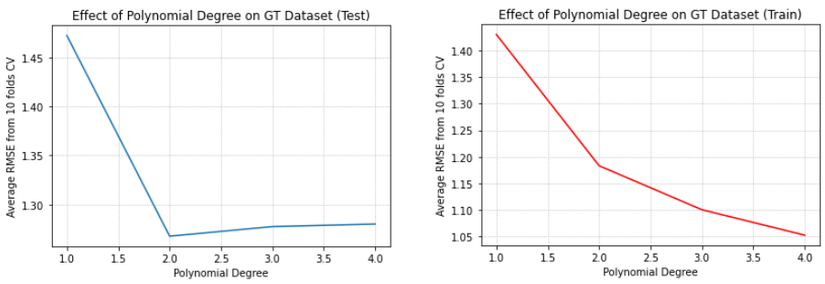


Figure 30: Impact of Polynomial Degree on the GT Dataset

Explanation of RMSE variation with change in Polynomial Degree on GT Dataset

- From the plots, we can clearly observe that our model attains the least minimum value of RMSE around the polynomial degree of value 2.0.
- From the test results plot of GT dataset, the RMSE value starts improving as the polynomial degree increases and approaches close to 2.0 and after reaching 2, we see that it starts degrading gradually for the polynomial regression based regularizations for Polynomial degrees greater than 2.
- Unlike the trends in the Diamond dataset, we observe that the degradation in RMSE with the increase in polynomial degree beyond the seemingly optimum degree of 2.0 is not linear, but exhibit more of a saturated behaviour with the RMSE value still remaining very close to the minimmum value of 3.26 at 2.0.
- This might mean that Polynomial regularizations having the highest order of terms/degree greater than 2 do not inflict serious/undesired penalties on the model during the training and backpropagation. Hence the higher order polynomials in addition to not being helpful in improving the resultant metric do not lead to severe degradation of the resultant metric as well.
- It might also mean that any multiplicative combination of more than 3 input arbitraty features from the dataset is not a fair way of regularizing the learning behaviour of the model during overshoot or undershoot.
- This might be inferred by visualizing how the multiplicative combination of three unique features degrades their individual absolute values and hence the resulting penalty term that might be generated during the process might not be more significant than the regularization defined by 2nd order Polynomial Regression.
- On the contrary, upon looking at the train results sweep of Polynomial Degree vs Average RMSE, the RMSE value keeps on decreasing. But this is expected because, the model is continuously exposed to the known/previously encountered data throughout the process, hence its error seems to decrease continuously with the increase in the Polynomial Degree of Regression

Rank	mean_test_score	mean_train_score	degree	alpha
1	0.5602573924	0.5273179766	2	0.1
2	0.5645625542	0.5216594742	3	1
3	0.565728768	0.5250935946	4	10
4	0.5712478403	0.5124910498	4	1
5	0.576828473	0.5384905996	2	1

We observe that higher degree of polynomial implies more complex models. These models can fit the training data better and might lead to overfitting. This is observed for higher order polynomials. Hence we should choose the degree such that it is complex enough to avoid underfitting but not too complex that it overfits. For instance, in the diamond data, we see the training error and validation error decrease till degree = 3. So degrees of 1 and 2 were underfitting. But after 3, though training error decreases, validation error increases. Therefore degree > 3 shows overfitting. So we choose degree as 3.

Question 14

For the diamond dataset it might make sense to craft new features such as z = x1 × x2, etc. Explain why this might make sense and check if doing so will boost accuracy.

Multiplying and taking inverse of particular features might yield a new feature that is highly correlated with the target variable. It is to be noted that it is impossible for polynomial regression to craft these features as it cannot yield inverse relationships (degree is always positive). For the Diamond dataset, an example of a feature where it is not just multiplication but inverse is used and which might help in the estimation of the price is 'Density of the diamond'. Thus, the density feature is:
Density = carat / (x * y * z)
We retrained using Grid Search CV like last time using the same hyperparameter space. We obtained the following results.

```
Average Test RMSE with density feature: 0.17403316392998497
Best Parameters:
'steps': [('kbest', SelectKBest(k=7, score_func=mutual_info_regression)),
('PR', PolynomialFeatures(degree=3)),
('standardize', StandardScaler()),
('model', Ridge(alpha=1000.0, random_state=42))]
```

We observe that the test RMSE drops in this case dropped, compared to the previous one of '0.1856840557'. Therefore, this proves our hypothesis that Density feature is correlated with the Diamond prices.

Question 15

Why does it do much better than linear regression?
The values for RMSE for Linear regression with standardization are given below:

- Linear Regression: 0.300928763
- Ridge Regression with alpha = 0.0001: 0.300928765
- Lasso Regression with alpha = 0.0001: 0.3010687817

The values for RMSE for MLP is 0.2372866269
The above results are for the diamond dataset. We observe a similar trend for the Gas Turbine CO Emission Data Set.

- Artificial neural networks are algorithms that can be used to perform nonlinear statistical modeling and provide a new alternative to logistic regression.
  Regression methods are good at dealing with linear dependencies, neural networks can deal with non linearities. Since the data we use has some nonlinear dependencies, neural networks perform better than regression.
- Because neural networks have more information capacity (trainable parameters) than linear regression models, MLP can capture complex mathematical relationships and mappings via hidden layer connections between features and target variables that would otherwise be impossible to infer using linear dependencies with a limited number of trainable weights.
- By incorporating several hidden layers, a multi-layer perceptron model may fit highly intricate relationships, so we just include all of the characteristics in this section.

Question 16

Adjust your network size (number of hidden neurons and depth), and weight decay as regularization. Find a good hyper-parameter set systematically.
On the diamond and gas turbine emission dataset, we evaluate MLP performance in terms of train and test RMSE (The actual MSE is simply the positive version of the number you're getting) or various weight decays, model activation functions, and network size (number of hidden neurons and depth). Because F-Score-selected features outperformed MI, we chose the top ten F-Score features. The hyperparameter space looked like this:

- Model activation: tanh, logistic, ReLU
- Weight decay penalty term *alpha* (With regularization): $[10^{-4}, 10^{2}]$
- Network size: All combinations of (30,40,60) as the number of hidden neurons, with minimum network depth as 1, and maximum as 3.

We utilized 10-fold grid search cross-validation using the Sci-Kit Learn pipeline() to systematically determine the most optimum hyperparameters for each dataset in the parameter design space. Although it is possible to utilize a much larger design space for network sizes, we confined the network sizes to combinations of four integers to speed up the search process. Below table displays the best hyperparameter configuration for each dataset, as well as the best average train and test RMSE.

Network Architecture	**Activation function**	**Weight decay or *alpha***	**Mean Train Score**	**Mean Test Score**
(40, 40)	ReLU	0.01	0.1243414139	0.14728662687

Table 1: Performance summary of fully connected neural network with most optimal hyperparameters on diamond Dataset

Network Architecture	**Activation function**	**Weight decay or *alpha***	**Mean Train Score**	**Mean Test Score**
(30, 30, 60)	ReLU	10	0.5744953489	0.6295302481

Table 2: Performance summary of fully connected neural network with most optimal hyperparameters on gas turbine Emission Dataset

Question 17

What activation function should be used for the output? You may use none

The neural network is solving a regression issue with continuous outputs ranging from negative infinity to positive infinity, therefore the output activation function should be none or linear (identity), both of which can generate outputs in the range.

ReLU activation is limited to 0 and positive values, tanh activation is limited to -1 and 1, and sigmoid/logistic activation is limited to 0 and 1.

Regression outputs, on the other hand, can take any value beyond such limited bounds in the full domain of real numbers, which is achieved in the output layer by linear activation or no activation at all. For classification problems, we pair the Neural network with a softmax or sigmoid.

Question 18

What is the risk of increasing the depth of the network too far?
There are various reasons why we should not increase the depth of the neural network indefinitely. Some of the reasons are given below:

- Overfitting:
  A deeper neural network contains more trainable parameters than a shallow one, resulting in a complicated model with ample memory to memorize the full training set, resulting in overfitting on the training data and poor generalization on the test set.

- Vanishing Gradients:
  If a network is excessively deep, the gradients will become exponentially tiny as information returns from the last layers to the starting layers. This causes sluggish or early convergence to a sub-optimal point during the training phase, with the weights in the beginning layers being exceedingly tiny in comparison to the weights in the final layers.

- Compute constraints:
  For training, neural architecture search, and real-time deployment, a complicated network with numerous layers requires more time, memory, and computational resources than a simpler model. Furthermore, doing grid search to discover suitable hyperparameter values for a neural network with numerous hidden layers takes a long time.

- Data requirements:
  If the network includes millions of trainable weights, a large training set is required to avoid overfitting and give robust and useable outputs upon deployment.

- Interpretability:
  Large networks with millions of parameters can result in intricate correlations between characteristics and goal variables that are beyond the scope of human comprehension, resulting in black-box models.

Question 19

Apply a random forest regression model on datasets, and answer the following. Fine-tune your model. Explain how these hyper-parameters affect the overall performance? Do some of them have regularization effect?

On the diamond and gas turbine emission dataset, we evaluate random forest performance in terms of train and test RMSE (The actual MSE is simply the positive version of the number you're getting) for various number of trees, maximum number of features and depth of each tree on the Diamond and gas Turbine Emission Datasets. Because F-Score-selected features outperformed MI, we chose the top ten F-Score features. The hyperparameter space looked like this:

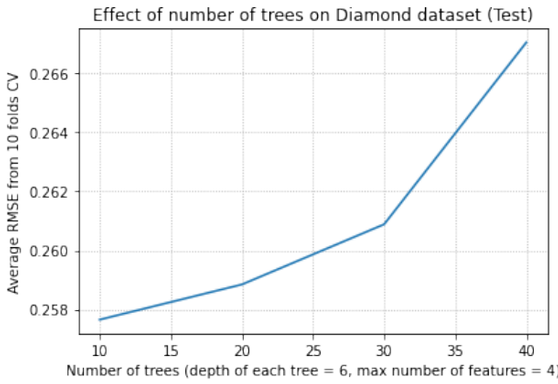
- Maximum number of features: 1 to 7
- Number of trees/estimators: 10 to 50
- Depth of each tree: 1 to 7

We utilized 10-fold grid search cross-validation using the Sci-Kit Learn pipeline() to systematically determine the most optimum hyperparameters for each dataset in the parameter design space. On each dataset, we additionally calculate the "Out-of-Bag Error" (OOB) for the top random forest models. To investigate the impacts of each hyperparameter individually on overall performance, we maintain two of the three hyperparameters fixed while visualizing the effects of the target hyperparameter on average train and test RMSE using 10-fold grid search cross-validation.

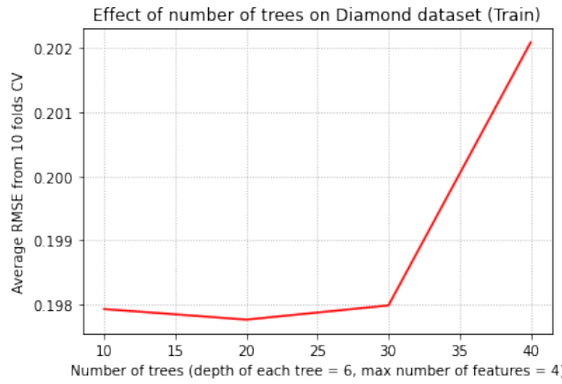
Dataset	**Maximum Number of features**	**Depth of tree**	**Number of trees**	**Mean Train score**	**Mean Test score**
Diamond	7	6	40	0.1158789838	0.1909485699
Gas Turbine Emission	4	6	40	0.4581012859	0.5806434108

Table 3: Performance summary of Random Forest with most optimal hyperparameters on Diamond and Gas Turbine Emission Dataset

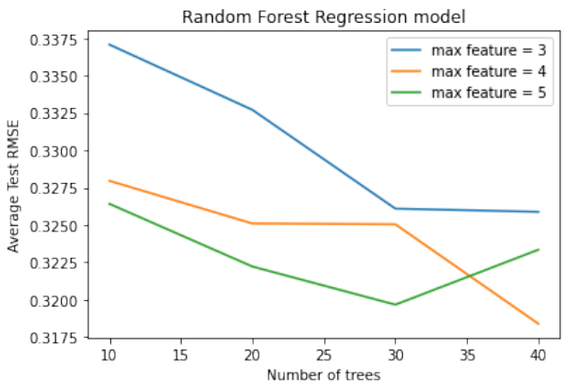
The below figure depicts the effect of number of trees on the diamond dataset.



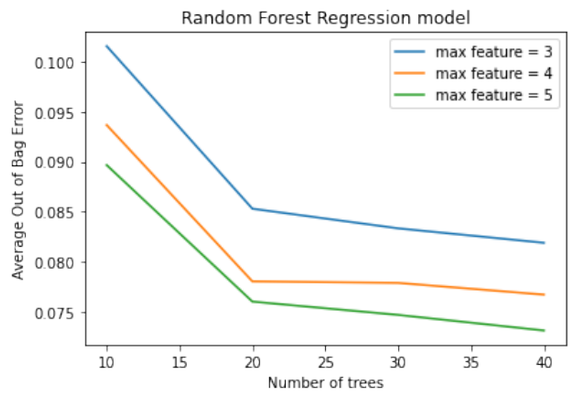
(a) Max Depth = 6, Number of features = 4



(b) Max Depth = 6, Number of features = 4



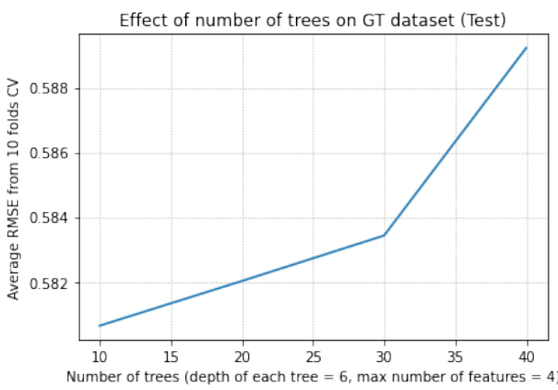
(c) Max Depth = 4



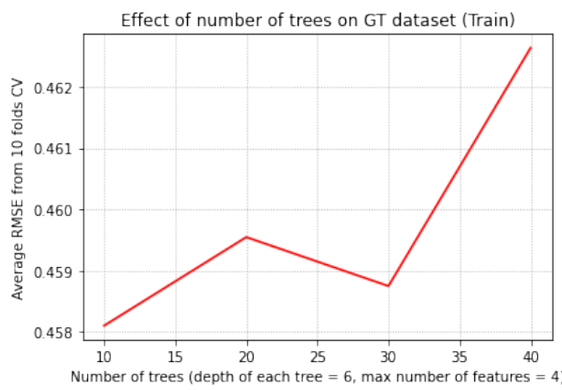
(d) Max Depth = 4

Figure 31: Effect of number of trees on Diamond Dataset

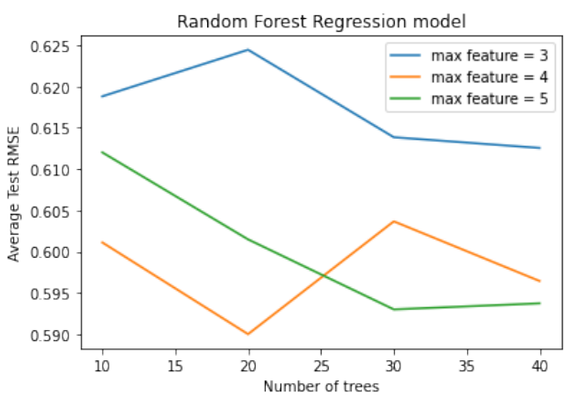
The below figure depicts the effect of number of trees on the Gas Emission dataset.



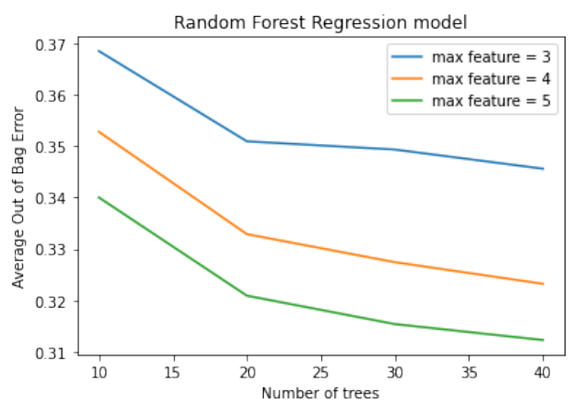
(a) Max Depth = 6, Number of features = 4



(b) Max Depth = 6, Number of features = 4



(c) Max Depth = 4



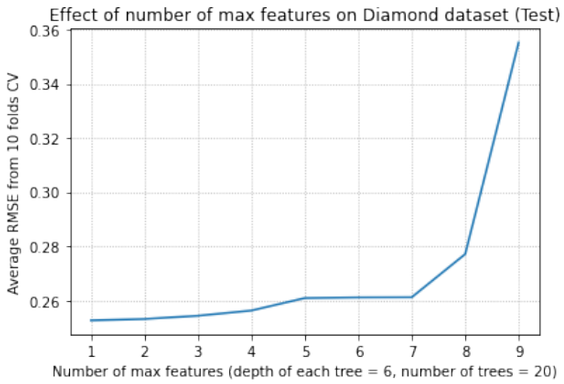
(d) Max Depth = 4

Figure 32: Effect of number of trees on Gas Emission Dataset

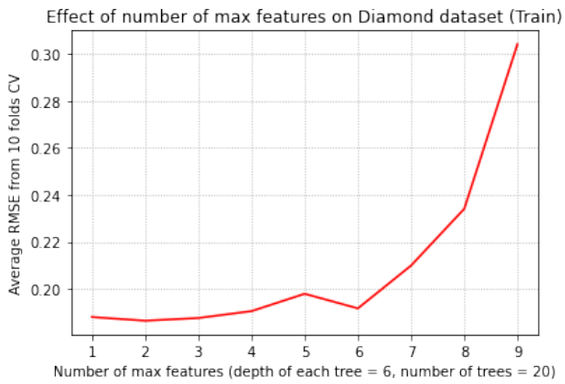
Observations when we vary the number of trees:
We make the below observations on the above figures for both the datasets.

- We see that the overall model performance is non-monotonous with respect to the number of trees, i.e. increasing the number of trees seem to improve or stabilize the performance but the performance improvement is non-monotonic. When the maximum depth is set as 6, we see that increasing the number of trees increases the RMSE. So in such cases we need to choose lower number of trees. On the other hand for maximum depth of 4, one should select a sufficiently large value for the number of trees (e.g. 64 to 128) within compute constraints, as more number of trees does not cause overfitting.
- In a random forest, each tree and its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN), because the trees are grown using a randomization technique on their individual bootstrap subsamples, which are uncorrelated with the growth of other trees. Thus, according to WLLN, the target variable and tree-decision have limited variance, causing the random forest's overall decision (and any other statistic) to converge towards a mean value with decreasing rewards as the number of trees approaches infinity (Jensen's inequality).
- The anticipated error rate for a random forest ensemble is a non-monotonous function of tree count. In reality, after a significant number of needed estimators have been utilized, error measurements such as RMSE or OOB become noisy. The convergence rate of the error rate curve is independent of the number of trees and solely depends on the distribution of the anticipated value of the trees' choices.
- While increasing the number of trees does not result in overfitting according to WLLN, other hyperparameters may result in superfluous variance in the model and over-correlation within the ensemble, shattering the random forest's idea of independent trees.

