# Project 4: Regression Analysis and Define Your Own Task!

ECE 219 - Large Scale Data Mining
University of California, Los Angeles


| | |
|---|---|
| Member 1: | Shruti Mohanty (705494615) |
| Member 2: | Kimaya Kulkarni (805528337) |


## 1 Question 1

Standardize feature columns and prepare them for training.

In this question we are asked to standardise the feature column, before training the data. Before standardising the dataset, certain pre-processing was performed on the datasets.

Pre-processing on diamond dataset -

- The Cut, color and clarity columns were scalar encoded such as - 'Fair': 0, 'Good':1, 'Very Good': 2, 'Premium':3, 'Ideal':4; 'J':0, 'I':1, 'H':2, 'G':3, 'F':4, 'E':5, 'D':6 ; 'I1':0, 'SI2':1, 'SI1':2, 'VS2':3, 'VS1':4, 'VVS2':5, 'VVS1':6, 'IF':7. There existed an ordering relationship between the categorical features because of which scalar encoding made more sense.

Pre-processing on CO gas emission dataset -

- There are 5 CSV files for each year. We concatenated all data points and added a column for the corresponding year and treated that as a categorical feature.
- NOX column was dropped from the dataset and CO was retained as the target variable.
- The years 2011 to 2015 were then scalar encoded from 0 to 4 for ease of processing.

Standardization of datasets is a common requirement for many machine learning estimators; they might behave badly if the individual features do not more-or-less look like standard normally distributed data: Gaussian with zero mean and unit variance. If a fea- ture has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

For standardizing the training sets for both the diamond and CO gas emission dataset, we used StandardScaler() function from SciKit-Learn, using fit and transform together on the training sets to mitigate skewness of the features by transforming the features to have zero mean and unit variance and hence look like Gaussian or normally distributed features. This ensures that all parameters contribute proportionally to the estimation step when minimized on squared error with a high confidence interval. In addition, standardization ensures that the measures of location (moments) points to central tendency, with the resulting distribution having a bounded tail (univariance). Mathematically, standard scaling subtracts the feature column from its mean divided by the standard deviation of the feature.

We have also normalised our target values as per the discussion in class, to ensure a uniform performance across the datasets and models.

## 2 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.

The Pearson correlation coefficient is a measure of linear correlation among two variables, ranging between -1 and 1. -1 indicates a perfect negative correlation (as one feature decreases, the other increases linearly), 0 indicates no correlation and 1 indicates a perfect position correlation (both features increase or decrease linearly with each other).

In this question, we are asked to plot the heatmaps of Pearson correlation matrix of dataset columns for the diamond dataset and the CO gas emission dataset.

The feature columns in diamond dataset -

- carat: weight of the diamond (0.2–5.01);
- cut: quality of the cut (Fair, Good, Very Good, Premium, Ideal);
- color: diamond colour, from 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: length in mm (0–10.74)
- y: width in mm (0–58.9)
- z: depth in mm (0–31.8)
- depth: total depth percentage
- table: width of top of diamond relative to widest point (43–95)

The target column in the diamond dataset -

- price: price in US dollars

The figure below shows the heatmap of Pearson correlation matrix of dataset columns for diamond dataset. To plot the heatmap, we used the heatmap() function in the seaborn library.

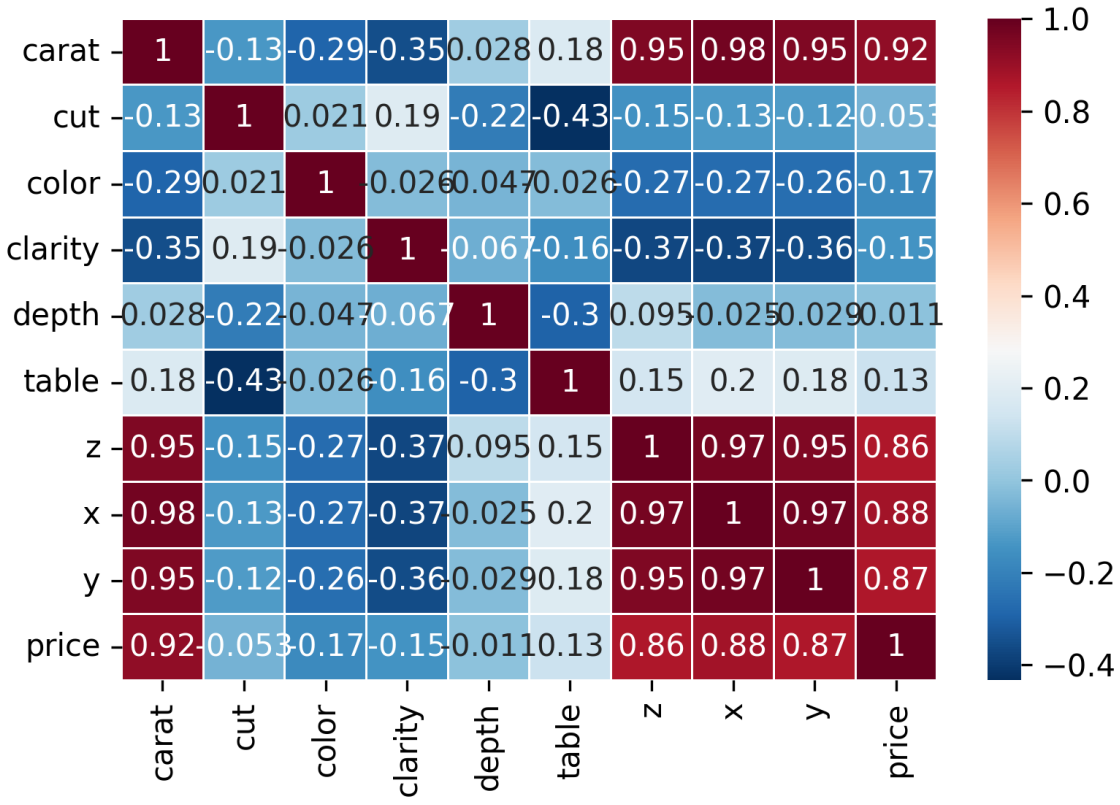


Figure 1:
Pearson Correlation Matrix Heatmap for Diamond dataset

For the target variable Price, the features with the highest absolute correlation values are - carat, x, y, z. This makes sense as the weight or carat of the diamond should affect the price of the diamond. Also the bigger the diamond, its priced more, so the dimensions length , width and depth should also play a role in the price of the diamonds. This makes the most intuitive sense as well with high correlations in the range 0.86 to 0.92.

The feature columns in the CO dataset are -

- 9 sensor measurements aggregated over one hour (by means of average or sum) from a gas turbine - AT', 'AP', 'AH', 'AFDP', 'GTEP', 'TIT', 'TAT', 'TEY', 'CDP'.
- year - The year of the measurement

The target columns in the CO dataset is -

- CO emission

The figure below shows the heatmap of Pearson correlation matrix of dataset columns for the CO emission dataset. To plot the heatmap, we used the heatmap() function in the seaborn library.