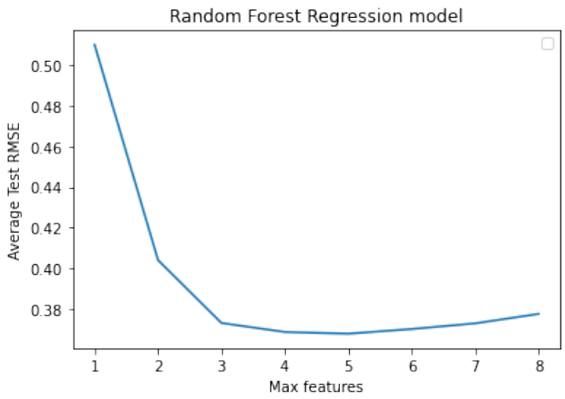
The below figure depicts the effect of number of features on the diamond dataset.



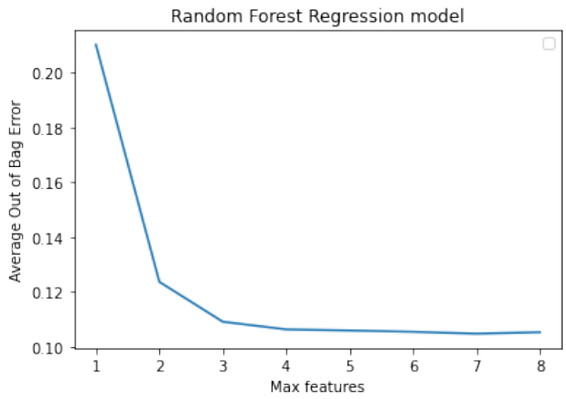
(a) Number of trees = 20, Max Depth = 6



(b) Number of trees = 20, Max Depth = 6



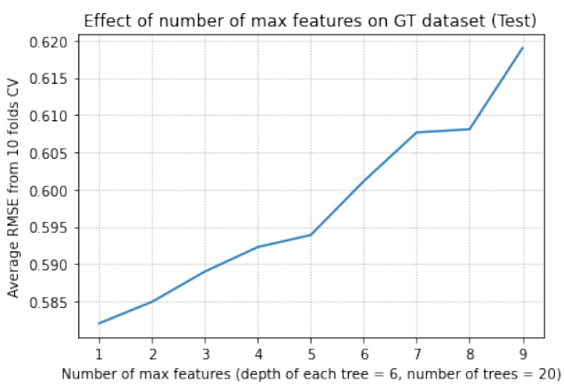
(c) Number of trees = 166, Max Depth = 3



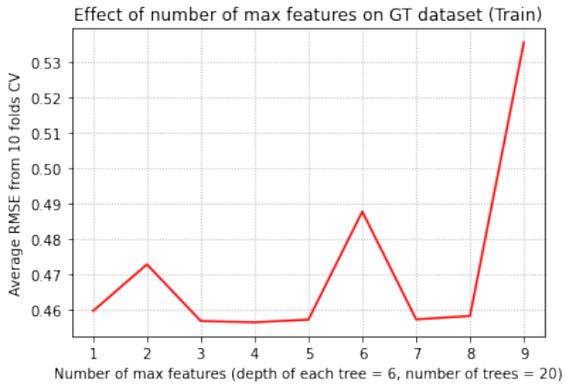
(d) Number of trees = 166, Max Depth = 3

Figure 33: Effect of number of features on Diamond Dataset

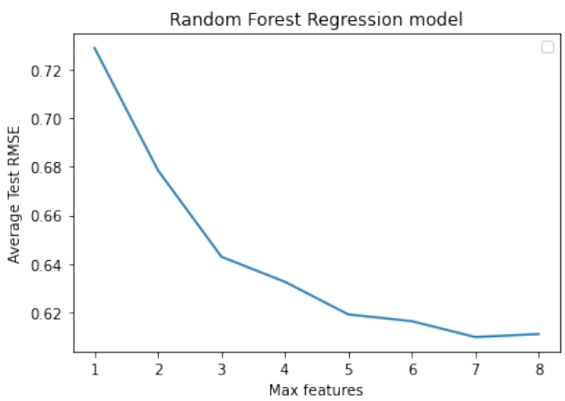
The below figure depicts the effect of number of features on the Gas Emission dataset.



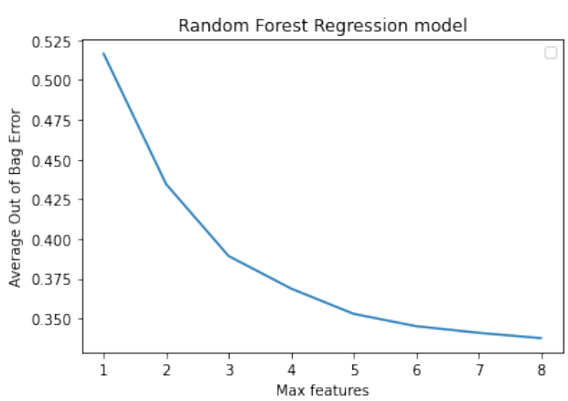
(a) Number of trees = 20, Max Depth = 6



(b) Number of trees = 20, Max Depth = 6



(c) Number of trees = 50, Max Depth = 3

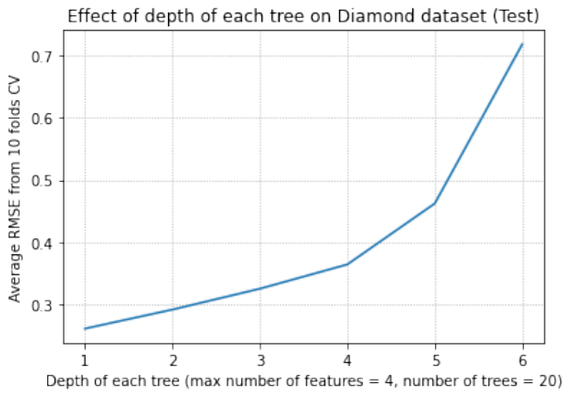


(d) Number of trees = 50, Max Depth = 3

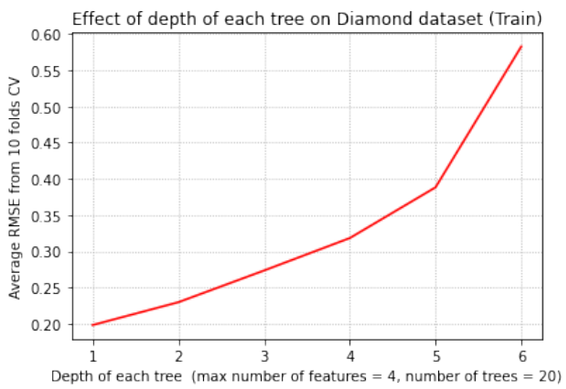
Figure 34: Effect of number of features on Gas Emission Dataset

Observations when we vary the number of features:
For both datasets, raising the maximum number of features increases the training and testing RMSE monotonically. This is observed when we run for number of trees as 20 and depth as 6. But when we set the number of trees to 166 and depth to 3, we observe that the RMSE and OOB Error decreases till number of features is 4 and then it is constant. This explains that with 4 features, the model is able to explain the target variable accurately. This also suggests that the number of features, like the depth of the tree, functions as a regularizer, with excessively large values resulting in overfitting on the training set and poor generalization on the test set. This is because increasing the amount of characteristics for each tree enhances the model capacity of individual trees but also raises the connection between trees, shattering the random forest's idea of independent trees. Selecting a fraction of the features aims to decorrelate the individual trees and improve generalization error rates, increasing the strength of the random forest.

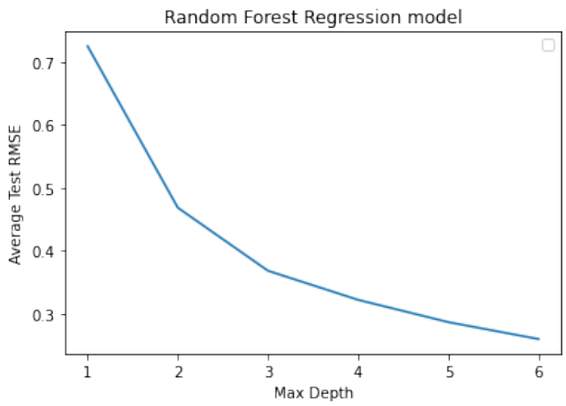
The below figure depicts the effect of maximum depth on the diamond dataset.



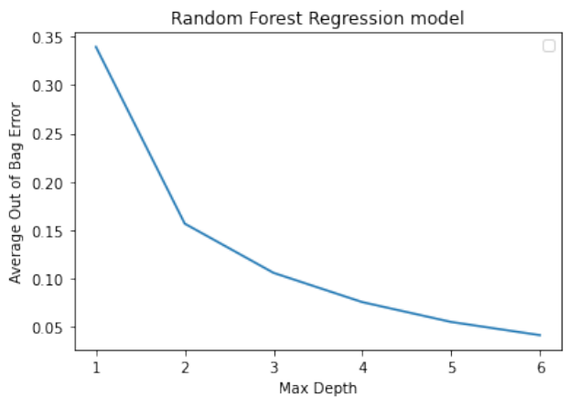
(a) Number of trees = 20, Features = 4



(b) Number of trees = 20, Features = 4



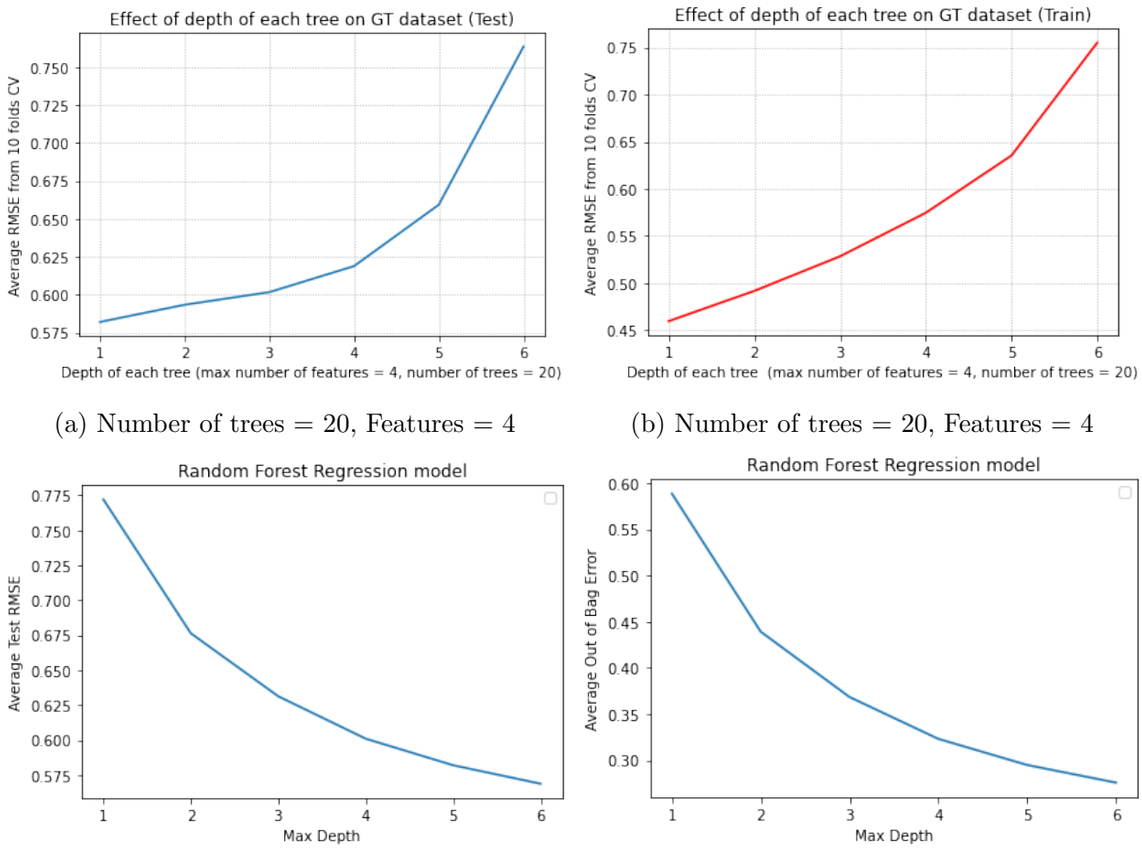
(c) Number of trees = 166, Features = 4



(d) Number of trees = 166, Features = 4

Figure 35: Effect of max depth on Diamond Dataset

The below figure depicts the effect of maximum depth on the Gas Emission dataset.



(a) Number of trees = 20, Features = 4

(b) Number of trees = 20, Features = 4

(c) Number of trees = 50, Features = 4

(d) Number of trees = 50, Features = 4

Figure 36: Effect of max depth on Gas Emission Dataset

Observations when we vary the maximum depth of trees:
Increasing the depth of each tree greatly improves the error rate for both the RMSE and OOB error for number of trees as 166 and features of 4, eventually converges to a constant value for both datasets. If the depth of each tree in the test set surpasses a particular threshold, the error rate may begin to decline. This indicates that the depth of each tree acts as a regularizer; moderately increasing the depth of each tree improves both fitting and generalizability of the random forest, whereas very large depths of each tree lead to overfitting by treating each data point in the dataset, including noise, as a leaf in the tree. A tree with a greater depth will have more fine-grained splits than a tree with a lesser depth, as well as more model capacity and the ability to fit difficult mathematical connections, but it will also be more prone to overfitting. To put it another way, increasing tree depth reduces bias but increases variation. On the other hand, for number of trees of 20, we observe that the error increases with increasing depth.

A smaller proportion of features included in tree construction and a shallower depth of each tree have a larger regularization impact on the random forest model, resulting in a monotonically increasing training RMSE but an optimal validation RMSE somewhere in between.

Question 20

Why does random forest perform well?
In a random forest, each tree and its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN), because the trees are grown using a randomization technique on their individual bootstrap subsamples, which are uncorrelated with the growth of other trees. Thus, according to WLLN, the target variable and tree-decision have limited variance, causing the random forest's overall decision (and any other statistic) to converge towards a mean value with decreasing rewards as the number of trees approaches infinity (Jensen's inequality). A random forest, in other words, contains a large number of uncorrelated models, with the final target variable being a majority decision fusion from all of the trees. Independent model bagging outperforms individual model bagging for the following reasons:

- Reduces mistakes or overfitting impacts of individual models, providing that the majority of the trees make right selections.
- Each decision tree is trained on a distinct and independent random subspace, resulting in variation, decorrelation, and diversification across individual trees. This guarantees that the random forest uses all of the characteristics in the superspace to make a choice.
- By assembling, we effectively extract the "mean of means," a robust central estimate of the target variable from a group of somewhat varied but overlapping weak learners that, when joined, yields a robust model.
- It is not necessary to scale or standardize features.

Random Forests are a good model if you want high performance with less need for interpretation. Below are the advantages of random forests:

- **Impressive in Versatility**
  It can handle binary features, categorical features, and numerical features. There is very little pre-processing that needs to be done. The data does not need to be rescaled or transformed.
- **Great with High dimensionality**
  Random forests is great with high dimensional data since we are working with subsets of data.
- **Quick Prediction/Training Speed**
  It is faster to train than decision trees because we are working only on a subset of features in this model, so we can easily work with hundreds of features. Prediction speed is significantly faster than training speed because we can save generated forests for future uses.
- **Robust to Outliers and Non-linear Data**
  Random forest handles outliers by essentially binning them. It is also indifferent to non-linear features.
- **Low Bias, Moderate Variance**
  Each decision tree has a high variance, but low bias. But because we average all the

trees in random forest, we are averaging the variance as well so that we have a low bias and moderate variance model.

- **Handles Unbalanced Data**
  Apart from the above, in case of classification (Not relevant to the current dataset tasks), it has methods for balancing error in class population unbalanced data sets. Random forest tries to minimize the overall error rate, so when we have an unbalance data set, the larger class will get a low error rate while the smaller class will have a larger error rate.

Due to the given observations above, random forests work very well with our datasets.

Question 21

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 features? Do the important features match what you got in part 3.2.1?
We picked the best tree for Diamond dataset, out of all 4 depth trees (max_depth=4, max_features=5, n_estimators=40). We used export_graphviz() from Scikit-Learn and pydot library to visualize the one of the trees in that random forest model.

Feature branching at the root node is **y**. Thus, it's safe to conclude that the most significant feature in this randomly picked tree is **y**. More generally, any feature that appears at the top node of the tree structure tends to be important. Furthermore, top nodes here also match the conclusion derived from the linear regression model.
Feature y is selected for branching at the root node. This means that y is the most important feature used to start the splitting process.In a decision tree, the features closer to the root node are more salient and significant than the features near the leaf nodes. The top features (in order) for this sample tree are:

- Level 1: y
- Level 2: y
- Level 3: y, carat, color, z
- Level 4: x, y, clarity

In part 3.2.1, based on p values, we observed that the most significant features were carat, color and clarity. We do observe carat, color and clarity as important features here as well. Only parameter y do not overlap.
Thus, it is safe to say that the most important features found from p-values of linear regression have significant overlap with the most important features found in a random decision tree within the random forest for the Diamond dataset.

Information gain measures the reduction of uncertainty given some feature and it is also a deciding factor for which attribute should be selected as a decision node or root node. It is just entropy of the full dataset – entropy of the dataset given some feature.
Decision trees are built by recursively splitting our training samples using the features from the data that work best for the specific task. This is done by evaluating certain metrics, like the Gini index or the Entropy for categorical decision trees, or the Residual or Mean Squared Error for regression trees.
The process is also different if the feature that we are evaluating at the node is discrete or continuous. For discrete features all of its possible values are evaluated, resulting in N calculated metrics for each of the variables, being N the number of possible value for each categorical value. For continuous features the mean of each two consecutive values (ordered from lowest to highest) of the training data are used as possible thresholds.
The result of this process is, for a certain node, a list of variables, each with different thresholds, and a calculated metric (Gini or MSE) for each variable/threshold tandem. Then,

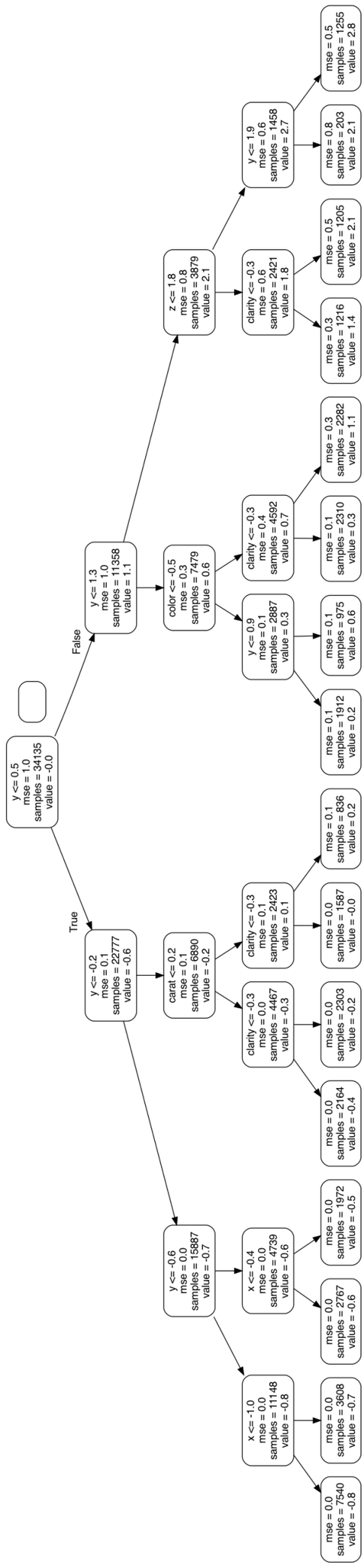


Figure 37: Random Forest of depth 4.

we pick the variable/threshold combination that gives us the highest/lowest value for the specific metric that we are using for the resulting children nodes (the highest reduction or increase in the metric).

Question 22

Read the documentation of LightGBM and CatBoost and experiment on the picked dataset to determine the important hyperparameters along with a proper search space for the tuning of these parameters.

LightGBM and CatBoost are different types of boosted trees, We read through the important hyperparameters associated with each of these boosted trees and we have described about each of the Boosted trees and their related hyperparameters in the following section:

Hyperparameters of LightGBM and CatBoost:

LightGBM- Important Key Hyperparameters:

- We pick the following Hyperparameters for the LightGBM along with the related search space.
  - Boosting Type- ('boosting_type'): 'gbdt'- gradient boosting decision tree, 'dart'- Dropouts Meet Multiple Additive Regression Trees, 'RF'- Random Forest. We briefly describe our selection here
    - Gradient Boosting Decision Tree—XGBoost (GBDT)- GBDT essentially builds the model by considering each distinct decision trees within an ensemble as a set of weak learners. The objective of the first layer is to establish relationship or fit input features to target variables, whereas the succeeding consecutive trees are then tasked with the objective of reducing the residual error between the target variable (ground truth) and the predicted value of the target variable of the trees in the previous levels while the training of the whole ensemble happens via backpropagation mechanism. Hence GDBT is widely known for its efficieny, accuracy and reliability of operation without the additional need of hyperparameter tuning. But the GDBT also suffers from the limitation of overfitting /over-specialization. This happens as the consecutive layers which would try to reduce the residual errors of the preceding trees would gradually lose its ability to perform effective correction as the levels of the trees are scaled up, hence they end up producing valid or correct predictions only for few samples belonging to the dataset and fail to generalize well for new unseen samples.
    - DART- This method treats the overspecialization problem that the GDBT trees face by introducing the known dropout solution which promotes generalization capability by dropping selected weights by going through some sort of Regularization process.
    - RF (Random Forest) - Every tree in addition with its output target variable are both independent, identically distributed random variables. Hence, by weak law of large numbers - WLLN, as the trees are grown utilizing some randomization technique on their individual bootstrap subsamples, each of them uncorrelated with growth of other trees. Thus, according to WLLN, the target variable and tree-decision possess finite variance, leading to the overall decision of the random forest to converge towards a mean value with diminishing returns as the number of trees approach infinity (by virtue of

Jensen's inequality). More Precisely, a random forest contains a large number of uncorrelated tree models, with the final target variable being a majority decision fusion combined from all of the trees.

- Depth of each tree ('max_depth'): 1 to 100 in steps of 10. As the depth of each tree is increased, it enhances the error rate for both the test and training sets, converging to a constant value. Error rate might start to degrade for the test data if the depth of each tree increases beyond a certain threshold. This particular hyperparameter functions as a regularizer and performance contributor, moderately rising the depth of each tree improves both fitting and generalizability, while very large values of depth of any tree will eventually lead to overfitting by using each data point including noise in the dataset as a leaf in the tree. Trees with larger depths will have higher fine-grained splits than trees with shallower depths, having higher model capacity and ability to fit complicated mathematical relationships. On the contraray, their limitation is that they are also prone to overfitting and large training time. Briefly, increasing tree depth decreases bias at the cost of increased variance.
- Number of leaves denoted by ('num_leaves'): 4 to 1000 in steps of 10. Number of leaf nodes acts as a regularizer and performance contributor (similar to the depth of each tree) and moderates values which improve both generalizability and fitting , while very large values lead to overfitting by treating each data point including noise in the dataset as an individual leaf in the tree. LightGBM appends leaf nodes to trees based on trends in the performance gain from the addition of those leaves, regardless of depth of each tree. Consequently, it is essential to tune both the maximum number of leaves as well as the depth of each tree to control the complexity of the model.
- Number of trees denoted by ('n_estimators') is sweeped from 10 to 4000 in steps of 100. This quantity mentions the number of boosted trees to include in the fitting process. Increasing the number of trees improves and stabilizes model performance non-monotonically. The expected rate of error for a boosted tree ensemble is usually a non-monotonous function of the number of trees involved in it,being noisy once a sufficiently large number of required estimators have been used. With all other hyperparameters fixed, the only impact that the number of trees will have on the model's loss is to decrease it stochastically. It needs the selection of a sufficiently high value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The final aggregated decision of the ensemble converges towards a mean value with diminishing returns when the number of trees approach infinity (given by Jensen's inequality). Model Performance will be hurt by having very few trees.
- L1 regularization with (alpha) as penalty parameter denoted by ('reg_alpha'): spanning (0.0001) to (10^4) Lasso regularization penalty term on the weights. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance.
- L2 regularization (lambda) penalty parameter ('reg_lambda'): spanning-(0.0001) to (10^4) where Tikhonov regularization penalty term on the weights. Small values highlight no regularization (which may lead to overfitting when other hyperparameters are considerably large), large/high values improve and render generalization

capability to model but may hurt model's runtime performance.

– Subsampling ratio or bagging fraction ('subsample'): . mentions the fraction of rows (samples) to be randomly sampled for fitting each tree. The process of training over multiple random samples without replacement is called "bagging", and is similar to how individual trees in a random forest are fitted. Decreasing subsampling ratio helps with overfitting and generalization issues and acts as a regularizer; very small values of bagging fraction may hurt model performance while very large values (no bagging) improves model performance on the training set but generalizes poorly on the test set. Bagging aims at reducing the variance of the ensemble. A similar hyperparameter for selecting features is feature fraction ('colsample_bytree') or random subspace method, which we do not use as we already selected the 10 most salient features for the ensemble to work on. We do however, test this on CatBoost for sake of completeness.

– Subsampling frequency or bagging frequency ('subsample_freq'): 0 to 50 in steps of 5. Controls how often bagging is performed, in terms of epochs.

– Minimal gain to perform split ('min_split_gain'): Swept from (0.0001 to 0) Minimum reduction in training loss that results from adding further split points required to partition a leaf node of the tree. LightGBM selects the split point that has the highest gain when adding a new tree node. Increasing this ratio acts as a regularizer, causing the ensemble to ignore very small training performance improvements that do not have meaningful impacts on generalizability of the model.

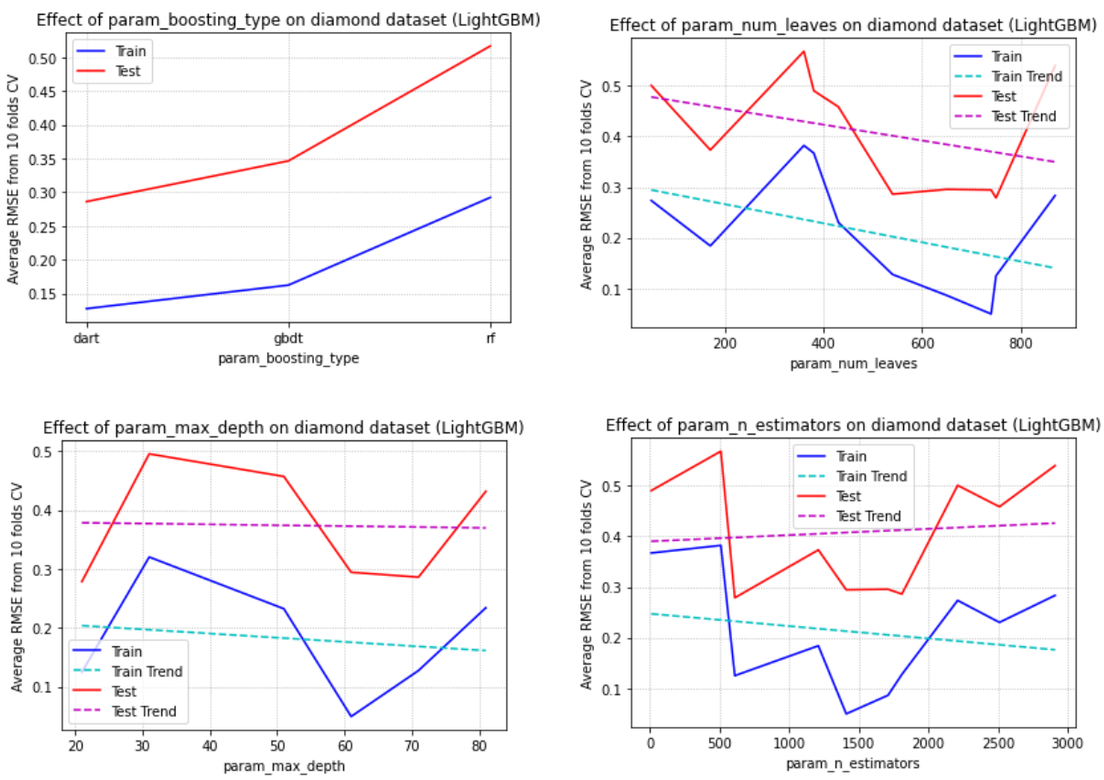


Figure 38: LightGBM Hyperparameters impact on the Diamond dataset

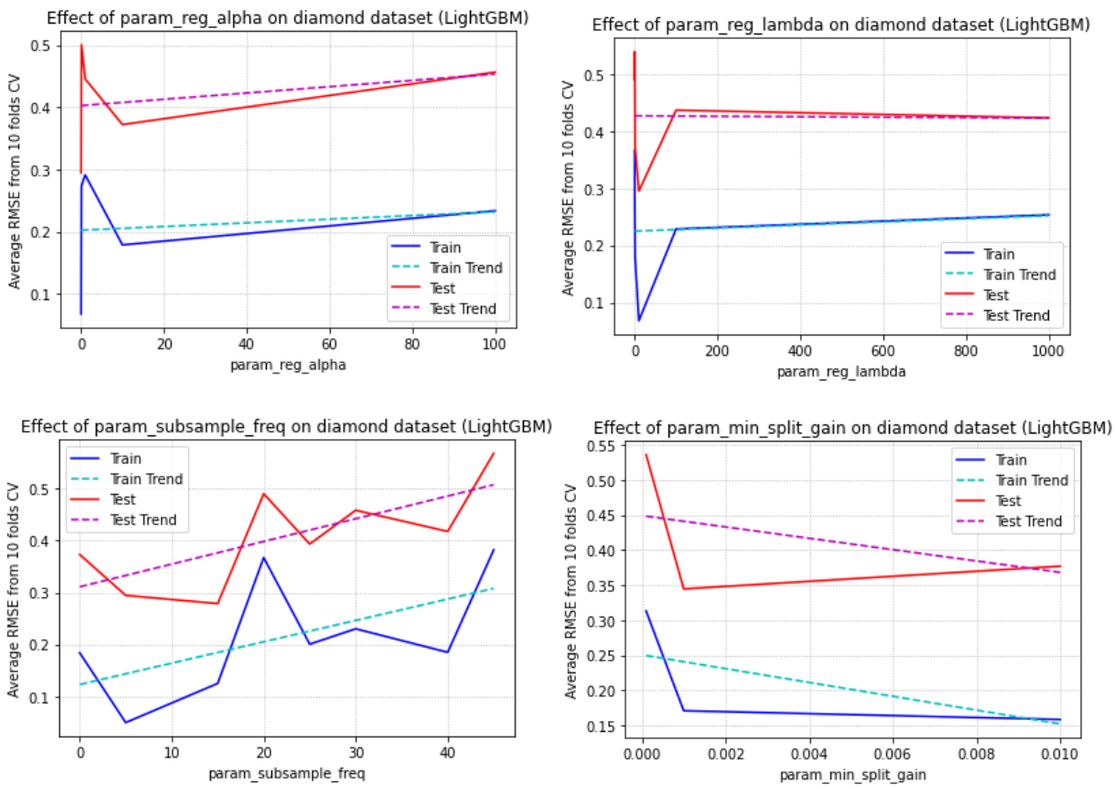


Figure 39: LightGBM Hyperparameters Impact on the Diamond dataset

Catboost- Important Key Hyperparameters:

- Feature fraction ('colsample_bylevel' or 'rsm')- [0.5, 1] in steps of 0.1- This is a random subspace method that selects the fraction of features to use at each split. Decreasing feature fraction helps with overfitting and generalization issues by selecting only the most salient and important features that have an impact on most of the samples, acting as a regularizer; very small values of feature fraction may hurt model performance while very large values (no random subspace) improves model performance on the training set but generalizes poorly on the test set. Random subspace aims at reducing the variance of the ensemble.

- Number of trees ('num_trees' or 'num_boost_round' or 'n_estimators'): 10 to 4000 in steps of 100. Mentions the number of boosted trees to include in the fitting process. Increasing the number of trees improves and stabilizes model performance in a non-monotonical way. The expected error rate for a boosted tree ensemble will take on a non-monotonous function of the number of trees, starts becoming more noisy once a sufficiently large number of required estimators have been used. Once all other hyperparameters are fixed here, the only impact the number of trees could cause on the model's loss is to reduce it stochastically. One should make a proper selection comprising of sufficiently large value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The overall decision of the ensemble gradually converges towards a mean value with diminishing returns when the number of trees tend to approach infinity (provided by Jensen's inequality). However, having a very low number of trees will hurt model performance.

- Number of leaves ('num_leaves' or 'max_leaves'): 20 to 1000 in steps of 10. This is a hyperparameter which is specifically mentioned for the lossguide growing policy. Simi-

lar to the depth of each tree, moderate values improve both fitting and generalizability, while very large values lead to overfitting by treating each data point including noise in the dataset as a distinct leaf in the tree.

- L2 regularization penalty parameter ('l2_leaf_reg' or 'reg_lambda'):

$$[10^{-4}, 10^{4}]$$

. also referred to as Tikhonov regularization penalty term on the weights. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization at the cost of affecting the model's performance parameters.

- Depth of each tree ('max_depth' or 'depth'): 1 to 16 (16 is the maximum CatBoost supports on CPU) in steps of 2. Increasing the depth of each tree improves the error rate for both the test and training datasets, eventually converging to a constant value. For the test set, the error rate might begin to degrade if the depth of each tree exceeds a certain threshold. This particular hyperparameter acts as a regularizer and performance contributor, moderately increasing the depth of each tree improves both fitting and generalizability, while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting and large training time. In other words, increasing tree depth decreases bias at the cost of increased variance.

- Bagging temperature ('bagging_temperature'):

$$[0.1, 10]$$

. CatBoost uses Bayesian bootstrap aggregation for regression problems. If bagging temperature is 0, the weights assigned to the sampled subspaces are equal to 1. If bagging temperature increases, the intensity of bootstrap increases and the weights are sampled from exponential distribution. CatBoost uses minimum variance sampling or weighted sampling at the tree level and not the split level.

- Grow policy ('grow_policy'): Symmetric Tree, Depthwise Tree and LossGuide. Specifies how the trees will be generated from the leaves. Unlike LightGBM which uses leaf-wise tree growth (best-first) to allow for an imbalance tree (may cause overfitting for small datasets) regardless of the depth of each tree, Catboost grows a balanced tree such that at each level, the feature-split pair that minimizes the loss function is selected and utilized for all level nodes.

  - Symmetric Tree (ST): Level by level growth strategy; on each iteration, all leaves from the last tree level are split with the same condition.
  - Depthwise Tree (DT): Level by level growth strategy; on each iteration, all non-terminal leaves from the last tree level are split depending on the feature-split pair that minimizes the loss function.
  - LossGuide (similar to LightGBM) (LG): Leaf by leaf growth strategy; on each iteration, non-terminal leaf with the best loss improvement is split.

- Score function ('score_function'): Cosine, L2. This is used only for symmetric or depthwise growing policies. Score function is used to choose between candidate trees when adding a new tree to the ensemble. L2 and Cosine are first-order score functions.

Two different iterations of the CatBoost method is carried out:

Depthwise was crashing for Catboost and hence that value is avoided in the hyperparameter grow_policy. Following are the range of different hyperparameters specific to the two different pipelines of the CatBoost Process:

Grow_Policy- Lossguide

```
'colsample\_bylevel': 0.5 to 1 in steps of 0.1
'num\_trees': 10 to 4000 in steps of 100
'l2\_leaf\_reg': 0.0001 to 10\^4
'num\_leaves': 20 to 1000 in steps of 10
'max\_depth': 1 to 16 in steps of 2
'bagging\_temperature': 0.1 to 10 in steps of 1
'grow\_policy': 'Lossguide'
```

Grow_Policy- Symmetric Tree

```
'colsample\_bylevel': 0.5 to 0.9 in steps of 0.1
'num\_trees': 10 to 1000 in steps of 100
'l2\_leaf\_reg': 0.0001 to 10\^4
'max\_depth': 1 to 16 in steps of 2
'bagging\_temperature': 0.1 to 10 in range of 1
'grow\_policy': 'SymmetricTree'
'score\_function': 'Cosine','L2'
```