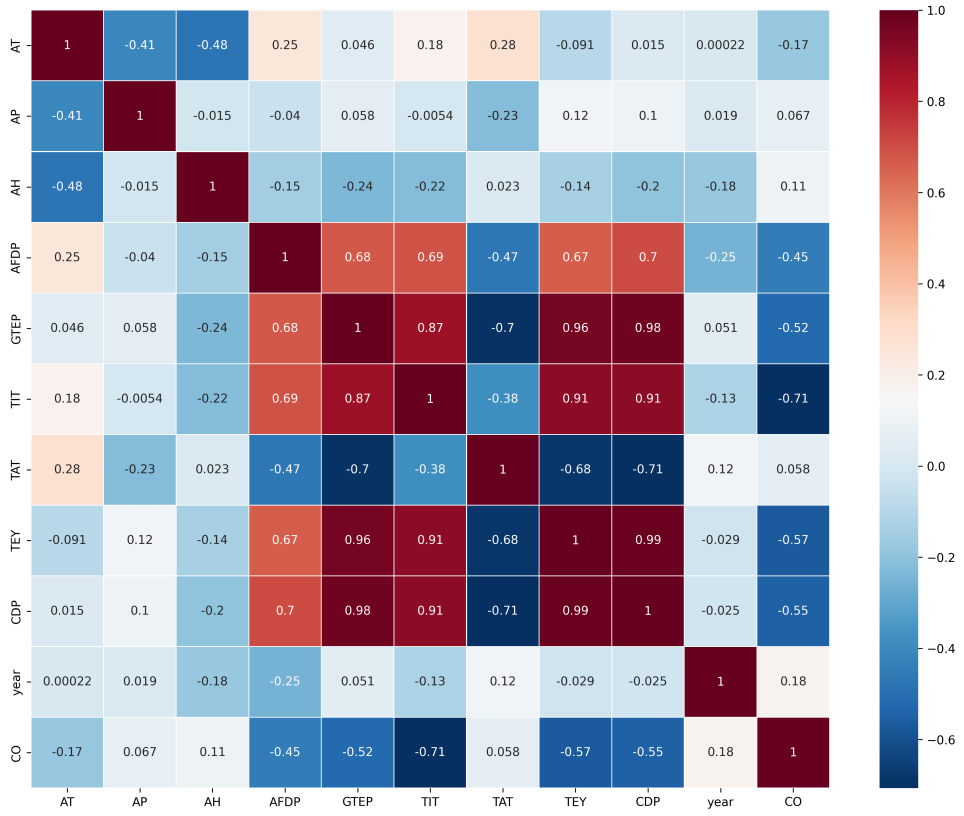


Figure 2:
Pearson Correlation Matrix Heatmap for CO emission dataset

For the target variable CO, the features with the highest absolute correlation values are - TIT, TEY, CDP, GTEP, and AFDP with the values being in the range 0.45 to 0.71. There isn't much intuitive explanation of why these measurements have high correlation with the target CO emission. But they are sensor measurements so we assume they affect CO emission the most compared to other sensory measurements.

# 3 Question 3

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

In this question, we are asked to plot the histogram of numerical features for both the datasets. We use plt.hist() function for the histogram plots. The valid numerical features for each dataset include:

- Diamonds - 'carat','depth', 'table','price', 'x','y','z'. Price is a numerical but target value. Cut, color and clarity are not numerical features, they are categorical features.
- CO Gas emission - 'AT', 'AP', 'AH', 'AFDP', 'GTEP', 'TIT', 'TAT', 'TEY', 'CDP' are the numerical features. CO is numerical target variable. Year is a categorical feature in this dataset.

Figure 3 shows the histogram of numerical features for the diamond dataset.

From Figure 3, we see that 'depth', 'table' are normally distributed .'Carat', 'price','x', and 'z' is positively skewed.

Figure 4 shows the histogram of numerical features for the CO emissions dataset.

From Figure 4, we see that 'AH', 'TAT', 'TIT' are negatively skewed. 'AP', 'AT', 'CDP', 'GTEP', 'TEY' have no skewness and have different modalities of normal distribution. 'AFDP' is slightly positively skewed.

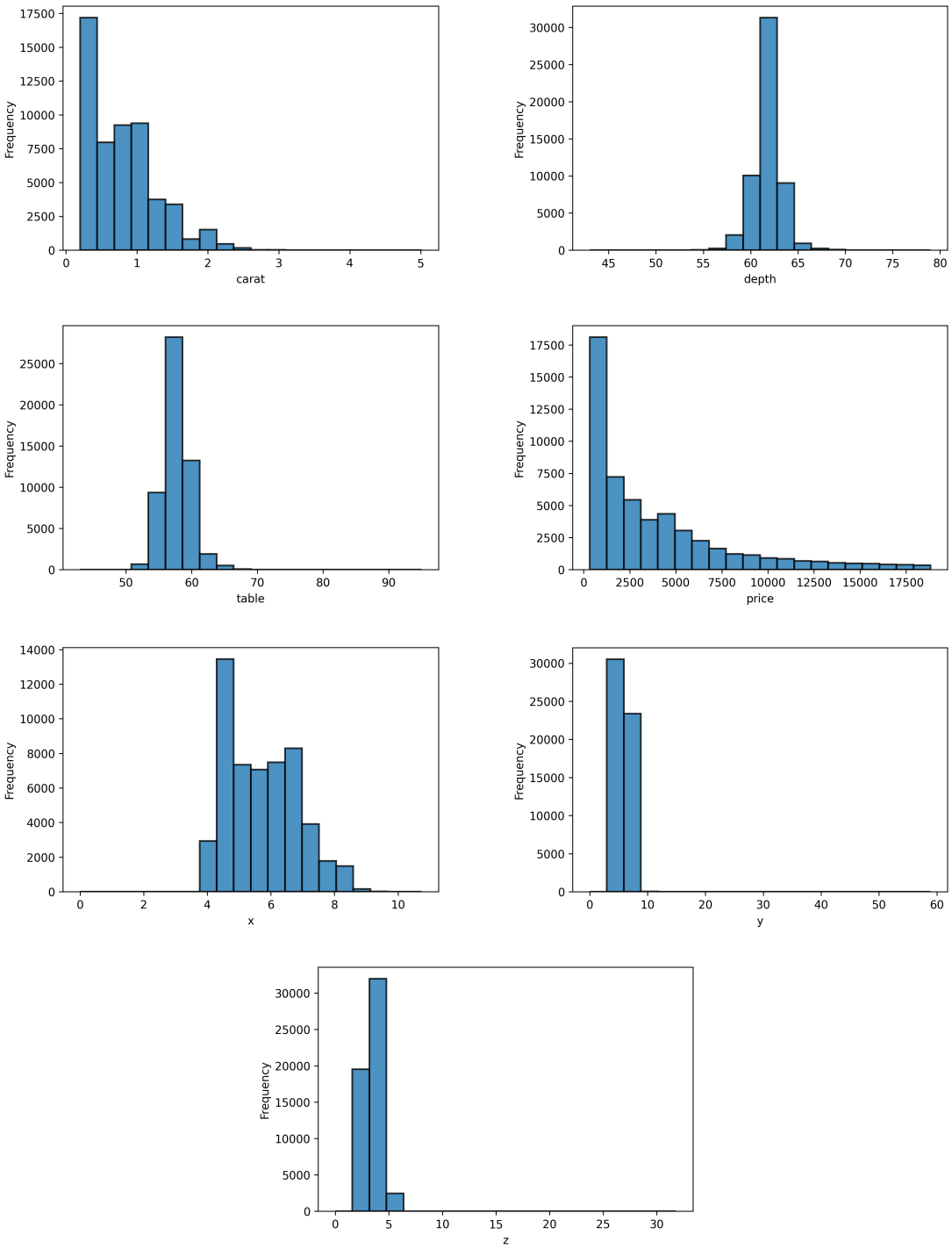


Figure 3: Histogram of 'carat','depth', 'table','price', 'x','y','z' in diamonds dataset