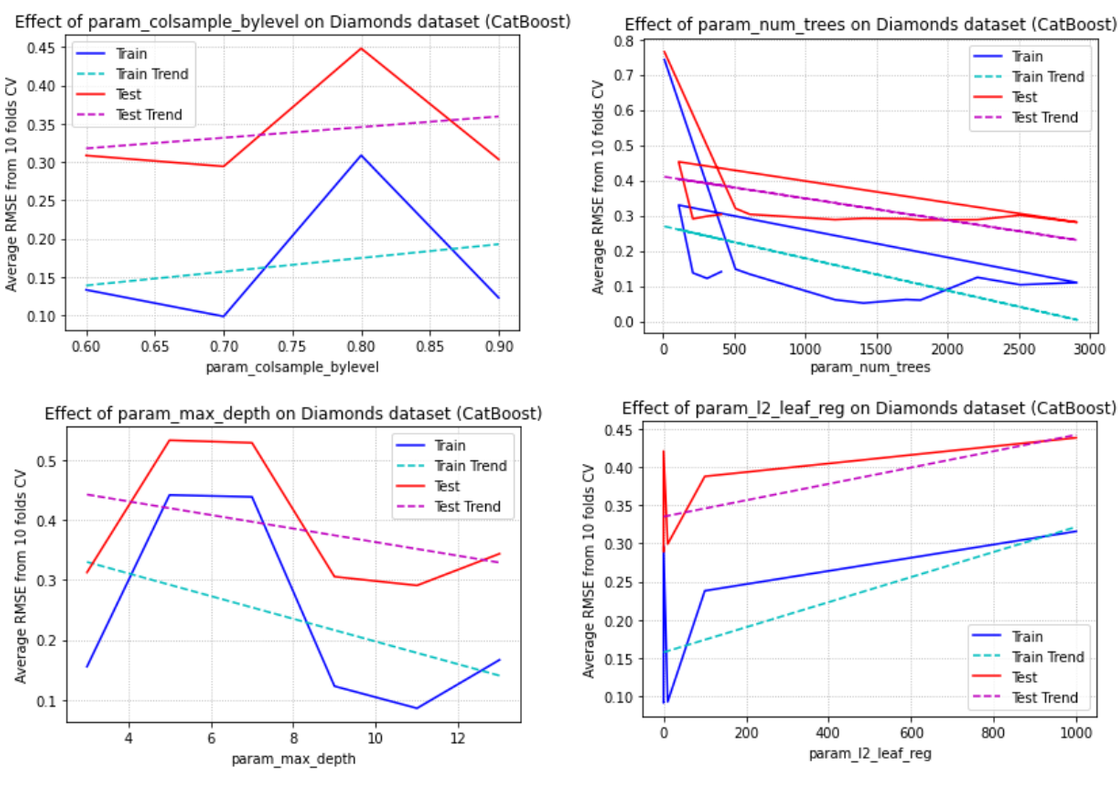


Figure 40: CatBoost Hyperparameters impact on the Diamond dataset

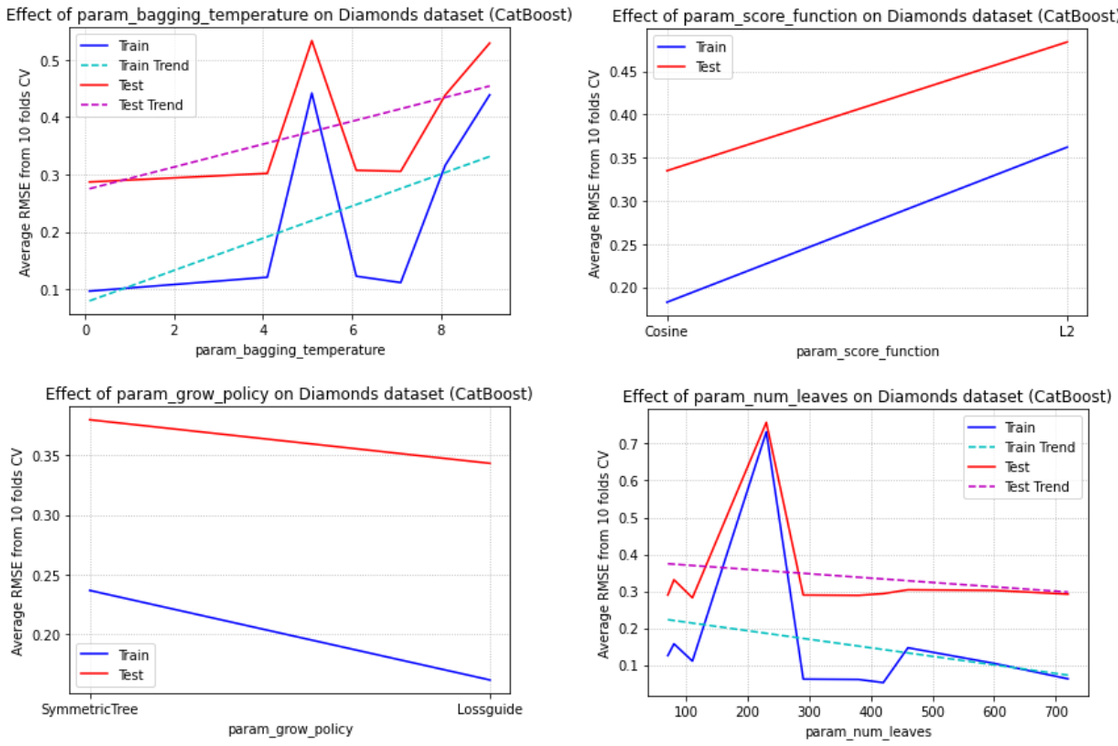


Figure 41: CatBoost Hyperparameters Impact on the Diamond dataset

Question 23

Apply Bayesian optimization using skopt.BayesSearchCV from scikit-optmize to search good hyperparameter combinations in your search space. Report the best hyperparameter found and the corresponding RMSE, for both algorithms.

In Bayesian optimization, a Gaussian process is utilized to approximate the objective function, which allows priors on the distribution of moments such as mean and uncertainty to propagate in the forward direction as the search process progresses. The acquisition function dictates the next set/iteration of hyperparameters to sample from the design space with Bayesian Upper-Confidence Bounds (UCB) using Monte Carlo sampling method , which is also known as the Thompson sampling, which balances the exploration and exploitation tradeoff. Thompson sampling ensures that the acquisition function does not get stuck in a local optimum very early in the search process, with the exploration parameter continuously decreased as the confidence in the Pareto-frontier tends to grow. As part of the 10 Fold Cross validation, We used 10 iterations for the CatBoost regressor, and 20 iterations for the LightGBM regressor. The provided result highlights the best group of hyperparameters obtained for both the regressors. We observe that lightGBM offers slightly better performance than CatBoost in terms of average train and test RMSE on the diamond dataset. However, there is no significant/appreciable performance difference among the two classifiers. LightGBM is more robust to overfitting than catboost, possibly due to the presence of more regularization hyperparameters than CatBoost.

Depthwise was crashing for Catboost and hence that value is avoided in the hyperparameter grow_policy.

mean_test_score	mean_train_score	param_boosting_typ	param_num_leaves	param_max_depth	param_n_estimators	param_reg_alpha	param_reg_lambda	param_subsample	param_subsample_fr	param_min_split_ga
-0.2788034986	-0.1250500826	gbdt	750	21	610	1	1000	0.7	15	0.001

Figure 42: LGBM Best Hyperparameters on the Diamond dataset

```
mean_test_score = 0.2788034986
mean_train_score = 0.1250500826
boosting_type = gbdt
num_leaves = 750
max_depth = 21
n_estimators = 610
reg_alpha = 1
reg_lambda = 1000
subsample = 0.7
subsample_freq = 15
min_split_gain = 0.001
```

mean_test_score	mean_train_score	param_colsample_b	param_num_trees	param_max_depth	param_l2_leaf_reg	param_num_leaves	param_bagging_tem	param_grow_policy
-0.2822963091	-0.1108662923	0.6	2910	9	100	110	7.1	Lossguide

Figure 43: CatBoost Best Hyperparameters on the Diamond dataset

```
mean_test_score = 0.2822963091
mean_train_score 0.1108662923
```

```
colsample_bylevel = 0.6
num_trees = 2910
max_depth = 9
l2_leaf_reg = 100
num_leaves = 110
bagging_temperature = 7.1
grow_policy = Lossguide
```

Question 24

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? Endorse your interpretation with numbers and visualizations.

Effect of LightGBM hyperparameters on the Diamond dataset:

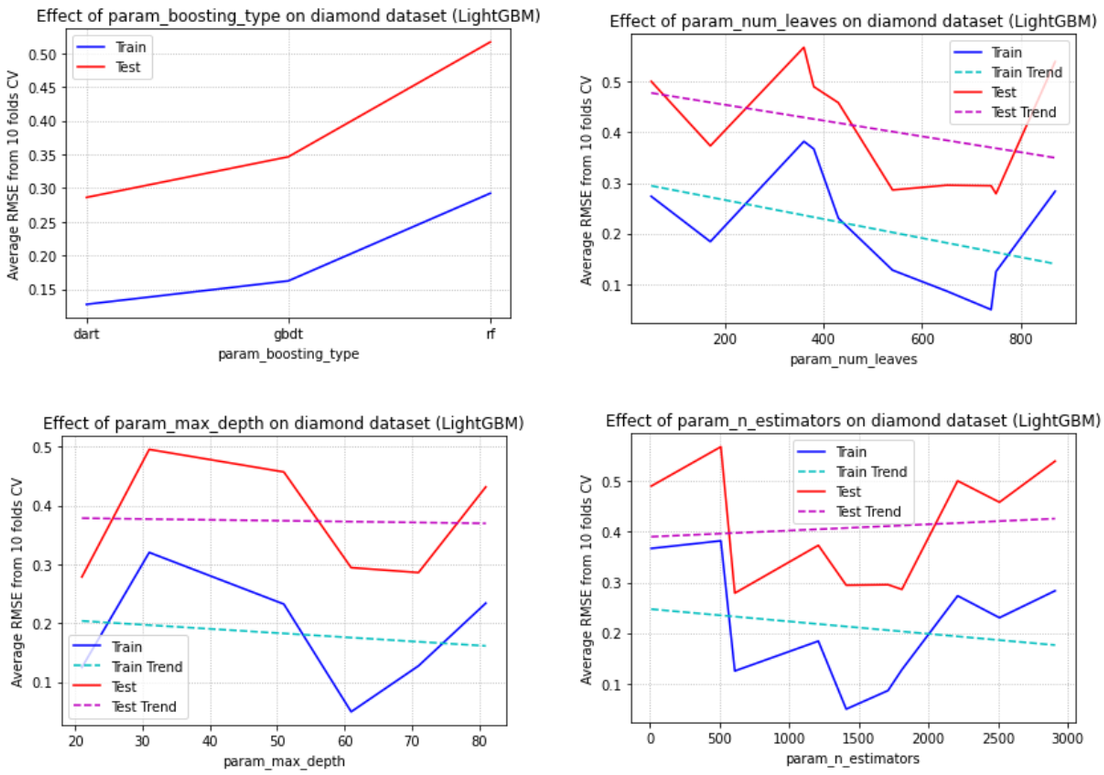


Figure 44: LightGBM Hyperparameters impact on the Diamond dataset

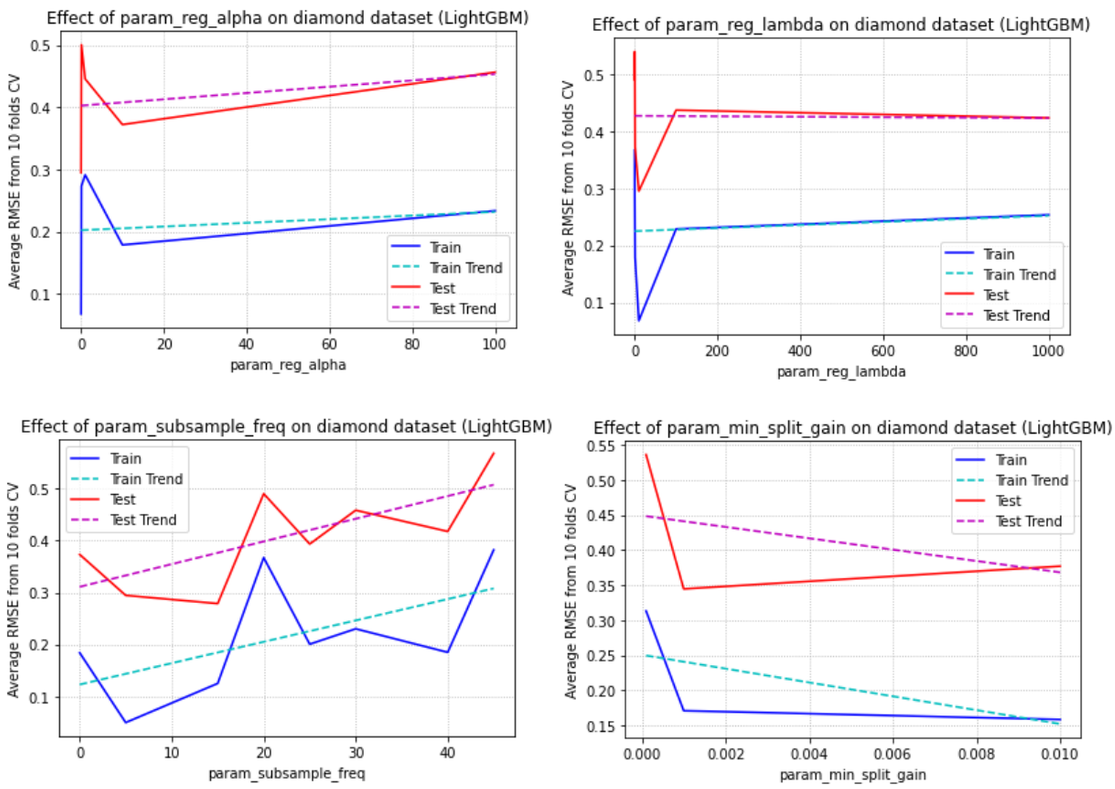


Figure 45: LightGBM Hyperparameters Impact on the Diamond dataset

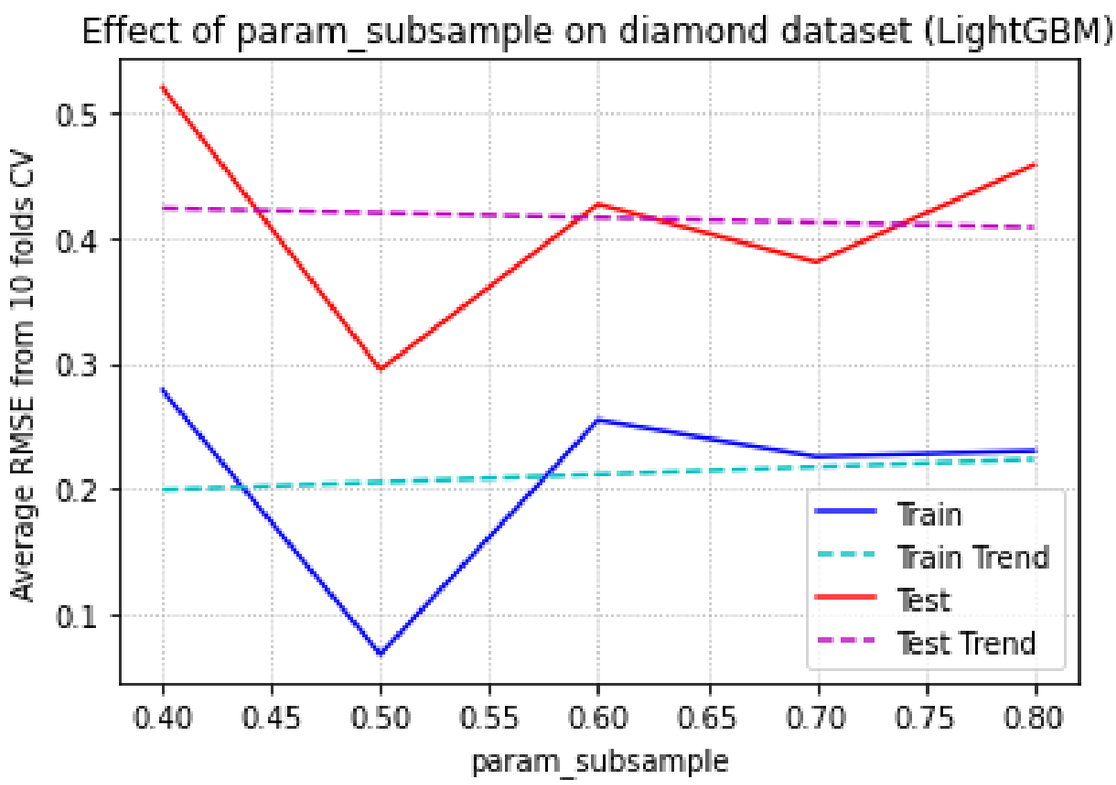


Figure 46: LightGBM Hyperparameters Impact on the Diamond dataset

- Boosting Type: From the plots, it is evident that GBDT and RF perform similar to each other, while together both these boosting types jointly outperform DART. Theoretically, we can expect DART to improve generalizability of GBDT by introduction of dropout. However, DART also has implicit additional hyperparameters that require tuning, such as model seed, the probability to skip dropout during an iteration,

dropout rate and the decision to use XGBoost dropout vs uniform dropout. We did not tune between these two cases due to compute constraints. Large number of floating parameters can result in unexpected performance of DART.

- Number of leaves ('num_leaves' or 'max_leaves'): Similar to the case of depth of each tree, the number of leaf nodes acts both as a performance contributor and regularizor, moderate values improve both generalizability and fitting, while very large values lead to overfitting by assuming each data point including noise in the dataset as a distinct leaf in the tree. In the corresponding figure, we can observe a mild improvement in performance as the number of leaves goes up.

- Depth of each tree: From the plots, we see that increasing the depth of each tree enhances and optimizes the error rate in case of both the training and test sets. This particular hyperparameter acts both as a regularizer and a performance contributor, moderately raising the depth of each tree enhances both fitting and generalizability, while very high values of depth of each tree might eventually lead to the overfitting phenomenon(as seen in the plot) by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting and large training time. Briefly, increasing tree depth decreases bias at the cost of increased variance.

- Number of trees— Number of Estimators: From the plots, we do not see any significant performance gains or losses as the number of trees increases. Increasing the number of trees may improve and stabilize model performance non-monotonically (as could be seen clearly from the plots). The expected error rate for a boosted tree ensemble might take a form of non-monotonous function of the number of trees, being noisy once a sufficiently high number of required estimators have been utilized. Once all other hyperparameters fixed, the only effect the number of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The overall decision of the ensemble converges towards a mean value with poor returns decaying gradually when the number of trees approach infinity (provided by Jensen's inequality). Too few trees, however, will hurt model performance (weak law of large numbers- WLLN). Thus, unless the number of trees is very small, increasing the number of trees has no effect on the model performance.

- L1 regularization term: From the plots, we observe that both train and test RMSE initially improves with finite values of Lasso regularization, but performance drops when aggressive regularization is performed. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance.

- L2 regularization term: From Figure 38 (7), we observe that both train and test RMSE initially improves with finite values of Tikhonov regularization, but performance drops when aggressive regularization is performed. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance. Note that L1 regularization is a stronger regularization and causes more aggressive regularization than L2, as L1

regularization zeros out weights more often than L2 regularization. From more details on L1 and L2 regularization, we urge the grader to look at Question 11.

- Bagging frequency: From the plots, we see that as bagging is conducted more often, the performance drops. This is expected because bagging acts as a regularizer. More frequent bagging (dropping samples) drops a subset of training data more frequently, which can cause important training samples to never be considered during training.

- Bagging fraction: From the plots, we observe that as subsampling ratio increases, the performance also increases monotonically. Decreasing subsampling ratio helps with overfitting and generalization issues and acts as a regularizer; very small values of bagging fraction may hurt model performance while very large values (no bagging) improves model performance on the training set but generalizes poorly on the test set. Bagging aims at reducing the variance of the ensemble.

- Minimal gain to perform split: From the given plots, we see that the both train RMSE and test RMSE improves marginally or slightly with an increase in the ratio of gain. Increasing this ratio acts as a regularizer, informing the ensemble to neglect very small training performance improvements that do not have meaningful significant impacts on generalizability of the model. Although, the graph does not convey the full picture as the Bayesian agent did not cover the entire range of the ratio. Higher aggressive bagging is likely to cause drop on model performance thanks to regularization effects.

Helps performance / Faster Speed:

1. Grow shallower trees.

- Decrease max_depth: This parameter is an integer that controls the maximum distance between the root node of each tree and a leaf node.

- Decrease num_leaves. It isn't straightforward to use max_depth alone to limit the complexity of trees. The num_leaves parameter sets the maximum number of nodes per tree. Decrease num_leaves to reduce training time.

- Increase min_split_gain. When adding a new tree node, LightGBM chooses the split point that has the largest gain. Gain is basically the reduction in training loss that results from adding a split point. By default, LightGBM sets min_split_gain to 0.0, which means "there is no improvement that is too small". However, in practice you might find that very small improvements in the training loss don't have a meaningful impact on the generalization error of the model. Increase min_split_gain to reduce training time.

2. Grow less trees.

- Decrease num_iterations: The num_iterations parameter controls the number of boosting rounds that will be performed. Since LightGBM uses decision trees as the learners, this can also be thought of as "number of trees".

- Some hyperparameters we have not considered here that can help reduce tree count are using early stopping, consider fewer splits, enable pre-filtering etc.

3. Use less data:

- Subsampling ratio or bagging fraction can help. By default, LightGBM uses all observations in the training data for each iteration. It is possible to instead tell LightGBM to randomly sample the training data. This process of training over multiple random samples without replacement is called "bagging". Set subsample_freq to an integer greater than 0 to control how often a new sample is drawn. For example, subsample_freq: 5, bagging_fraction: 0.75 tells LightGBM "re-sample without replacement every 5 iterations, and draw samples of 75% of the training data". Decrease subsample_freq to reduce training time.

Helps in regularization:

- Use small num_leaves. More leaves can overfit. Therefore decrease the leaf count to regularize. This can be observed from the plots. As num_leaves increases, the training and testing accuracies decrease but the gap between them increases (generalization gap increases).
- Subsampling ratio or bagging fraction can help. By default, LightGBM uses all observations in the training data for each iteration. It is possible to instead tell LightGBM to randomly sample the training data. This process of training over multiple random samples without replacement is called "bagging". Set subsample_freq to an integer greater than 0 to control how often a new sample is drawn. Increasing the bagging seems to shrink the generalization gap as the plot shows.
- Try decreasing max_depth to avoid growing deep tree. As the plot shows, decreasing the depth, decreases overfitting.
- Try increasing the n_estimators. More estimators act like a regularizer.
- As expected the reg_alpha and reg_lambda parameters add regularization and hence increasing them helps shrink the generalization gap.
- Other potential methods we did not experiment yet are: 1. Use small max_bin. 2. Use min_data_in_leaf and min_sum_hessian_in_leaf. 3. Use bigger training data. 4. Try increasing path_smooth.

Affects fitting efficiency / Better Accuracy:

- Use large num_leaves (may cause over-fitting). From the plot we can see how the training and test accuracy fall from number of leaves of 100 or 200 to 600.
- Try dart. From the plot, it is evident that dart has much lower RMSE than any other boosting type.
- Other potential methods not experimented: 1. Use large max_bin. 2. Use small learning_rate with large num_iterations. 3. Use bigger training data.

Effect of CatBoost Hyperparameters on the Diamond dataset:

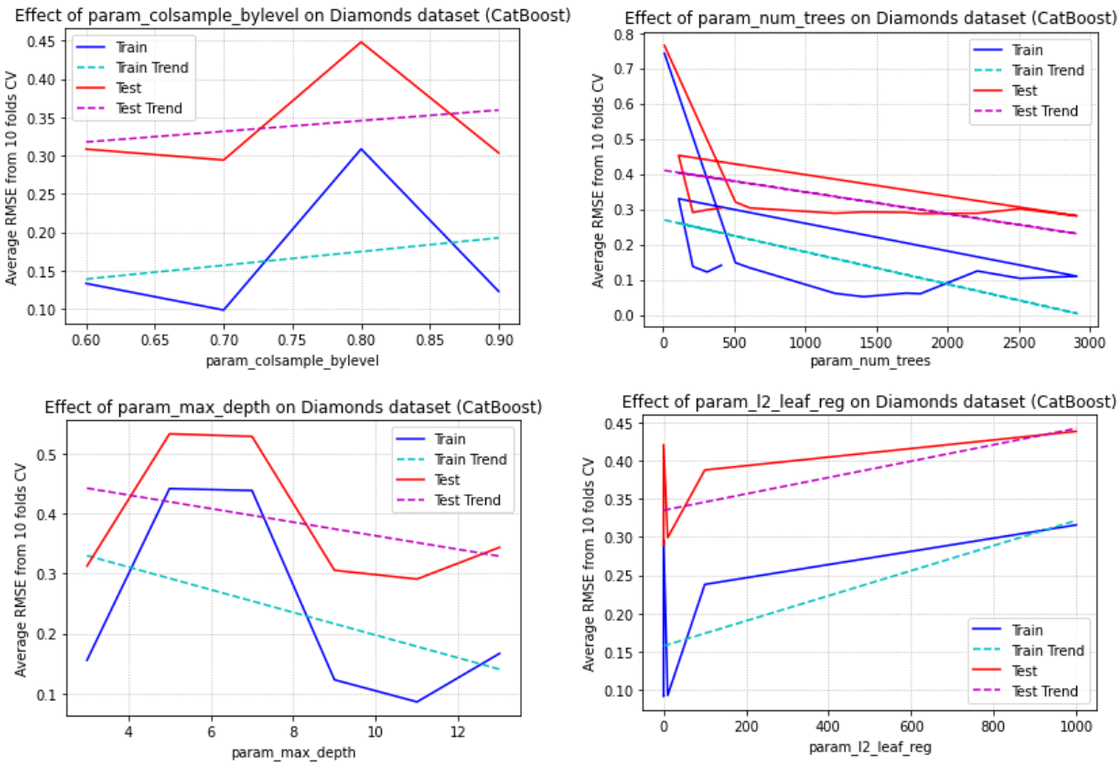


Figure 47: CatBoost Hyperparameters impact on the Diamond dataset

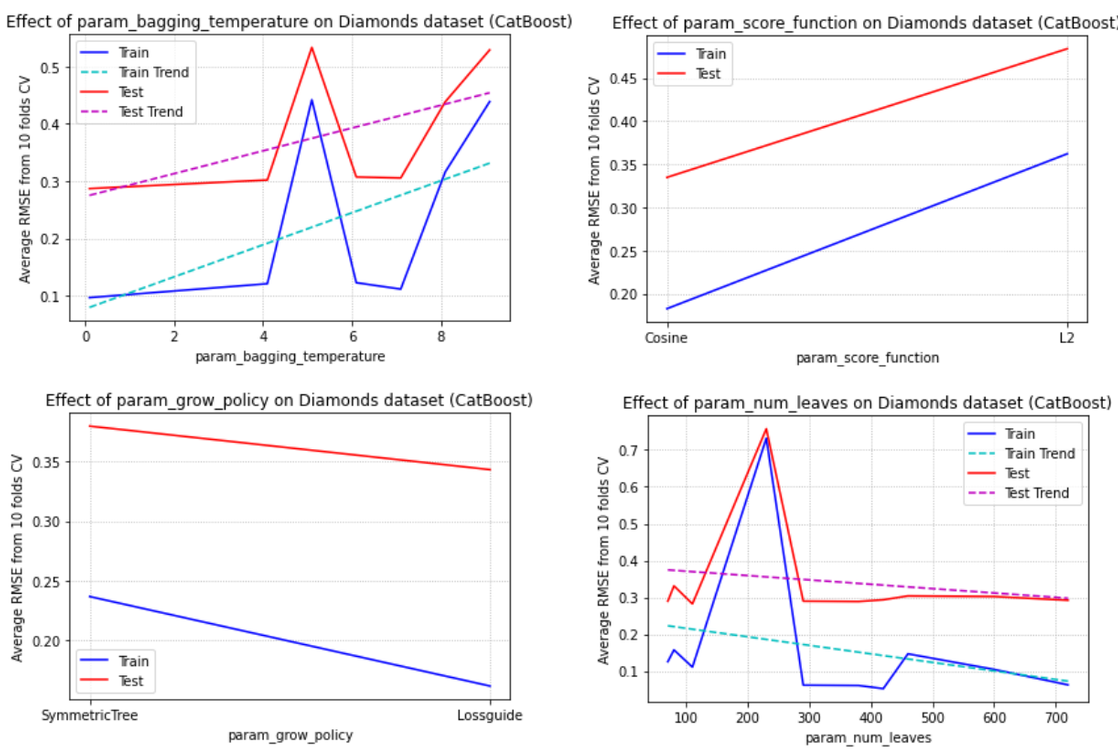


Figure 48: CatBoost Hyperparameters Impact on the Diamond dataset

- Bagging temperature: From the plots we see that as bagging temperature rises, the train and test RMSE rises. If the bagging temperature is at a value of 0, the weights being assigned to the sampled subspaces are equal to 1. If bagging temperature increases, the intensity of bootstrap increases and the weights are sampled from exponential distribution. Since, we are already providing top 10 salient features to CatBoost, assigning extreme weights to these features can hurt model performance.

- Feature fraction: From the plots we see that, we see that as feature fraction increases, the performance improves as well. Decreasing feature fraction helps with overfitting and generalization issues by selecting only the most salient and important features that have an impact on most of the samples, acting as a regularizer; very small values of feature fraction may hurt model performance while very large values (no random subspace) improves model performance on the training set but generalizes poorly on the test set. Random subspace aims at reducing the variance of the ensemble.

- Grow policy: From the plots we see that, we observe that symmetric growth policy performs better than both depthwise and lossguide growth policies. Lossguide uses leaf-wise tree growth (best-first) to allow for an imbalance tree, which usually causes overfitting when the feature space is small. In addition, symmetric tree considers all leaves from the last iteration instead of just the non-terminal leaves for making splits, thus having a more generalized overview of the earlier feature subspaces than depthwise tree.

- L2 regularization: From the plots we see that, we observe that overall, more aggressive regularization leads to performance drops. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance.

- Depth of each tree: From the plots we see that, we see that increasing the depth of each tree improves the error rate for both the training and test sets slightly. This particular hyperparameter acts both as a regularizer and a performance contributor, moderately increasing the depth of each tree improves both fitting and generalizability, while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting and large training time. In other words, increasing tree depth decreases bias but increases variance.

- Number of leaves (only for lossguide): From the plots we see that, we observe that similar to the depth of each tree, the number of leaf nodes acts as a regularizer and performance contributor, moderate values improve both fitting and generalizability, while very large values lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. Figure 39 (6), however, shows no significant performance changes with changes in number of leaves, which is probably because other hyperparameters rendered the leaf hyperparameter change contribute insignificantly towards the final performance.

- Number of trees: From the plots we see that, we see slight performance gains as the number of trees increases. Increasing the number of trees may improve and stabilize model performance non-monotonically. The expected error rate for a boosted tree ensemble is a non-monotonous function of the number of trees, being noisy once a sufficiently large number of required estimators have been used. With all other hyperparameters fixed, the only effect the number of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The overall decision of the ensemble converges towards

a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). Too few trees, however, will hurt model performance (weak law of large numbers). Thus, unless the number of trees is very small, increasing the number of trees has no effect on the model performance.

- Score function: From the plots we see that, we see that cosine distance performs better as a score function than L2 distance. This is expected because cosine similarity is not affected by the magnitude of the features, meaning that the range and scale of the features does not affect the distance metric, but rather it associates features based on angle between sample points. This corrects the effects of non-standardized or unscaled features. In addition, in high dimensions, the rapid increase in volume causes the data to become sparse in each dimension, causing the Euclidean distances to converge to a constant value between all sample points. Since all feature points become equidistant, models working with L2 distances cannot find distinguishable features within the data. Thus, for textual clustering, cosine similarity is the ideal metric over L2 distances

Helps performance / Faster Speed:

- Decrease max_depth: This parameter is an integer that controls the maximum distance between the root node of each tree and a leaf node.
- Decrease num_leaves. It isn't straightforward to use max_depth alone to limit the complexity of trees. The num_leaves parameter sets the maximum number of nodes per tree. Decrease num_leaves to reduce training time.
- For datasets with hundreds of features colsample_bylevel parameter speeds up the training and usually does not affect the quality. It is not recommended to change the default value of this parameter for datasets with few (10-20) features. For example, set the parameter to 0.1. In this case, the training requires roughly 20% more iterations to converge. But each iteration is performed roughly ten times faster. Therefore, the training time is much shorter even though the resulting model contains more trees.
- Subsampling ratio or bagging fraction can help. By default, CatBoost uses all observations in the training data for each iteration. It is possible to instead tell CatBoost to randomly sample the training data.
- By default, CatBoost uses symmetric trees, which are built if the growing policy is set to SymmetricTree. Such trees are built level by level until the specified depth is reached. On each iteration, all leaves from the last tree level are split with the same condition. The resulting tree structure is always symmetric. Symmetric trees have a very good prediction speed (roughly 10 times faster than non-symmetric trees) and give better quality in many cases. However, in some cases, other tree growing strategies can give better results than growing symmetric trees. Try to analyze the results obtained with different growing trees strategies. Specifics: Symmetric trees, that are used by default, can be applied much faster (up to 10 times faster).
- Use GPU's and try to decrease the number of iterations. Increase the learning rate.
- By default, the boosting type is set to for small datasets. This prevents overfitting but it is expensive in terms of computation. Try to set the value of this parameter to speed up the training.

Helps in regularization:

- Use small num_leaves. More leaves can overfit. Therefore decrease the leaf count to regularize. This can be observed from the plots. As num_leaves increases, the training and testing accuracies decrease but the gap between them increases (generalization gap increases).
- Subsampling ratio or bagging fraction can help. By default, CatBoost uses all observations in the training data for each iteration. It is possible to instead tell CatBoost to randomly sample the training data.
- Try decreasing max_depth to avoid growing deep tree. As the plot shows, decreasing the depth, decreases overfitting.
- Try increasing the num_trees. More estimators act like a regularizer.
- As expected the l2_leaf_reg parameter add regularization and hence increasing them helps shrink the generalization gap.

Affects fitting efficiency / Better Accuracy:

- Use large num_leaves (may cause over-fitting). From the plot we can see how the training and test accuracy fall from number of leaves of 100 or 200 to 600.
- Try lossguide. From the plot, it is evident that lossguide has much lower RMSE than any other grow policy type.
- Decreasing the bagging_temperature seems ti improve accuracy.
- Other potential methods not experimented: 1. Use small learning_rate with large num_iterations. 2. Use bigger training data.

Question 25

Perform 10-fold cross-validation and measure average RMSE errors for training and validation sets. Why is the training RMSE different from that of validation set?

In this question, we are asked to perform 10-fold cross-validation and measure average RMSE errors for training and validation sets for all models and all datasets. We already performed this task earlier and here, we report the performance of the best model (hyperparameter tuned either using 10-fold grid search cross validation or Bayesian optimization) for each dataset. The hyperparameter values are mentioned in the previous questions or in the final results attached in the appendix. The best model is polynomial regression for both datasets.

Diamond Dataset:

Model	**Mean Train score**	**Mean Test score**
Linear Regression	0.3125287816	0.300928763
Linear Regression(Lasso)	0.312531704	0.3010687817
Linear Regression(Ridge)	0.3125287816	0.300928765
Polynomial Regression	0.1823719903	0.1856840557
Neural Networks	0.1243414139	0.14728662687
Random Forests	0.1158789838	0.1909485699
LightGBM	0.1250500826	0.2788034986
CatBoost	0.1108662923	0.2822963091

Table 4: Performance summary of Diamond Dataset

Gas Emission Dataset:

Model	**Mean Train score**	**Mean Test score**
Linear Regression	0.6373622583	0.6495247607
Linear Regression(Lasso)	0.6373622583	0.6495247607
Linear Regression(Ridge)	0.6341046523	0.6543662606
Polynomial Regression	0.5273179766	0.5602573924
Neural Networks	0.5744953489	0.6295302481
Random Forests	0.4581012859	0.5806434108

Table 5: Performance summary of Gas Emission Dataset

We see that usually, the training RMSE is lower than the test (validation) RMSE. This happens because complex models tend to overfit on the characteristics of the training set, which may not be present in the validation set and lead to models not generalizing well on the validation set. This can be observed in almost all models except for linear regression model of diamond dataset. That is an example of underfitting. But in all other cases, we observe overfitting. This is especially true for the more complex models of LightGBM and CatBoost, where the generalization gap between the train and test scores are huge. In Polynomial regression and Neural Networks (MLP), we observe not much difference between the train and test errors. This is the optimal balance where the model is complex enough to

learn the features and not too complex enough to overfit to the training data and hence is able to generalize well to the test/validation data as well.

When you are creating a predictive model, what actually we are doing is create the model that captures the signal not the noise of the data. RMSE of training of model is a metric which measure how much the signal and the noise is explained by the model. So when you add more variables to a model, the model become more "flexible", it captures the pattern of training data very well and reduces RMSE to a smallest amount, however because our statistical learning procedure is working too hard to find patterns in the training data, and may be picking up some patterns that are just caused by random chance rather than by true properties of the unknown function f which you are trying to estimate . When you overfit the training data, the test MSE will be very large because the supposed patterns that the method found in the training data simply don't exist in the test data.

For all the grid searches, we evaluate both training and validation RMSE as is reported above.

These two metrics can be significantly different, since as we increase the model complexity, the training RMSE will decrease monotonically. However, at last we will the point where overfitting problem occurs. In general, as training RMSE decreasing monotonically, validation RMSE tends to decrease first and then increase. All the fine-tuning procedures we do above is to find the optimal model structure that gives us the minimal validation RMSE.

Question 26

For random forest model, measure "Out-of-Bag Error" (OOB) as well. Explain what OOB error and R2 score means given this link.

- In a random forest, each tree and its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN), because the trees are grown using a randomization technique on their individual bootstrap subsamples, which are uncorrelated with the growth of other trees. Each decision tree is trained on distinct and independent random subspace, resulting in variation, decorrelation, and diversification across individual trees. The number of accurately predicted samples (rows) from the out of bag sample rows via majority decision fusion is defined as the OOB score, which is a strategy for verifying random forest models. A row in the original dataset that was not present in the bootstrap samples of a given decision tree is characterized as an out-of-bag sample. Because OOB is derived on unseen test data, it functions as a validation score for the ensemble.
  Or in other words, Because we bootstrap the training dataset, for one specific tree, certain data points may not be picked for tree construction, and the RMSE for that data point is referred to as "Out-of-Bag Error." As a result, we don't require a separate validation dataset for the random forest regression model, and this OOB error may be used to evaluate validation performance.

- R^2, on the other hand, gives the coefficient of determination for the trained model on the supplied training dataset. R^2 (coefficient of determination) is defined as the ratio of the target variable's variance that can be predicted by the characteristics in the dataset (also known as goodness of fit). It comprises all of the data in the training (and validation) set, regardless of whether a tree within an ensemble saw part of the samples or not, as well as all of the decision trees' decisions. Separate processes are required for R^2 score calculation for training and validation scores.

- In a cross-sectional data set (no time series or panel data), the OOB estimate of true performance of a random forest is usually very accurate and can sometimes even replace (cross-)validation. Put differently, you can trust the OOB accuracy in such cases. This is in constrast to the insample (training set) accuracy: By construction, random forests tend to extremely overfit on the training data because the individual trees are usually very deep and unstable. So don't get lured by an insample accuracy/R^2 of 97%.

- As compared to the validation score OOB score is computed on data that was not necessarily used in the analysis of the model. Whereas for calculation validation score, a part of the original training dataset is actually set aside before training the models. Additionally, the OOB score is calculated using only a subset of Decision Trees not containing the OOB sample in their bootstrap training dataset. While the validation score is calculated using all the Decision Trees of the ensemble.

- As mentioned above, only a subset of Decision Trees is used for determining the OOB score. This leads to reducing the overall aggregation effect in bagging. Thus in general, validation on a full ensemble of DTs is better than a subset of Decision Tree for estimating the score. However, occasionally the dataset is not big enough and hence

set aside a part of it for validation is unaffordable. Consequently, in cases where we do not have a large dataset and want to consume it all as the training dataset, the OOB score provides a good trade-off. Nonetheless, it should be noted that validation score and OOB score are unalike, computed in a different manner and should not be thus compared.

- R^2 (coefficient of determination) regression score function. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a score of 0.0.