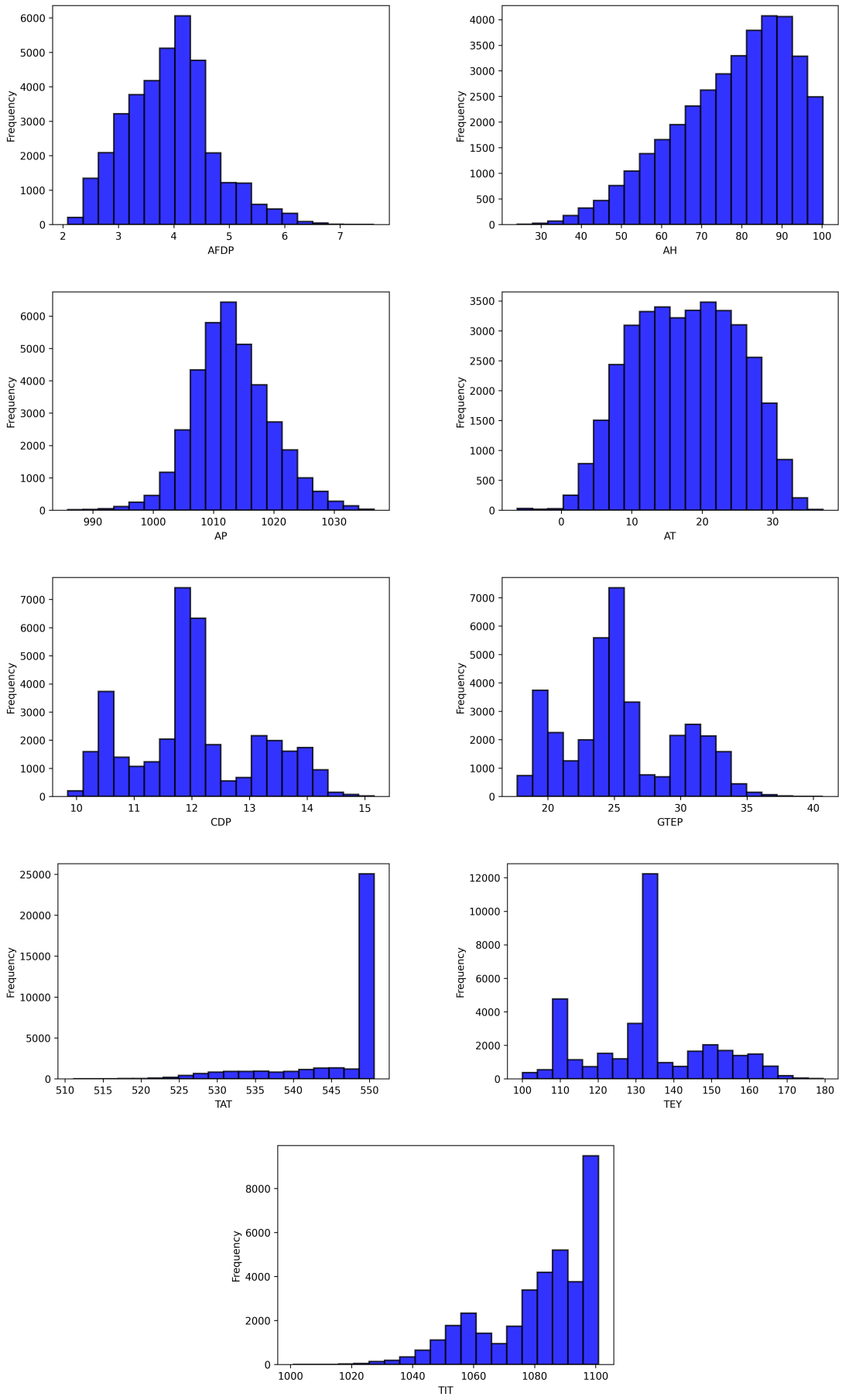


Figure 4: Histogram of 'AT', 'AP', 'AH', 'AFDP', 'GTEP', 'TIT', 'TAT', 'TEY', 'CDP' in CO emissions dataset

Features with high skewness are not desirable in regression modelling because:

- Common parametric statistical tools, including regression analysis, require the distribution of the dependent variable to be Gaussian conditioned on the features and error component in the linear regression model.
- Skewed features provide misleading measures of location (moments), i.e., the mean cannot be used to represent the distribution of the features. Least-squares regression requires the mean to point to central tendency.
- Parameter estimation on skewed distributions leads to disproportionate influence on the parameter estimates when minimized on squared error.

If a certain feature has high skewness, one can perform the following pre-processing techniques:

- Transformations: Positive/right skewed distributions: Square root (weak), cube root (moderate), logarithmic (strong but cannot be applied to 0 values) or reciprocal (strong but cannot be applied to 0 values) transforms. Negative/left skewed distributions: Power transforms (e.g., squares, cubes, box-cox etc.).
- Removing outliers (values with extreme distances from the mean of the distribution).
- Transformations coupled with normalization (e.g., z-score or min-max) and scaling or standardization.

# 4 Question 4

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

In this question, we are asked to provide box-plots of categorical features for both the datasets against respective target variables. The valid categorical features and target variables for each dataset include:

- Diamonds - Cut, color and clarity are the categorical features, and 'price' is the target variable.
- CO Gas emission - Year is a categorical feature in this dataset, and 'CO' is target variable.

Figure 5 shows the box plots of the categorical features for target variable price from the diamond dataset.

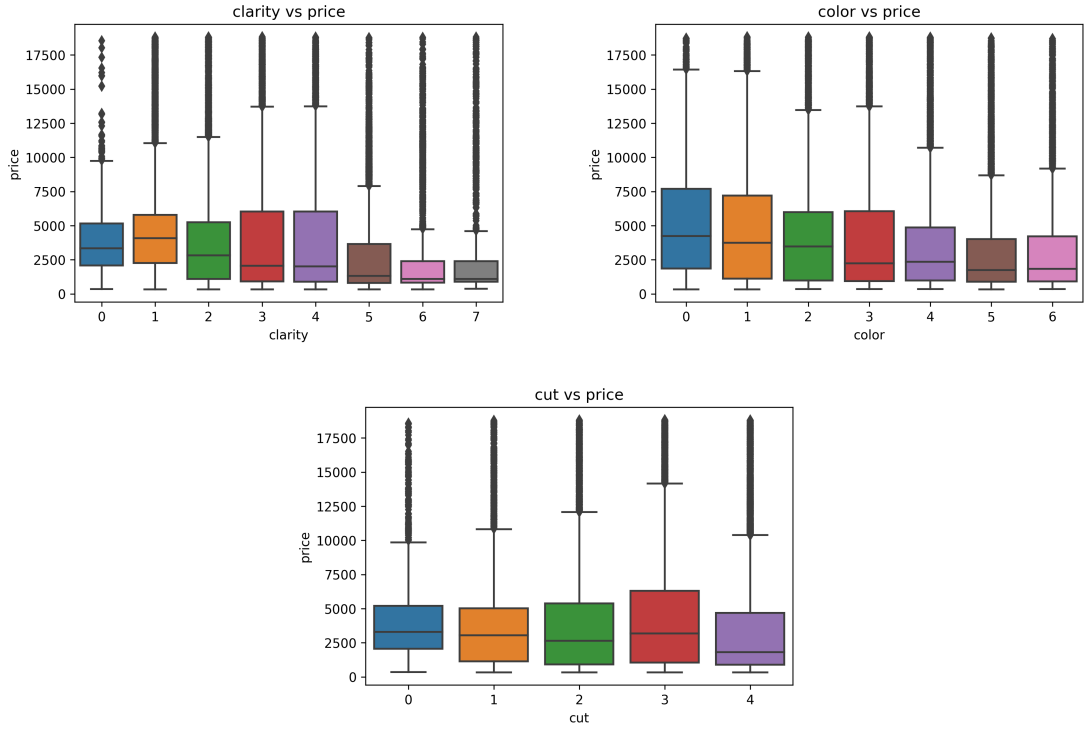


Figure 5: Histogram of 'clarity','color', 'cut' v/s price in diamonds dataset

The inferences from this box plot are -

- There are a large number of outliers present for this distribution of clarity v/s price specially as the clarity goes on increasing. The price range different is the maximum for clarity 3 and 4 compared to the other clarities. Clarity 6 and 7 have the maximum number of outliers present.
- For 'cut' the box plot trend is roughly the same across the different cuts. The price range for 50 percentile of them is in the same range. There are few outliers present for this distribution as well.
- For color v/s price, the box plot shows a slightly decreasing pattern as the color category increases from 0 to 6. The number of outliers also increases as the color goes upwards.

Figure 6 shows the box plots of the categorical feature for target variable 'CO' from the emissions dataset.

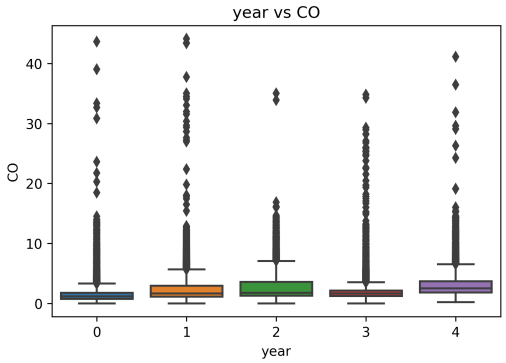


Figure 6: Histogram of 'year' v/s 'CO' in emissions dataset

The inference from this box plot is -

- All the box plots are agreeing with each other in terms of the trend of the CO emission. They are all close to each other for all 5 years present in this plot. However, year 1 and 3 has the maximum number of outliers present for CO emission, and year 2 has the least outliers.