OOB Scores for both the datasets are given below:

- **OOB Score for the Diamond Dataset:**
  OOB Score for Diamond Dataset: 0.9569190066991791
- **OOB Score for the Gas Emission Dataset:**
  OOB Score for GT Dataset: 0.7220917422471709

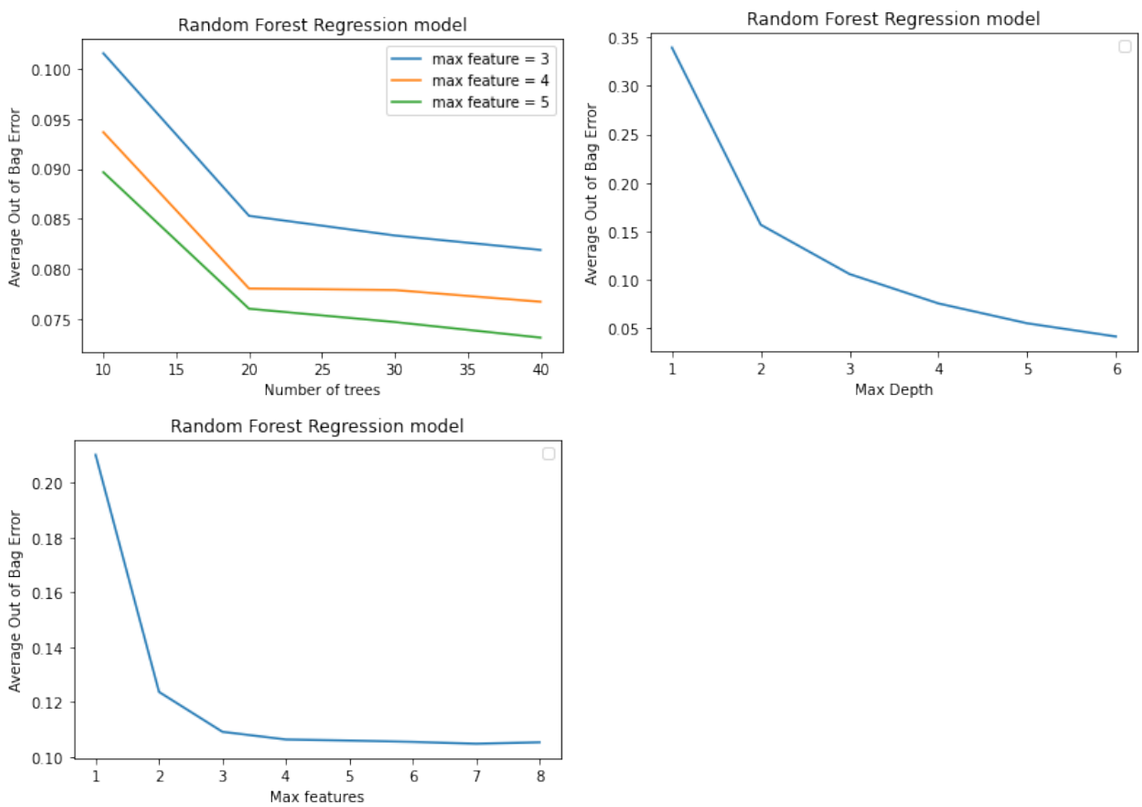


Figure 49: OOB Error as a function of Random Forest Hyperparameters for Diamond Dataset.

OOB Errors for both the datasets are given below:

- **OOB Error for the Diamond Dataset:**
  OOB Error for Diamond Dataset: 0.043081
- **OOB Error for the Gas Emission Dataset:**
  OOB Error for GT Dataset: 0.277908257752

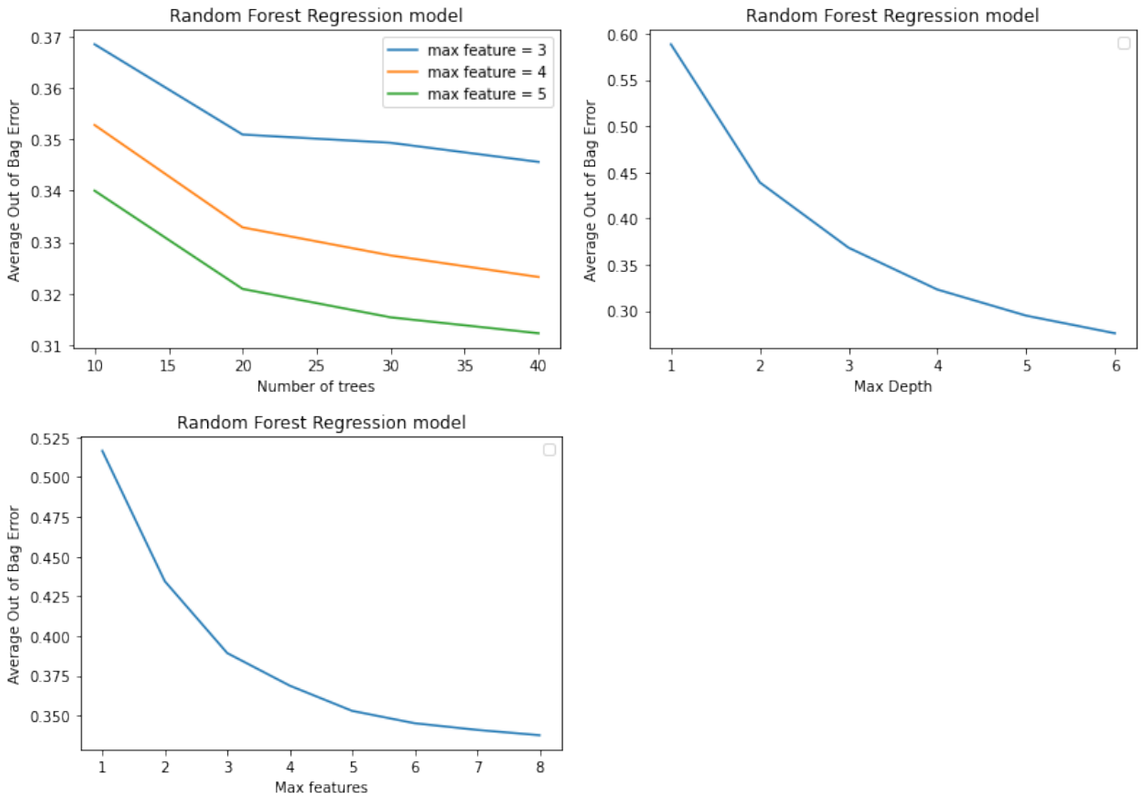


Figure 50: OOB Error as a function of Random Forest Hyperparameters for Gas Emission Dataset.

Question 27

Download the training tweet data. 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:

- **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**

We examine the prediction of future popularity of a subject or event using Twitter data in this article. The data is gathered by searching prominent hashtags connected to the 2015 Super Bowl from two weeks before the game to one week after the event. Our dataset contains the hashtags #gohawks, #gopatriots, #nfl, #patriots, #sb49, and #superbowl.
We summarized some basic statistics for each hashtag below. The statistics include the "average number of tweets per hour", "average number of followers of users per tweet" and "average number of retweets per tweet".
Results

tweets_#gohawks.txt

Average number of tweets per hour: 292.48785062173687
Average number of followers of users posting the tweets per tweet: 2217.9237355281984
Average number of retweets per tweet: 2.0132093991319877

---

tweets_#gopatriots.txt

Average number of tweets per hour: 40.954698006061946
Average number of followers of users posting the tweets per tweet: 1427.2526051635405
Average number of retweets per tweet: 1.4081919101697078

---

tweets_#nfl.txt

Average number of tweets per hour: 397.0213901819841
Average number of followers of users posting the tweets per tweet: 4662.37544523693
Average number of retweets per tweet: 1.5344602655543254

---

tweets_#patriots.txt

Average number of tweets per hour: 750.8942646068899
Average number of followers of users posting the tweets per tweet: 3280.4635616550277

Average number of retweets per tweet: 1.7852871288476946

---

tweets_#sb49.txt

Average number of tweets per hour: 1276.8570598680474
Average number of followers of users posting the tweets per tweet: 10374.160292019487
Average number of retweets per tweet: 2.52713444111402

---

tweets_#superbowl.txt

Average number of tweets per hour: 2072.11840170408
Average number of followers of users posting the tweets per tweet: 8814.96799424623
Average number of retweets per tweet: 2.3911895819207736

Among the six hashtags, it is clear that #sb49 and #superbowl have greater values for all three statistics. The explanation for this might be that these two hashtags are more generic and everyone wants to tag them. However, hashtags such as #gohawks, #gopatriots, and #patriots would only be tagged by their fans, who were a limited number of individuals.

Furthermore, because the Seattle Seahawks were the reigning champions, the hashtag #gohawks was more popular than #gopatriots, as seen by the three data. But when the New England Patriots ultimately won, the hashtag #patriots popped and the hashtag #hawks did not.

Question 28

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.

We plotted the Histograms of "number of tweets per hour over time" for all six hashtags in this query. And the histograms for the #Superbowl and #NFL are shown in the figures below.

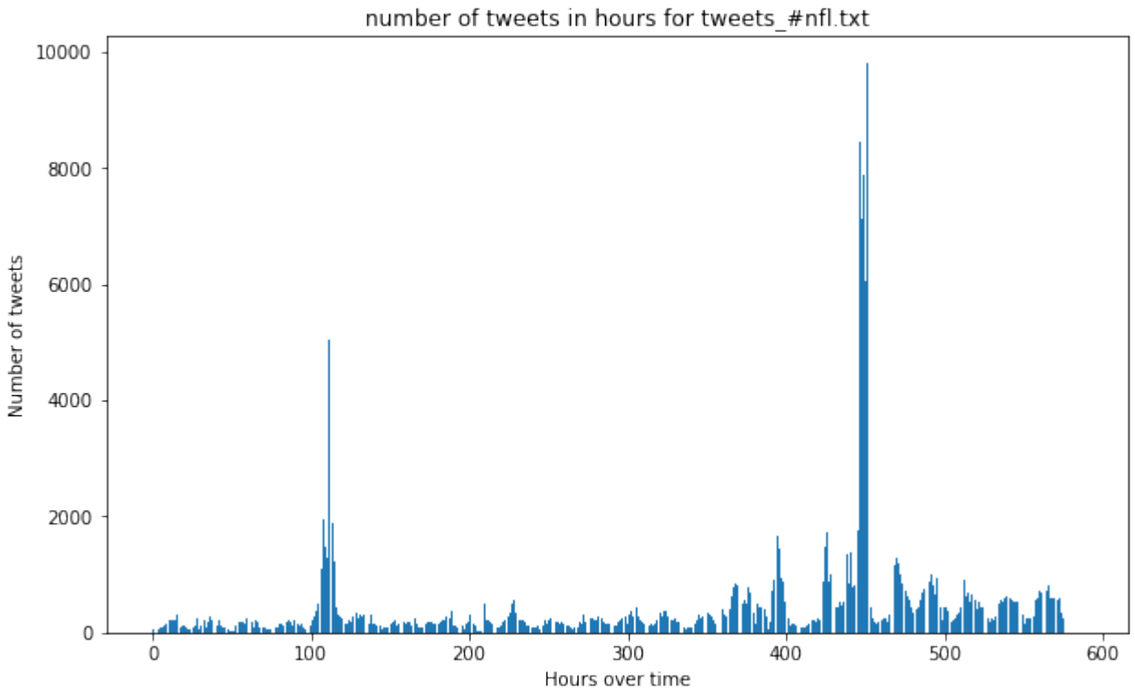


Figure 51: Histogram of number of tweets in hour for #NFL

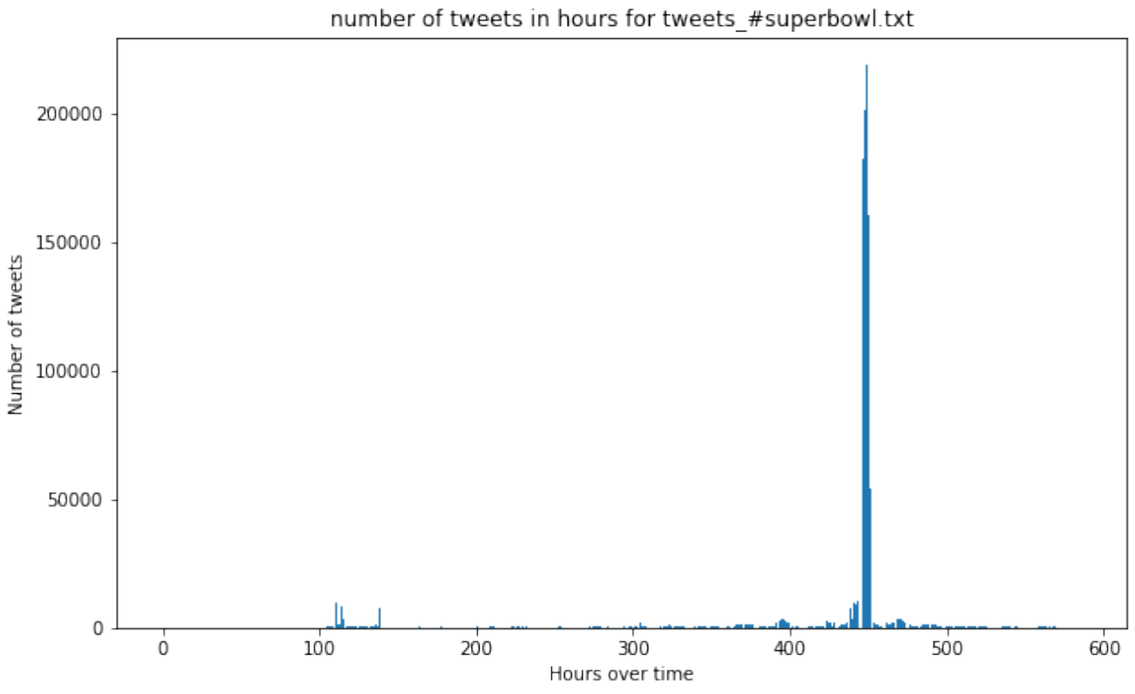


Figure 52: Histogram of number of tweets in hour for #Superbowl

The histograms above depict the evolution of two hashtags, #superbowl and #NFL. They

have varied lifetime patterns, yet they both peak around the 450th hour. Because the Twitter data is collected from two weeks before the game to one week after the game, the peak time corresponds perfectly to the time the Superbowl took place.
Furthermore, there are some comments concerning the #NFL at other times, as well as a tiny surge around the 100th hour, which might coincide to the NFL conference finals game time. However, the debate surrounding the #superbowl was centered on the exact time it took place.

Question 29

Twitter is a social media site where people may express their views, whether positive or negative, about big sporting, political, or natural catastrophe events. The dataset on which we are working is football twitter data from the Super Bowl between the Patriots of Massachusetts and the Seahawks of Seattle.

Project Details:

In **Part 1**, we have explored the data and tried to get a better understanding of the popular terms in each hashtag. In **Part 2, Part 3, Part 4** we have tackled various tasks. In each task, we have described the feature engineering process and the baseline ML model. Moreover, we have done a thorough evaluation and reported the results along with the graphs.

Part 1 - Word Cloud Solution:

In the backdrop of the Twitter picture, we used a word cloud solution to show the trending direction of tweets from multiple datasets. We began by looking at @ (mentions) in all aggregated tweet data to discover what received the most mentions across all tweets. The larger the term, the higher the frequency, as with any other word cloud solution. We produced the top 5 tweets with counts for each of the hashtag datasets based on tweet data. The word cloud and associated table clearly show that the higher the frequency, the more prominent the term is in the word cloud.

From the word cloud we can make the below 2 observations:

- Katy Perry was referenced more than any other athlete (even more than Russell Wilson or Tom Brady).
- How Budweiser received more mentions than other advertisements during the halftime performance.

Below are the word cloud plots with frequency tables across all hashtags:

1. Word Cloud Plot and Lexical Dispersion plot for #gopatriots:

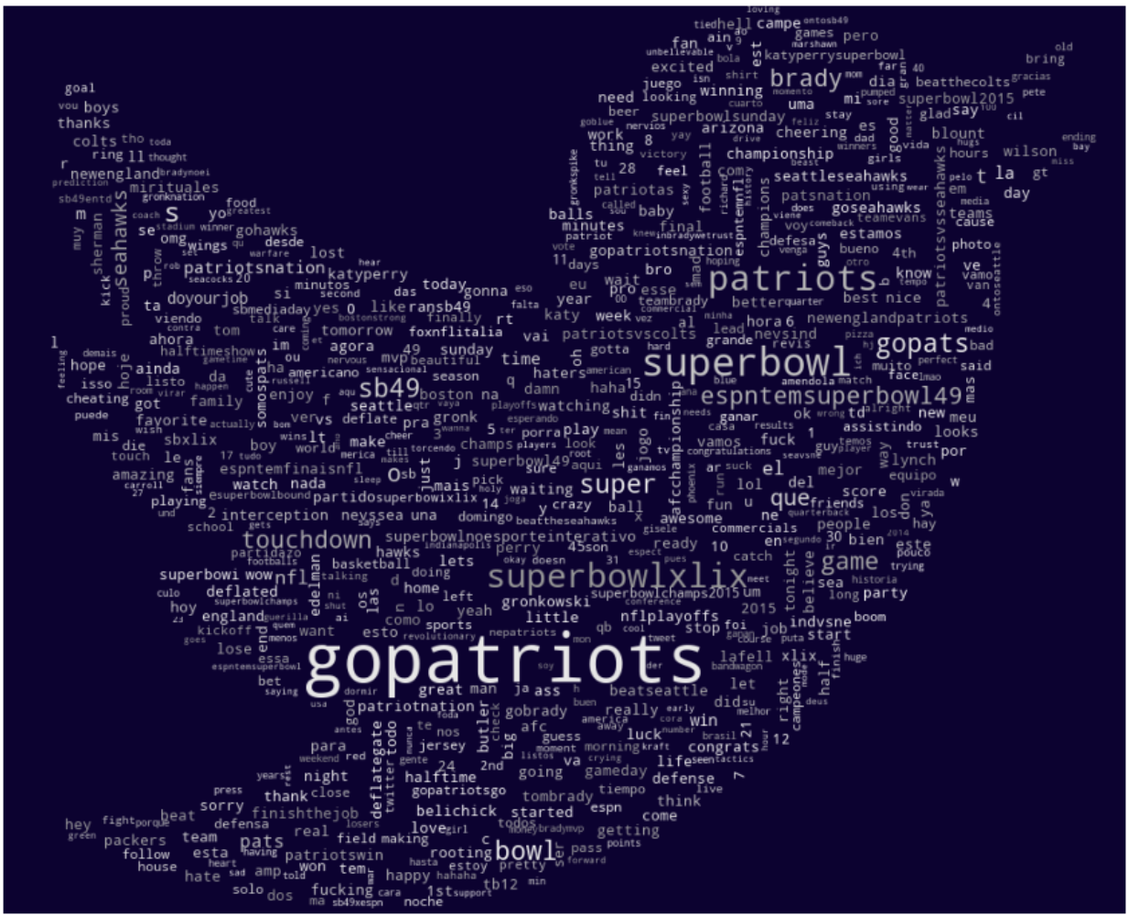


Figure 53: Word cloud for #gopatriots

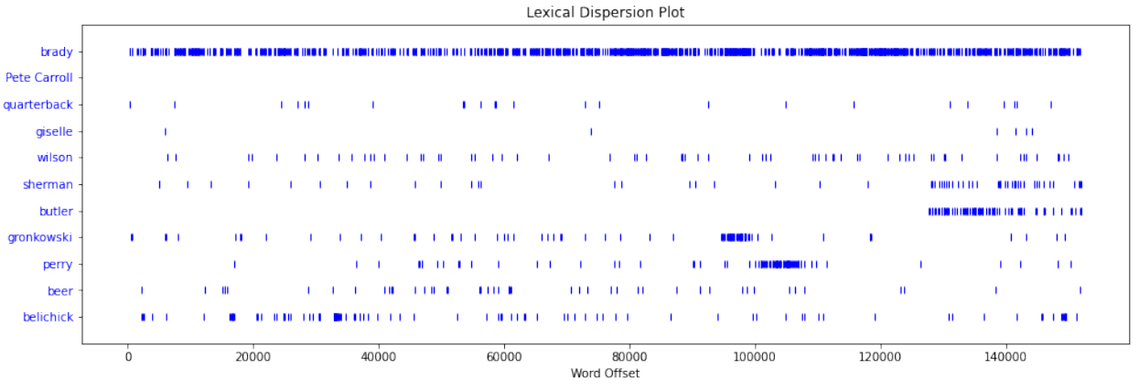


Figure 54: Lexical Dispersion plot for #gopatriots

Tweet Name	Frequency
gopatriots	23581
superbowl	3685
patriots	2769
superbowlxlix	2662
s	1789