# 5 Question 5

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

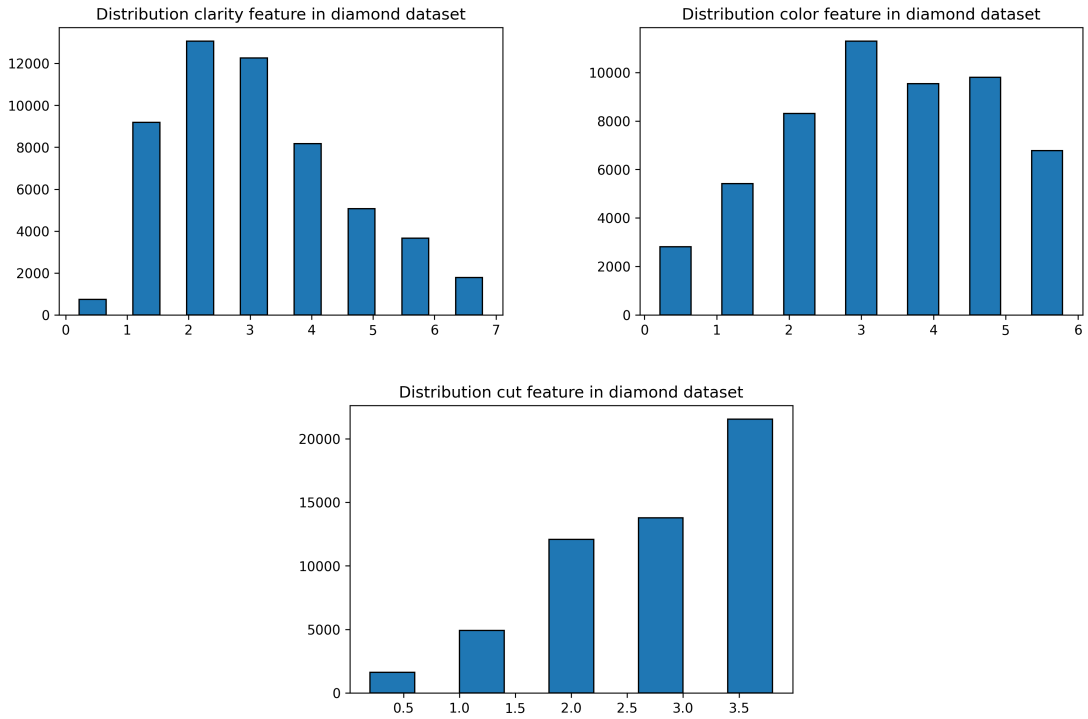


Figure 7: Counts of 'clarity','color', 'cut' in diamonds dataset

The inference from these plots are -

- The distribution of clarity feature in the diamond dataset is almost similar to a normal distribution, with the max data present for clarity 2 and 3 in the dataset.
- The distribution of the color feature in the dataset is also following a normal trend, where color 3 has maximum data in this dataset.
- The distribution of cut feature in the dataset follows an increasing trend, where cut 4 has the maximum data records present.

As seen from above the dataset is not uniformly distributed.

# 6 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.

The yearly trends for each feature are shown in the figure below

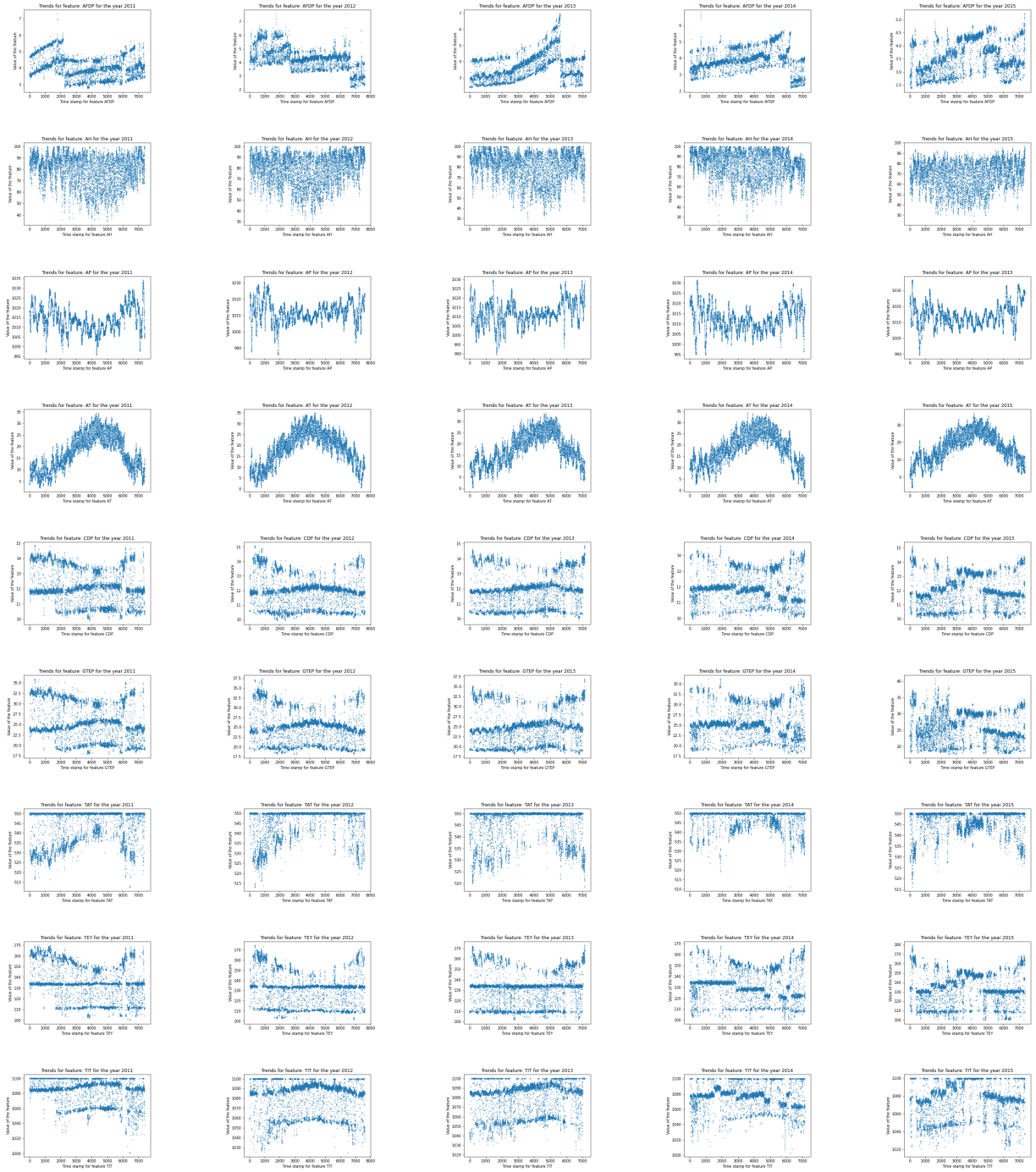

Figure 8: Yearly trends for 'AT', 'AP', 'AH', 'AFDP', 'GTEP', 'TIT', 'TAT', 'TEY', 'CDP' in CO emissions dataset from 2011 - 2015

Our inference from these yearly trend plots are -

- The values of the features are not constant throughout the year and keep changing with time stamp. There is slight variations seen in the trends amongst years as well.

- For AFDP, in year 2013 there is a sharp increase and then decrease seen. The other years, similar oscillations are noticed in the trend.
- All the other features display similar oscillations across the years. That implies feature distribution across year doesn't change. But every feature has a different distribution as the feature changes are different from one another.