Table 6: Top 5 frequency word counts in #gopatriots hashtag dataset

2. Word Cloud Plot and Lexical Dispersion plot for #patriots:

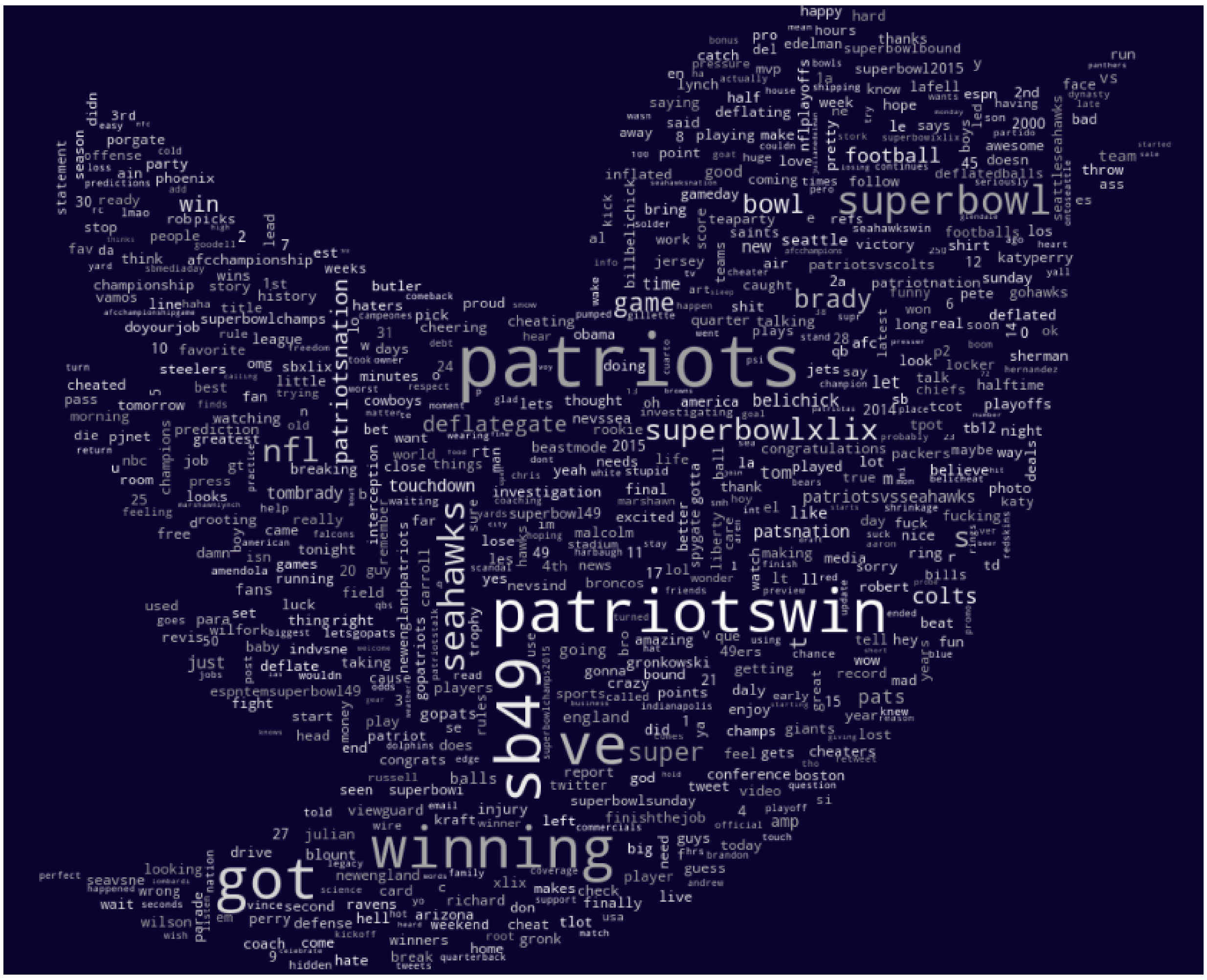


Figure 55: Word cloud for #patriots

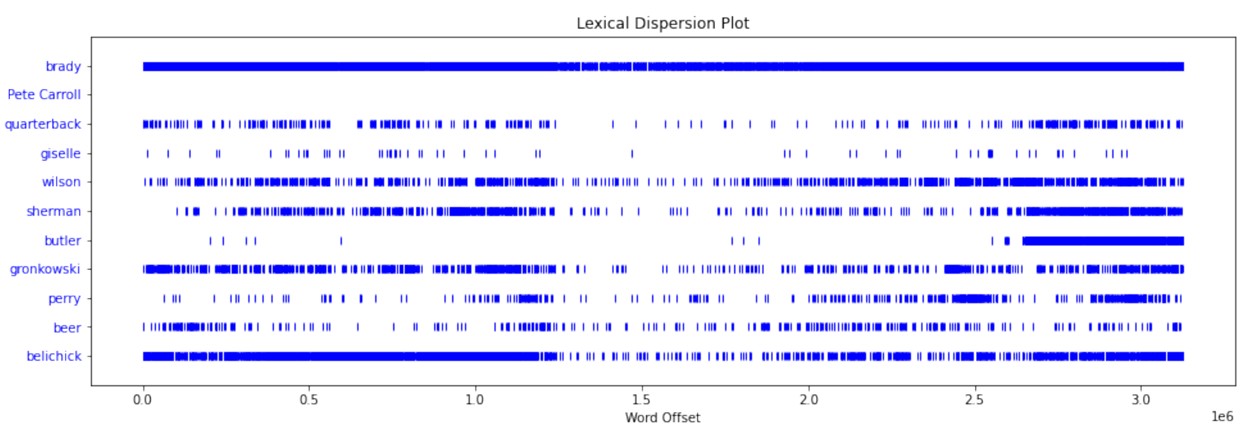


Figure 56: Lexical Dispersion plot for #patriots

Tweet Name	Frequency
patriots	267968
sb49	187849
patriotswin	179573
got	168507
winning	166352

Table 7: Top 5 frequency word counts in #patriots hashtag dataset

3. Word Cloud Plot and Lexical Dispersion plot for #gohawks:

Figure 57: Word cloud for #gohawks

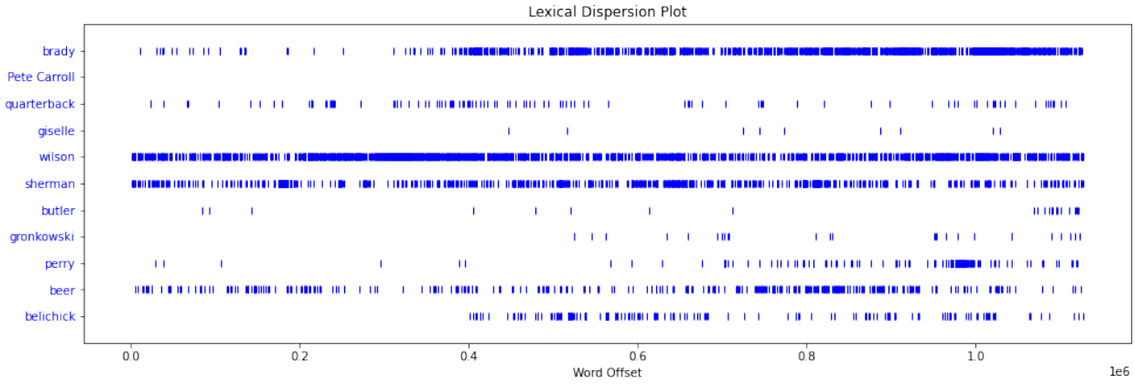


Figure 58: Lexical Dispersion plot for #gohawks

Tweet Name	Frequency
gohawks	171263
seahawks	27006
s	20035
sb49	15935
game	15011

Table 8: Top 5 frequency word counts in #gohawks hashtag dataset

4. Word Cloud Plot and Lexical Dispersion plot for #sb49:

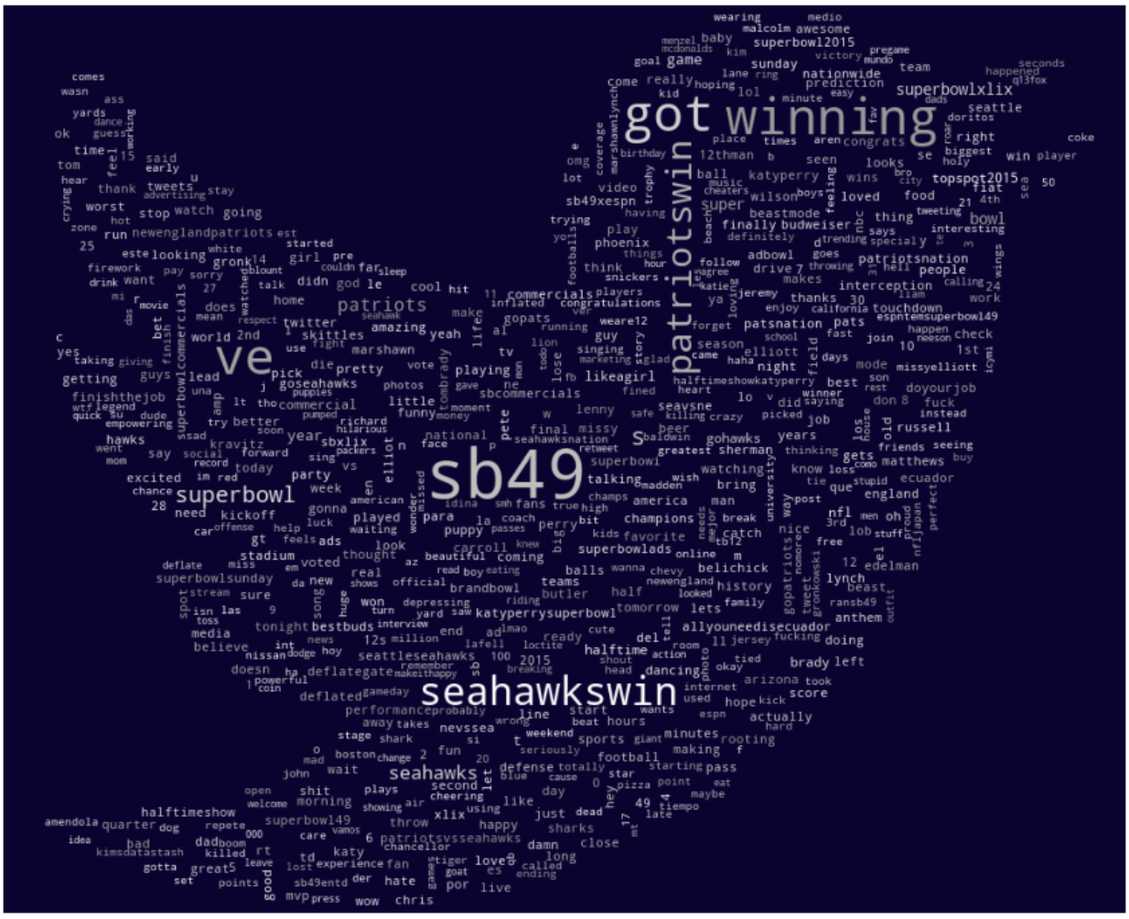


Figure 59: Word cloud for #sb49

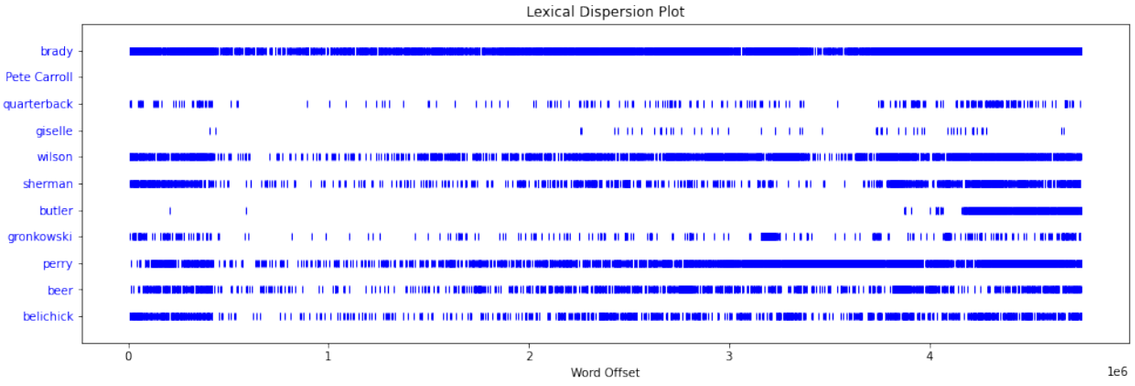


Figure 60: Lexical Dispersion plot for #sb49

Tweet Name	Frequency
sb49	742175
got	367153
ve	364663
winning	363160
seahawkswin	195618

Table 9: Top 5 frequency word counts in #sb49 hashtag dataset

5. Word Cloud Plot and Lexical Dispersion plot for #superbowl:

Figure 61: Word cloud for #superbowl

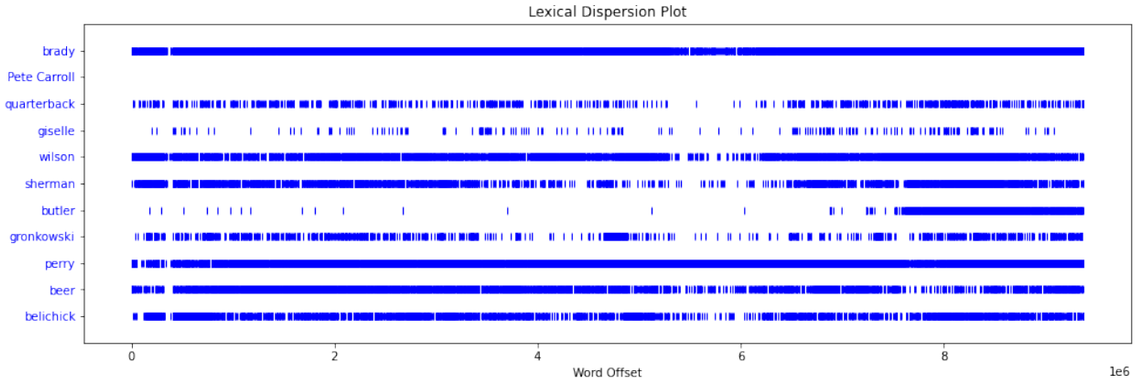


Figure 62: Lexical Dispersion plot for #superbowl

Tweet Name	Frequency
superbowl	749704
superbowlxlix	468806
s	144426
seahawks	121821
patriots	116917

Table 10: Top 5 frequency word counts in #superbowl hashtag dataset

6. Word Cloud Plot and Lexical Dispersion plot for #nfl:

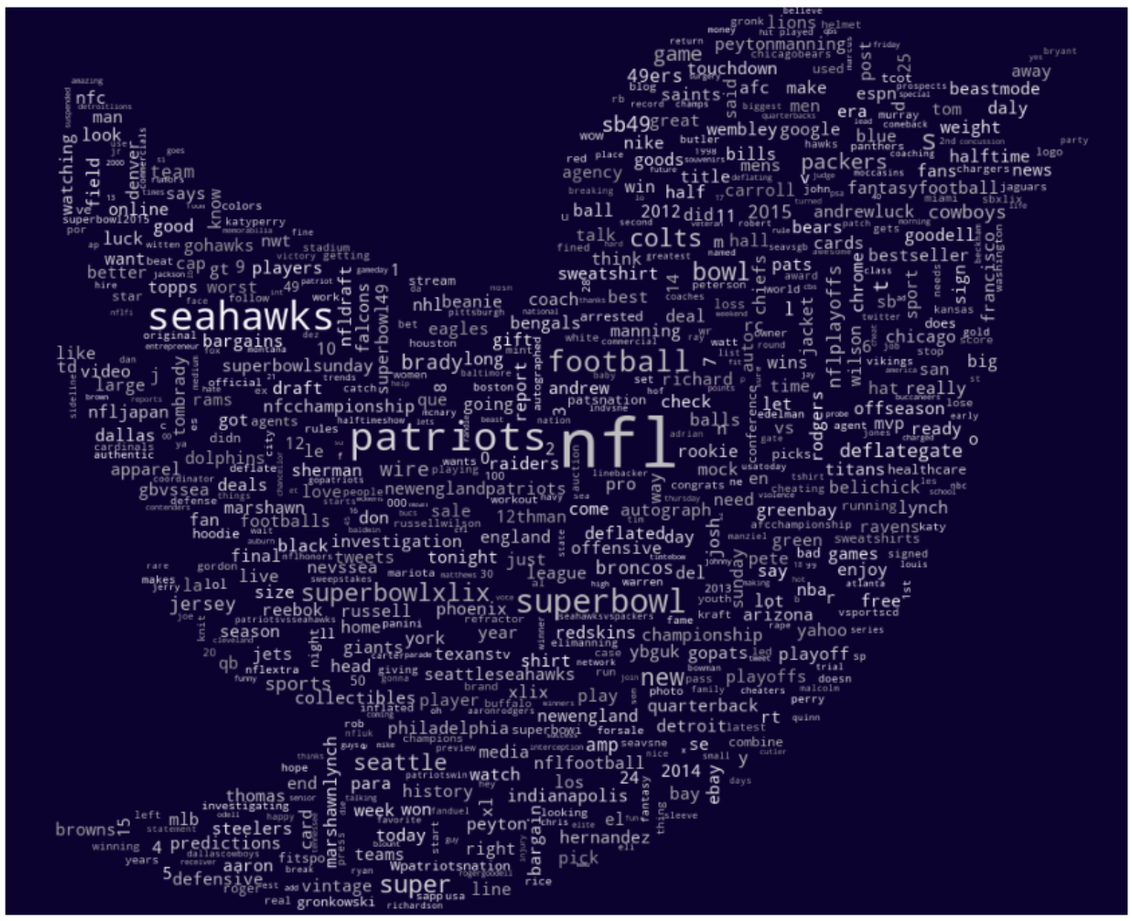


Figure 63: Word cloud for #nfl

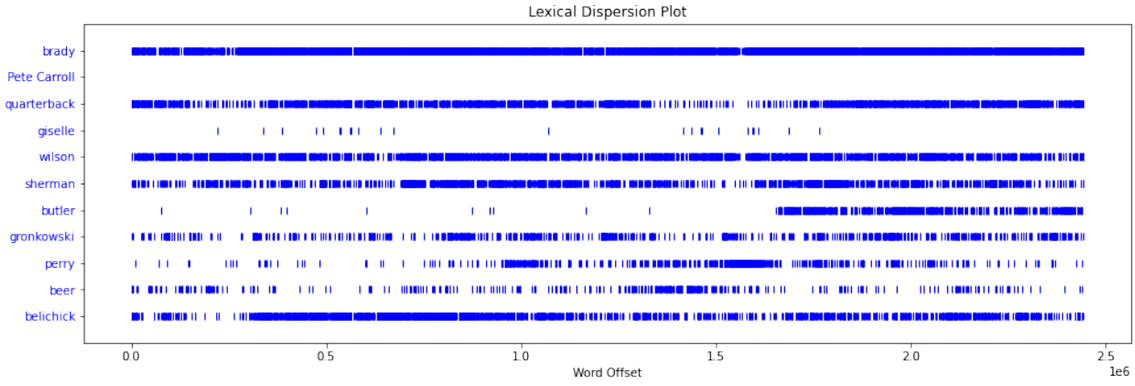


Figure 64: Lexical Dispersion plot for #nfl

Tweet Name	Frequency
nfl	262661
patriots	50814
seahawks	47533
superbowl	32519
s	26734

Table 11: Top 5 frequency word counts in #nfl hashtag dataset

Part 2 - To what extent do highly influential tweets (measured by number of tweets, number of followers, number of retweets and number of impressions) introduce a change in the general Twitter sentiment level?

We tackle the given problem during the time period of 3:00 PM to 9:00 PM on 1st Feb using 5-min time windows. The SentimentIntensityAnalyzer() function from the NLTK package is used to calculate sentiment level. We use Vader sentiment Analyser. VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is tuned in to social media sentiments. VADER employs a mix of A sentiment lexicon is a collection of lexical characteristics (e.g., words) that are categorized as positive or negative based on their semantic orientation. VADER not only shows us the Positivity and Negativity score, but also how positive or negative an emotion is. We employed the 'compound' score. In general, the greater the 'compound' score, the higher the amount of positivity. The Compound score is a metric that calculates the sum of all the lexicon ratings which have been normalized between -1(most extreme negative) and +1 (most extreme positive).
We work on this problem from 3:00 PM to 9:00 PM on February 1st, utilizing 5-minute time slots.

We plotted the number of impressions and the sum of 'compound' ratings to investigate the impact of the number of impressions.

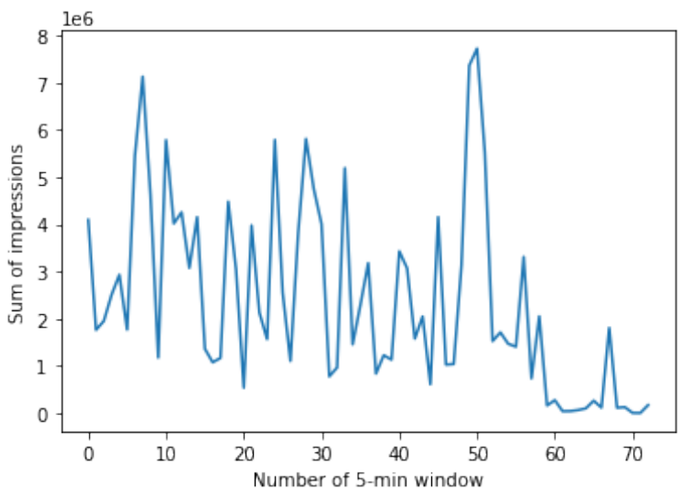


Figure 65: Sum of Impressions for train set = #nfl

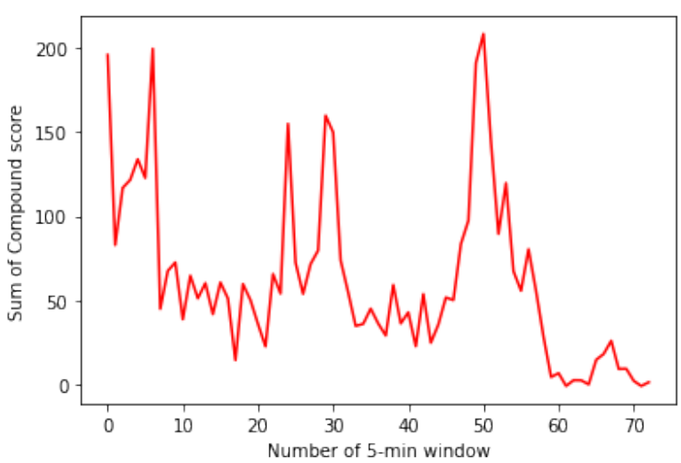


Figure 66: Sum of Compound Score for train set = #nfl

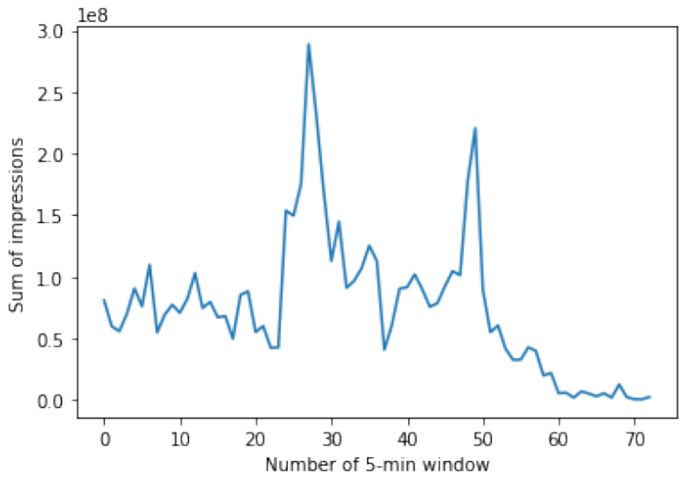


Figure 67: Sum of Impressions for train set = #superbowl

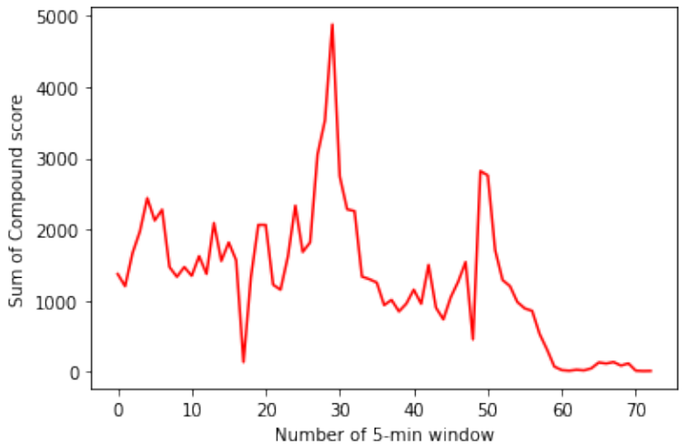


Figure 68: Lexical Dispersion plot for #superbowl

As can be observed, increases in the number of impressions are paralleled by increases in the 'compound' score. This shows that there is a link between the amount of impressions and the total degree of emotion. This is due to the fact that a highly powerful tweet has the potential to alter other people's views and attitudes. As a result, the general emotion level would shift.

Summary of what was done:

- We used a Neural Network Regression Model. The below Hyperparameters were tuned:
  - Model activation: tanh, logistic, ReLU
  - Weight decay penalty term *alpha* (With regularization): $[10^{-2}, 10^{2}]$
  - 'hidden_layer_sizes':

```
[(100,), (200,200,), (300,300,300,), (400,400,400,400,),
(500,500,500,500,500,), (600,600,600,600,600,600,),
(700,700,700,700,700,700,700,), (800,800,800,800,800,800,800,800,),
(900,900,900,900,900,900,900,900,900,)]
```

- Input features = number of tweets, number of followers, number of retweets and number of impressions

- Output to be predicted = sum of the 'compound' scores.
  As the sum of compound score represents the collective, overall sentiment level.
- Method = grid search 10-fold cross-validation to find the best architecture
- Training dataset = #nfl and #Superbowl hashtag datasets were used separately.
- 6.Test dataset = #gopatriots and #gohawks hashtag datasets

Below are the results for data trained on **#nfl** set:

```
Best estimator for Neural Network:
MLPRegressor(alpha=1.0, hiddenlayersizes=(300, 300, 300))
Best estimator for Neural Network:
MLPRegressor(alpha=1.0, hiddenlayersizes=(300, 300, 300))
Best score for Neural Network:  28.158039165260227
Best parameters for Neural Network:  {'activation': 'relu', 'alpha': 1.0,
'hidden_layer_sizes': (300, 300, 300)}
```

Test MSE for **tweets_#gopatriots.txt** using Neural Network: 223.368376

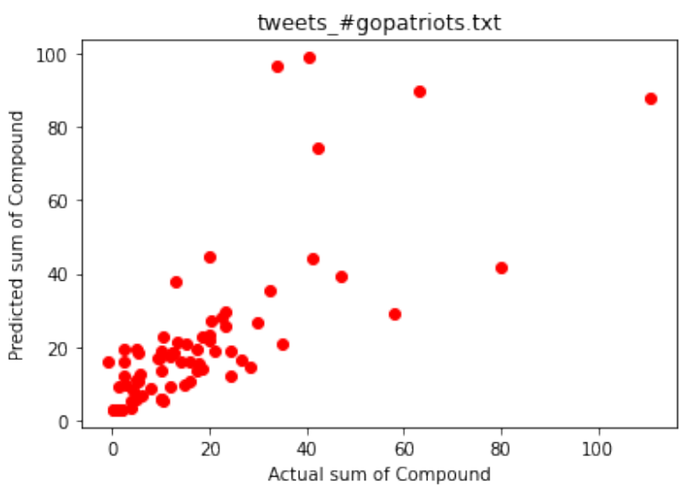


Figure 69: Predicted sum of Compound score Vs Actual sum of Compound score for **tweets_#gopatriots.txt**

Test MSE for **tweets_#gohawks.txt** using Neural Network: 1143.255968

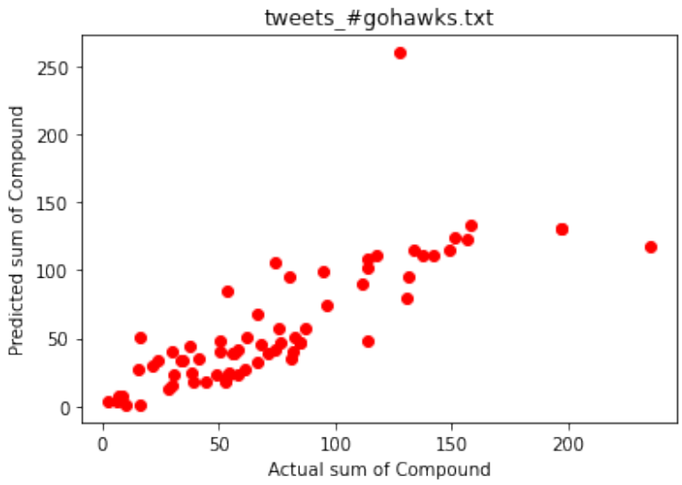


Figure 70: Predicted sum of Compound score Vs Actual sum of Compound score for **tweets_#gohawks.txt**

Below are the results for data trained on **#superbowl** set:

```
Best estimator for Neural Network:
MLPRegressor(alpha=10.0, hiddenlayersizes=(800, 800, 800, 800, 800, 800, 800, 800))
Best estimator for Neural Network:
 MLPRegressor(alpha=10.0,
              hidden_layer_sizes=(800, 800, 800, 800, 800, 800, 800, 800))
\Best score for Neural Network:  442.5833802828627
Best parameters for Neural Network:  {'activation': 'relu',
'alpha': 10.0, 'hidden_layer_sizes': (800, 800, 800, 800, 800, 800, 800, 800)}
```

Test MSE for **tweets_#gopatriots.txt** using Neural Network: 966.287679

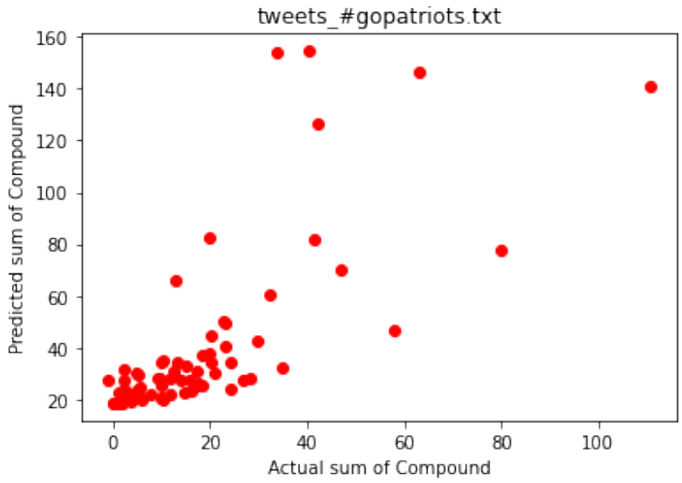


Figure 71: Predicted sum of Compound score Vs Actual sum of Compound score for **tweets_#gopatriots.txt**

Test MSE for textbftweets_#gohawks.txt using Neural Network: 2165.264380

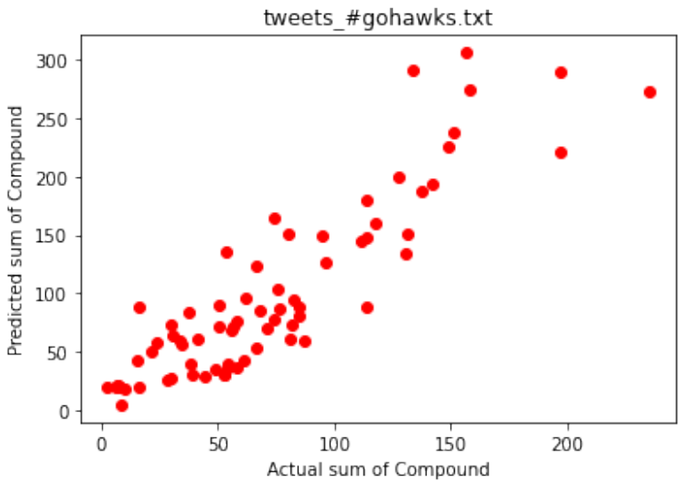


Figure 72: Predicted sum of Compound score Vs Actual sum of Compound score for **tweets_#gohawks.txt**

The charts above show that there is an intermediate-strength linear relationship between the true and forecasted values. As a result, extremely influential tweets have a huge and significant impact on the overall mood level. The quantity of impressions is a crucial aspect that influences the overall emotion level.

Part 3 - Prediction of retweet number:

The first concept is straightforward and is directly influenced by the previous questions. The predictive value is changed from the number of tweets in the next hour to the number of retweets in the next hour. The application here is that the number of retweets may be used as a key indication of web advertising. Consider a corporation that can use an embedded advertising technique in a single tweet and then spread the advertisement to a larger audience through a high number of retweets. To forecast the amount of retweets in the following hour, we used five parameters including the number of tweets, total number of retweets, follower numbers, maximum follower number, and time of day.

We tried gradient boosting method and neural networks and performed grid search to find the best fitting model from cross validation and reported the MSE here.

Gradient Boosting method:

The hyperparameters are given below:

1. max_depth: [10, 20, 50, 100]
2. max_features: ['auto', 'sqrt']
3. min_samples_leaf: [1, 2, 4]
4. min_samples_split': [2, 5, 10]
5. n_estimators: [5,10,50,100,200, 400, 600, 800]
6. kf = KFold(n_splits = 5,random_state = 42, shuffle = True)

Below are the results that we obtained for the optimal value:

min RMSE in testset= 39944.5684231106

parameters:

max_depth= 50

max_features= auto

min_samples_leaf= 2

min_samples_split= 2

n_estimators= 10

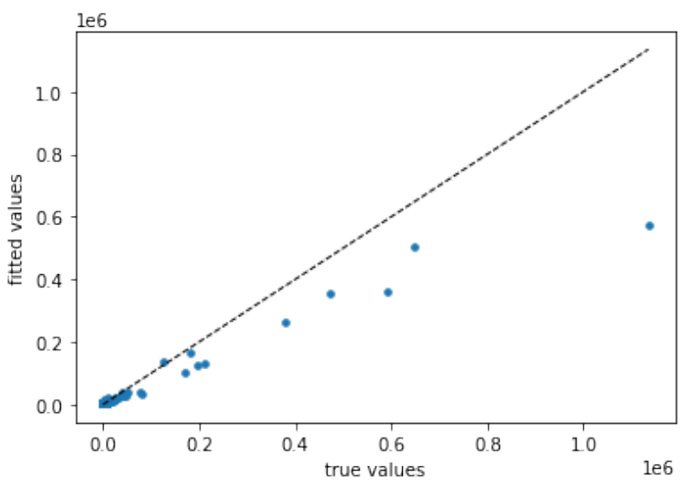


Figure 73: Plot of predicted values vs actual values

Neural Network method:

We varied the number of hidden_units = [50,100,200,300,500,600]

Below are the results that we obtained for the optimal value:
min RMSE in testset= 34658.62675087835
hidden layer sizes= 500 500 500

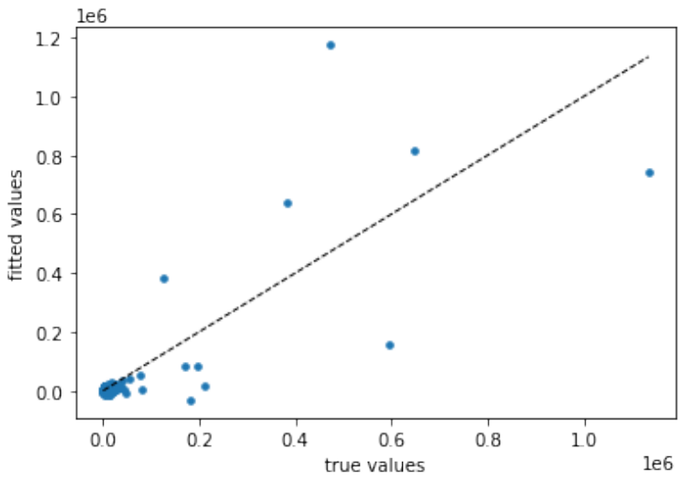


Figure 74: Plot of predicted values vs actual values

The charts above show that there is an intermediate-strength linear relationship between the true and forecasted values. As a result, extremely influential tweets have a huge and significant impact on the overall mood level. The quantity of impressions is a crucial aspect that influences the overall emotion level.

Part 4 - Predict which team does the tweet belong to: We solve the given problem by converting it to binary classification problem and apply a few concepts which we learnt in Project 1.
We used the below specifications provided to us to clean the data.

- Used the Code Snippet given to clean the data.
- Used a code snippet to clean the title. The https and the #, extra spaces were removed.
- Excluded terms that are numbers (e.g. “123”, “-45”, “6.7” etc.). Regex matching and char.isdigit() are used to remove these numbers.
- We Perform lemmatization with nltk.wordnet.WordNetLemmatizer that made use of the POS (part of speech) tag. We converted the given text into sentences and applied lemmatization one sentence at a time. This ensures contextual POS tagging and hence more accurate lemmatization. Punctuations were removed after lemmatization as the text had to be split into sentences. Lemmatization helps to bring down the number of features as all the words with the same root lemma are grouped together. This helps to improve the model training and testing time.
- We used CountVectorizer() to convert the lemmatized text into a bag-of-words matrix. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.
  To omit common words in the English language (called stop-words) that do not provide any contextual information such as “and”, “the”, “I” etc., we set the argument stopwords to “english”.
- We used min df=3. This ensures that words that appear less than 3 times in the training data are dropped. Therefore rare words will be dropped and this helps to reduce the feature set size.
- Then finally the TfidfTransformer() is applied to convert it into Term Frequency Inverse Document Frequency matrix. As the name explains, it helps to find words within the text that helps to differentiate it from the other texts. That is, there could be some words that appear uniformly across all documents and some other ones that appear only in select types of documents. The latter words are more important to us in a classification problem.

Note: We had to reduce the data we were working with as the system crashed with the original amount of data.

- We create empty lists with names: title, hashtags, label. We use the cleaned data and find the title, hashtags and label. The label is decided by whether patriots or hawks is used in the title.
- We used the file #superbowl.txt in all our calculations below.
- The data is divided in train and test, with 5000 data points in the train set and 1000 in the test set.
- We converted patriots and hawks to a 0-1 encoding.

- We performed SVM, L2 Linear Regression on the data.

We obtained an accuracy of 97.3% and an MSE of approximately 0.0220.