# 7 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.

In this question, we are asked to explore how feature selection affects model performance. For this question, we test "mutual information regression" (MI) and "F-regression) (F-Score) feature selection schemes for linear regression without regularization, lasso regression, and ridge regression using default value of hyperparameters for on both datasets with standardization applied to the entire dataset. We use SelectKBest() function to select the k best features for k ranging from 1 to maximum number of features in each dataset for each of the two feature selection schemes, and obtain average negative test root-mean-squared error (RMSE) (the more positive the "negative test RMSE" is, the better) from 10-fold cross validation. The target variable is "price" for diamond dataset, 'CO' for emissions dataset. Figure 9,10 shows the plots for the effect of feature selection on both datasets.

MI (or information gain) is defined as a non-parametric non-negative measure of dependency between two features, which is equal to 0 if two features are independent and positive depending on how dependent the features are. MI is also known as the expected value of the pointwise mutual information. The intuition is that features that are correlated with the target variables should contain large amounts of Shannon information about the dependent variable.

F-Score performs univariate linear regression tests, i.e., F-tests to evaluate the significance level of the improvement in performance of the model with respect to the addition of new variables. It tells us if a complex model with all the features of a simpler model is significantly better than the latter with respect to the p-value

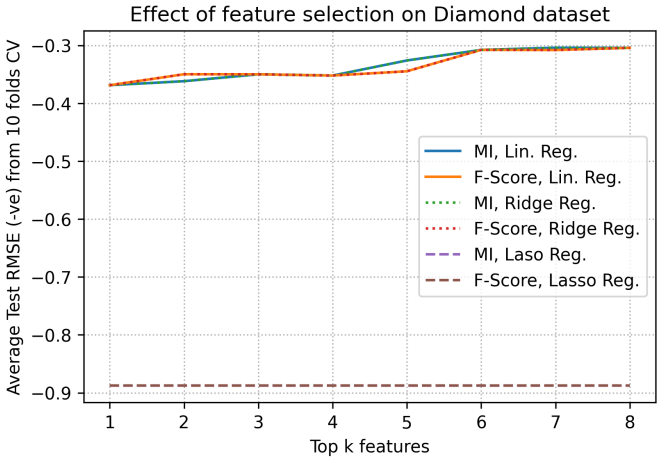


Figure 9:
Effect of feature selection on Diamond Dataset

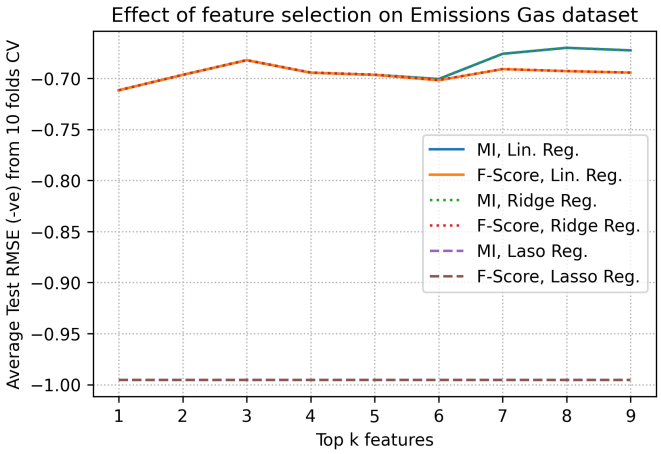


Figure 10:
Effect of feature selection on CO emissions Dataset

Our inference from these plots are -

- From Figure 9 we can see that the RMSE almost saturates as we go on increasing the number of features we are considering for both the mutual information, and f-score. So, we consider all the 9 features in the dataset for the questions and regression model analysis after this.
- From Figure 10, we also notice that the RMSE saturates with respect to f-score for the k values after 4. Mutual information also varies very slightly after k features increase beyond 4. So, we decide to select the top 4 important features for the CO emission dataset.
- By selecting all 9 features for diamond dataset, and top 4 features for the CO emission dataset, we improve our processing time and feature selection method while not increasing the test error value. The RMSE with these features is -0.3 for diamond dataset, and -0.7 for CO emission dataset.
- For all three models (linear, lasso and ridge regression), as the number of features increases, the test RMSE improves and converges to a constant value when a certain number of features are selected. This is because all of the information-conveying features that have the maximum correlation with the target variable have already been selected. Beyond the feature values we have selected, there's a possibility of overfitting and we should avoid selecting more features. Feature selection helps reduce complexity and improve interpretability and generalizability of chosen models.

# 8 Question 8

Explain how each regularization scheme affects the learned hypotheses

In questions 8 to 11, we are asked to compare the performance of linear, lasso and ridge regression models on the datasets, along with the effect of feature scaling and meaning of p-value.

The objective functions for each of these regression models are -

- Ordinary least squares:

$$J(W) = \frac{1}{2N}\sum_{i=1}^{M}(y_i - \sum_{j=0}^{P} w_j x_{ij})^2$$

- Lasso regression:

$$J(W) = \frac{1}{2N}\sum_{i=1}^{M}(y_i - \sum_{j=0}^{P} w_j x_{ij})^2 + \lambda \sum_{j=0}^{P} |w_j|$$

- Ridge regression:

$$J(W) = \frac{1}{2N}\sum_{i=1}^{M}(y_i - \sum_{j=0}^{P} w_j x_{ij})^2 + \lambda \sum_{j=0}^{P} w_j^2$$

y is the target variable, x represents the features and w represents the weights assigned to each feature, and $\lambda$ represents regularization strength.

This was our analysis on both the datasets - Diamond and CO emission, and how the regularisation schemes affect the learned hypotheses -

- Lasso regression is also known as linear regression with L1 regularization, while ridge regression is also known as linear regression with L2 or Tikhonov regularization. Regularization encourages generalization by yielding simpler models to prevent overfitting. Regularization attempts to improve generalizability by increasing estimator bias while reducing variance, causing many of the weights to be very small, leading to a simpler model which performs well on unseen test data but may have lower overall RMSE.

- For both kinds of regularization, as regularization strength $\lambda$ increases, more weights in the learned model are set to 0. This is due to the fact that the L1 and L2 regularization are formulated as minimizing the weights. Thus, to prevent the regularization portion of the least squares equation from becoming too large, the optimization program will attempt to minimize the weights, ideally the weights should approach 0. However, as models become more and more simpler, they start losing the most important weights required promoting the salient features to contribute to the final estimation. As a result, very large values of regularization strength will cause the test accuracy to drop dramatically (underfitting). However, finite values of regularization strength can help make the model more generalizable on unseen test data, preventing overfitting. Regularization also makes the model more stable by reducing variance and sensitivity to outliers and increasing bias.

- Lasso regularization is suitable for feature selection and construction of simple and sparse linear regression models, where features associated with 0 weights can be discarded. This is because L1 regularization encourages all kinds of weights to shrink to 0, regardless of size of w, as the subgradient of $\|w\|$ is sign(w). L1 regularization is more likely to zero out coefficients than L2 regularization for similar test accuracies because it assumes priors on the weights sampled from an isotropic Laplace distribution (linear descent), which has a much lower Q factor than Gaussian distribution (exponential descent). Thus, only a sub-selection of features are active in the learned hypothesis.

- Ridge regularization, which assumes priors on the weights sampled from a unit Gaussian distribution, is suitable for reducing the effects of collinear features, which can lead to increased variance (hence instability and sensitivity to outliers) of the model. This is because the subgradient of $\|w\|^2$ depends on not just the sign of w but also the magnitude of w, which can be thought of as scaling the variance of weights, reducing dependence of the model on few features and encouraging distributed contribution of all the features. This leads to regression models with diffuse weights compared to sparse weights from L1 regularization. Thus, ridge regularization promotions participation of all the features in the learned hypothesis to prevent overfitting.

# 9 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 (average negative test RMSE, higher is better) of linear regression, lasso regression and ridge regression on all the features in the diamond dataset, and top 4 selected features in the CO emissions dataset. For the gas dataset we test the features selected via both MI and F-Score. 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()), with $\lambda$in the range of [-3,3]. It is also possible to use LassoCV() and GridCV() to calculate the optimal penalty terms. We also test the effects of feature scaling , and standardization on these datasets and they are explained in Question 10.

The table below reports the best penalty parameters obtained for each possible choice of configurations and models for both the datasets, along with average test and train RMSE for best penalty parameters obtained via 10-fold grid search cross-validation.

- For the diamond dataset the best test RMSE obtained is -0.301 on standardised data using ridge regression, with model penalty set to 0.001 with all 9 features used for regression analysis.
- For the CO emissions dataset, the best test RMSE obtained is -0.685 on standardised data using lasso regression, with model penalty set to 0.01 for the top 4 selected features by F1 score.

Table 1: Linear, Ridge, and Lasso regression results on Diamond dataset

| Model | Standardisation | Features selected | Penalty Parameter | Train RMSE | Test RMSE |
|---|---|---|---|---|---|
| Linear | No | 9 (All) | N/A | -0.302 | -0.302 |
| Linear | Yes | 9 (All) | N/A | -0.302 | -0.302 |
| Ridge | No | 9 (All) | 0.001 | -0.300 | -0.303 |
| Ridge | Yes | 9 (All) | 0.001 | -0.300 | -0.301 |
| Lasso | No | 9 (All) | 0.001 | -0.302 | -0.304 |
| Lasso | Yes | 9 (All) | 0.001 | -0.302 | -0.303 |

Table 2: Linear, Ridge, and Lasso regression results on CO emissions dataset

| Model | Standardisation | Features selected | Penalty Parameter | Train RMSE | Test RMSE |
|---|---|---|---|---|---|
| Linear | No | 4 (F1Score) | N/A | -0.66 | -0.69 |
| Linear | Yes | 4 (F1Score) | N/A | -0.66 | -0.687 |
| Ridge | No | 4 (F1Score) | 100 | -0.66 | -0.69 |
| Ridge | Yes | 4 (F1Score) | 100 | -0.66 | -0.687 |
| Lasso | No | 4 (F1Score) | 0.01 | -0.66 | -0.69 |
| Lasso | Yes | 4 (F1Score) | 0.01 | -0.65 | -0.685 |

## 10 Question 10

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

From Table 1 and 2 above , we can see that feature scaling does make some difference in the RMSE when regularization is used, as our best RMSE is obtained with

Feature scaling does not cause any changes in the original dataset distribution. Without regularization, the change in values caused by feature scaling will be reflected by corresponding changes in the weights of the model to achieve the lowest RMSE. Since the data distribution is the same as before, the changes caused by feature scaling will reflect linearly on the changes in the coefficients, yielding no performance gains or losses. As a result, we see no change in test RMSE with or without feature scaling when regularization is absent.

As discussed in Question 3 , standardisation of data will help deal with skewness in the distribution. Consider two salient features, one which has a much smaller range and absolute value than the other one. Without normalization, the coefficients of these two features will be unbalanced, with the smaller feature having a larger coefficient than the larger feature in order to have a balanced outcome on the target variable. Since regularization aims at minimizing the weights of the regression model, it would remove or penalize the coefficients of the smaller feature, while having negligible effect on the weights of the larger feature. Feature scaling ensures that the weights assigned to each feature does not get adversely affected by a biased regularizer penalizing smaller feature when regularization is used and leads to more stable and well-conditioned models.

## 11 Question 11

Some linear regression packages return p-values for different features. What is the meaning of them and how can you infer the most significant features?

The p-value for each feature tests the null hypothesis that the feature has no correlation with the target variable (probability of the weights of the feature being 0) at the population level. A feature with a high p-value (greater than 0.05) indicates that there is insufficient evidence to find any meaningful correlation of that feature with the changes in the target-variable. On the other hand, a feature with a low p-value (lesser than 0.05) indicates that the probability of the coefficients of that particular feature being 0 is low, which indicates 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). Thus, the most significant features will have a small p-value.

We use the statsmodels.regression.linear model.OLS() to the CO emissions dataset to find the p-values.

We have tested this example on the CO gas emission dataset and some of them correspond to our heatmap analysis in the second question. From our p-values analysis the significant features in ascending order we obtain are AT, TIT, TAT, TEY, year, AP, CDP, GTEP, AH, AFDP. The values that partially match what we obtained in Question 2 are TIT, TEY. The slight difference is probably due to the influence of one-hot-encoded categorical variables or due to standardisation or feature scaling of the dataset.

## 12 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 both the datasets in Question 13. To find the most salient features, we find the coefficients with the highest absolute values in the polynomial model. The most salient (top 5) features for the best-performing polynomial regressors for the datasets are:

- Diamond : ['carat', 'clarity', 'x', 'color', 'z']
- CO gas emission: ['TIT', '$CDP^2$', 'GTEP', 'TEY', 'CDP']

For the Diamond Dataset the most salient features are in relation to what we found in Question 1. Carat, x, and z had high correlation values with the target variable price. As it was explained earlier the dimensions of the diamond, and it's weight plays a big role in deciding the price of the diamond. In addition to those, the salient features clarity and color found from polynomial regression is also intuitively right as good color diamonds with more clarity usually cost more. So, this supports our findings from all our previous questions.

For the CO emissions dataset the most salient feature includes a mathematical combination of the feature column - CDP. That means it weighs more in the way it affects the target variable CO emissions. Our findings are in accordance with the heatmap analysis as TIT, CDP, GTEP and TEY had very high correlation values as well with target variable. So, our polynomial model is correct and it supports the analysis from the other questions, even though we don't have much intuitive sense for this dataset features.

## 13 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?

For this question, we test the performance in terms of train and test RMSE of polynomial regression of various degrees on the diamond and CO emission dataset. Since F-Score selected features performed better than MI, we selected top 4 features for CO emissions dataset, and all 9 features were considered for the diamond dataset as their F-scores were all comparable and so was the test RMSE.

For the model, we used ridge regression to incorporate appropriate regularization term (L2) to prevent overfitting, with the penalty parameter being optimized using 10-fold grid search cross-validation using GridSearchCV(). The hyperparameter space for the grid search also included the degree of polynomial features, which ranged from 1 to 6 for both the datasets. To generate polynomial features, we used PolynomialFeatures() function from SciKit-Learn within the pipeline. The figures below shows the average test and train RMSE versus the degree of polynomial for diamond and CO emission datasets.

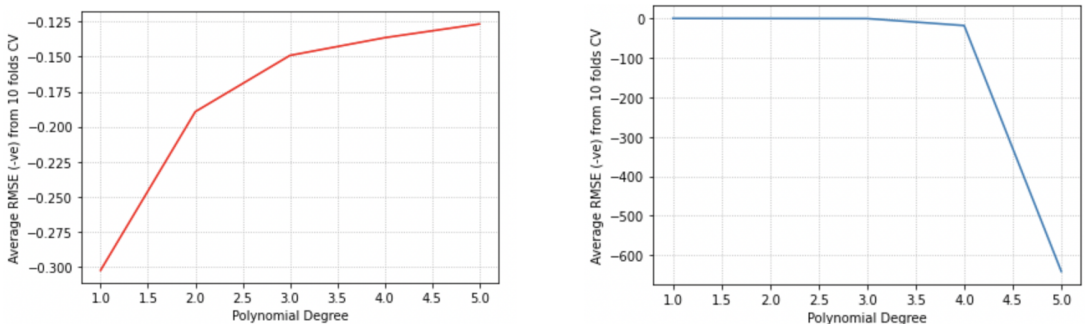


Figure 11: Train and Test RMSE v/s degree of polynomial for diamonds dataset

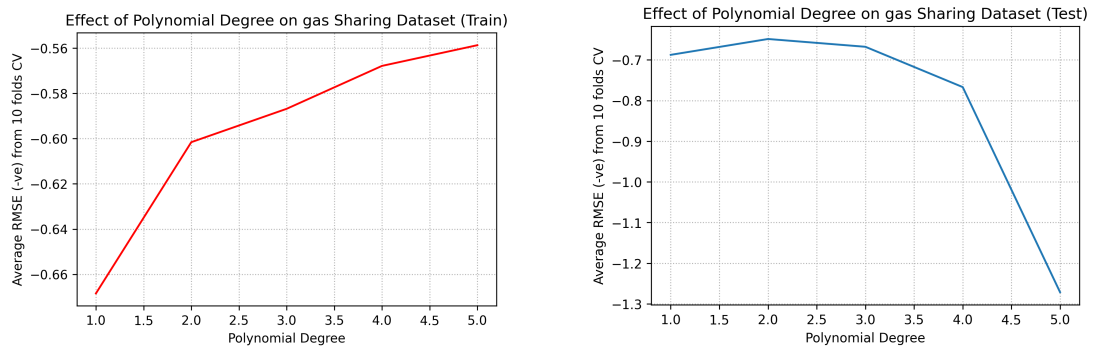


Figure 12: Train and Test RMSE v/s degree of polynomial for CO emissions dataset

From Figures 11 and 12, we see that for the test RMSE, as polynomial degree increases, the RMSE improves until a particular threshold, beyond which the RMSE either stays constant or starts to become worse. Higher polynomial degrees give rise to larger models with more model capacity, however, beyond a critical point, the complexity of the model tend to cause overfitting (which is clear from the training graphs in Figure 11 and 12, which converges to a constant value, which the test RMSE curves start to drop). The best degree of polynomial for both the datasets based on the RMSE values from a 10 fold grid search is -

- Diamond dataset - 4 ( Train RMSE -0.13, Test RMSE -0.15)
- CO gas emission dataset - 3 ( Train RMSE -0.59 , Test RMSE -0.61)

Having a very high order degree polynomial is a bad idea mainly because although higher-degree polynomial models have a larger model capacity, higher polynomial degrees lead to a greater possible combination of features from the raw features, which can lead to overfitted, complex and uninterpretable models. In fact, if we look at the test RMSE in Figures above, we observe that the test RMSE starts to become worse if the polynomial degree is increased beyond a certain threshold. As a result, we should only increase the polynomial degree upto a particular threshold and choose the degree beyond which the test RMSE does not change significantly or starts dropping. The search space should be small to ensure one does not run out of memory during grid search. Also, the memory expands for very high order degree polynomials.

## 14 Question 14

Product of particular features might yield a new feature that is highly correlated with the target variable, which would otherwise be impossible to craft using regression. For the diamond dataset, an example of a features whose product should in theory, perfectly correlate with the price is the diamond area which is xy. In other words, product of dimensions will be highly correlated with the price of the diamond. Thus, the equation for our new feature is:

$new\_feature = xy$

We retrained the Ridge Regression obtained in previous question for the diamond dataset with the newly appended feature to the dataset. The obtained metrics are as follows:

Average Test RMSE (-ve) without new feature (degree = 2): -0.32610

Average Test RMSE (-ve) with new feature (degree = 2): -0.302061

We see that the addition of the new feature has caused significant improvements in the test RMSE and proves our hypothesis that the area of diamond is correlated with the price.

## 15 Question 15

The results found in this question relate to the best MLP found via 10-fold grid-search cross-validation for each dataset as well as the best linear regressors from the previous questions. The table below shows the performance comparison of best-performing linear regression model and MLP model on Diamond and CO emission dataset.

Table 3: Performance comparison on best test rmse values

| Dataset | Linear Regression | Ridge | Lasso | MLP |
|---|---|---|---|---|
| Diamond | -0.302 | -0.301 | -0.303 | -0.142 |
| CO Emission | -0.687 | -0.687 | -0.685 | -0.616 |

It can be seen that overall, neural networks perform significantly better than linear regression both with and without regularization. There are several reasons for this:

- Neural networks are much more capable of capturing non-linear relationships with the help of non-linear activations of weights compared to linear regression models.

- Neural networks have more information capacity (trainable parameters) than linear regression models, allowing MLP to capture complex mathematical relationships and mappings via hidden layer connections between features and target variables that is otherwise not possible to infer via linear dependencies with a small number of trainable weights.

- Neural networks automatically extract salient non-linear features from the input dataset without the need for explicit transformations, feature engineering or feature selection.

# 16 Question 16

For this question, we test the performance in terms of train and test RMSE (average negative test RMSE, higher is better) of MLP for various weight decays, model activation functions and network size (number of hidden neurons and depth) on Diamond and CO Emission datasets. Since F-Score selected features performed better than MI, we selected top 10 features from F-Score. The hyperparameter space was as follows:

Model activation: [logistic, tanh, relu]

Weight decay (for regularization):$[10^{-3}, 10^{2}]$

Network sizes: Combinations from 10, 20, 30, 50 as the number of hidden neurons, with minimum network depth of 1 and maximum depth of 4

$[(10,), (20,), (30,), (50,), (10,10), (10,20), (10,30), (10,50), (20,20), (20,30), (20,50), (30,30), (30,50), (50,50),$
$(10,10,10), (10,10,20), (10,10,30), (10,10,50), (10,20,20), (10,20,30), (10,20,50), (10,30,30), (10,30,50),$
$(10,50,50), (20,20,20), (20,20,30), (20,20,50), (20,30,30), (20,30,50), (20,50,50), (30,30,30), (30,30,50),$
$(30,50,50), (50,50,50), (10,10,10,10), (10,10,10,20), (10,10,10,30), (10,10,10,50), (10,10,20,20), (10,10,20,30),$
$(10,10,20,50), (10,10,30,30), (10,10,30,50), (10,10,50,50), (10,20,20,20), (10,20,20,30), (10,20,20,50),$
$(10,20,30,30), (10,20,30,50), (10,20,50,50), (10,30,30,30), (10,30,30,50), (10,30,50,50), (10,50,50,50),$
$(20,20,20,20), (20,20,20,30), (20,20,20,50), (20,20,30,30), (20,20,30,50), (20,20,50,50), (20,30,30,30),$
$(20,30,30,50), (20,30,50,50), (20,50,50,50), (30,30,30,30), (30,30,30,50), (30,30,50,50), (30,50,50,50),$
$(50,50,50,50)]$

We used 10-fold grid search cross-validation through the Sci-Kit Learn's pipeline() to find the most optimal hyperparameters for each dataset systematically in the parameter design space. Note that it is possible to use a much wider design space for network sizes, but to accelerate the search process, we limited the network sizes to choose combinations from four integers. The following table shows the optimal hyperparameter set, as well as the best average negative train and test RMSE obtained for each dataset.

Table 4: Performance summary of fully connected neural network with most optimal hyperparameters on Diamond and CO Emission Datasets

| Dataset | Best Network Arch | Best Weight Decay | Best Activation Function | Train RMSE | Test RMSE |
|---|---|---|---|---|---|
| Diamond | (50, 50) | 0.05 | tanh | -0.1261 | -0.1420 |
| CO Emission | (20, 30, 30, 50) | 1.0 | relu | -0.4703 | -0.6166 |

## 17 Question 17

In our project, the neural network is solving a regression problem with continuous outputs that can range from negative infinity to positive infinity, the output activation function should either be none or linear (identity), both of which can provide outputs in the range . ReLU activation is bounded to only 0 and positive values, tanh activation is bounded between -1 and 1 and sigmoid/logistic activation ranges from 0 to 1. However, regression outputs can take any value beyond such finite limits in the entire domain of real numbers, which is achievable via linear activation or no activation at all in the output layer.

## 18 Question 18

There are several reasons as to why we cannot (or should not) increase the depth of the neural network in an unconstrained manner:

- Vanishing gradients: If a network is too deep, then the gradients will become exponentially small as the information back-propagates from the final layers towards the initial layers. This will lead to slow or premature convergence to a sub-optimal point during the training phase, with the weights in the initial layers being extremely small compared to the weights in the final layers.
- Overfitting: A deeper neural network has more trainable parameters than a shallow one, leading to a complex model with enough capacity to memorize the entire training set, consequently overfitting on the training data and generalizing poorly on the test set.
- Interpretability: Large networks with millions of parameters can lead to complex relationships between features and target variables beyond human understanding, leading to black-box models.
- Compute constraints: A complex network with many layers takes more time, memory and compute power than a simpler model for training, neural architecture search and real-time deployment.
- Data requirements: If the network has millions of trainable weights, the neural network requires a large training set to avoid overfitting and provide robust and usable outputs during deployment.

# 19 Question 19

For this question, we test the performance in terms of train and test RMSE (average negative test RMSE, higher is better) of random forests for various number of trees, maximum number of features and depth of each tree on diamond and CO emission datasets. Since F-Score selected features performed better than MI, we selected top 10 features from F-Score. The hyperparameter space was as follows:

Maximum number of features: 1 to 10
Number of trees/estimators: 10 to 200
Depth of each tree: 1 to 20

We used 10-fold grid search cross-validation through the Sci-Kit Learn's pipeline() to find the most optimal hyperparameters for each dataset systematically in the parameter design space. We also obtain the "Out-of-Bag Error" (OOB) for the best random forest models on each dataset.
To study the effects of each hyperparameter independently on the overall performance, for each of the three hyperparameters, we keep two of the hyperparameters constant while plotting the effects of the target hyperparameter on average train and test RMSE via 10-fold grid search cross-validation.

The following table shows the optimal hyperparameter set, as well as the best average negative train and test RMSE obtained for each dataset via random forest models.

Table 5: Performance summary of random forest with most optimal hyperparameters on Diamond and CO Emission Datasets

| Dataset | Max number of features | Depth of each tree | number of trees | Train RMSE | Test RMSE |
|---|---|---|---|---|---|
| Diamond | 6 | 10 | 200 | -0.1195 | -0.1948 |
| CO Emission | 2 | 6 | 200 | -0.2611 | -0.6232 |

The following figures show the effects of the number of trees on the overall model performance for Diamond and CO emission datasets respectively

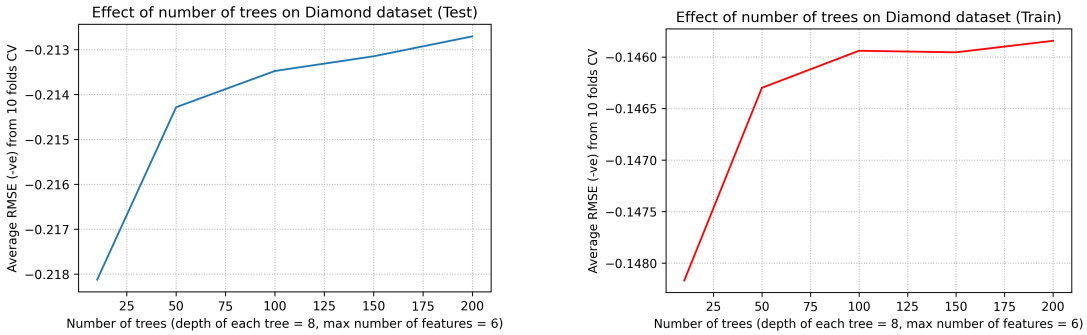


Figure 13: Effect of number of trees on diamond dataset (train and test)

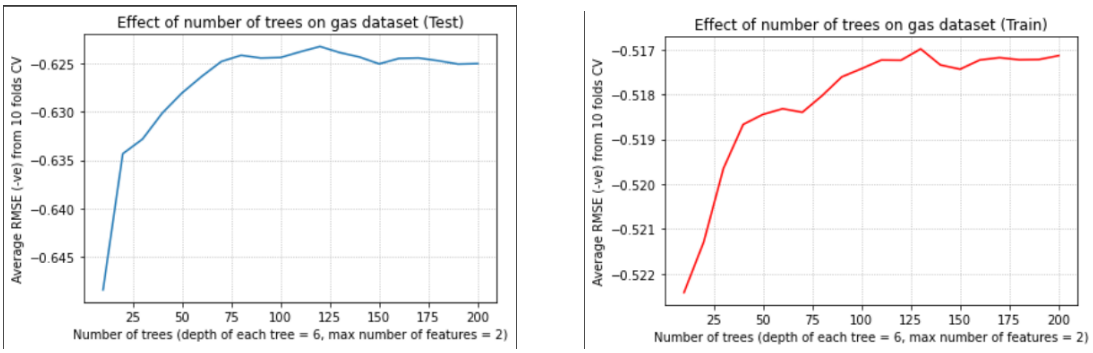


Figure 14: Effect of number of trees on CO emission dataset (train and test)

From Figures above, overall, 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. 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 (e.g. 64 to 128) within compute constraints, as more number of trees does not cause overfitting.

- In a random forest, each tree along with its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN) as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Thus, according to WLLN, the target variable and tree-decision has finite variance, leading to the overall decision (and any other statistic) of the random forest to converge towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality).
- The expected error rate for a random forest ensemble is a non-monotonous function of the number of trees. In fact, error metrics such as RMSE are noisy once a sufficiently large number of required estimators have been used. The convergence rate of the error rate curve does not depend on the number of trees and is only dependent on the distribution of the expected value of the decisions of the trees.
- While increasing the number of trees does not cause overfitting according to WLLN, other hyperparameters might result in unnecessary variance in the model and cause over-correlation within the ensemble, breaking the notion of independent trees within the random forest.

The following figures show the effects of the depth of each tree on the overall model performance for diamond and CO emission datasets respectively.

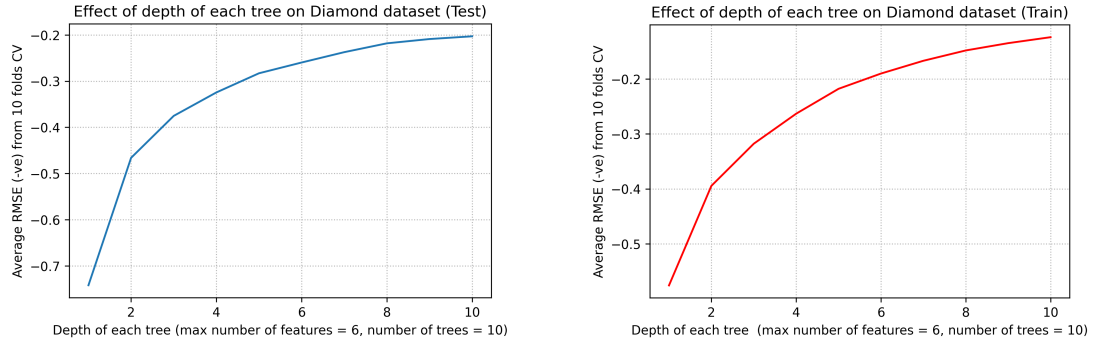


Figure 15: Effect of depth of each tree on diamond dataset (train and test)

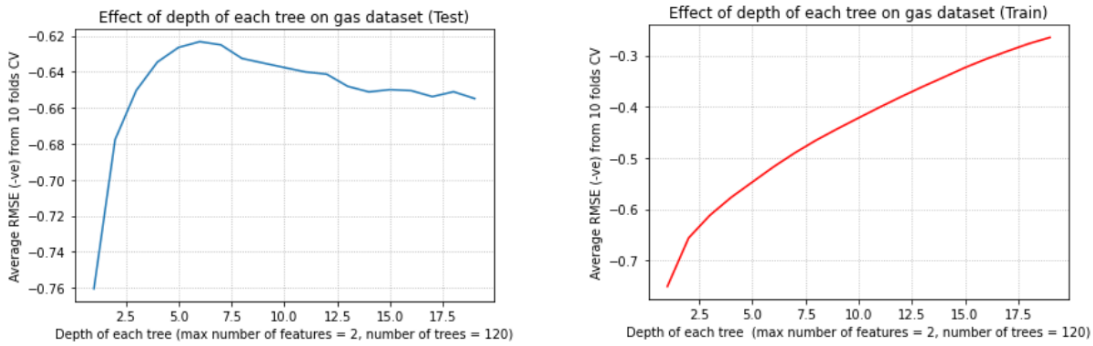


Figure 16: Effect of depth of each tree on CO emission dataset (train and test)

From the above figures, for both datasets, increasing the depth of each tree dramatically improves the error rate for both the training and test sets, converging to a constant value. For the test set, the error rate might start to degrade if the depth of each tree exceeds a certain threshold. 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, 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. In other words, increasing tree depth decreases bias but increases variance.
The figures below show the effects of the maximum number of features on the overall model performance for diamond and co emission datasets respectively.

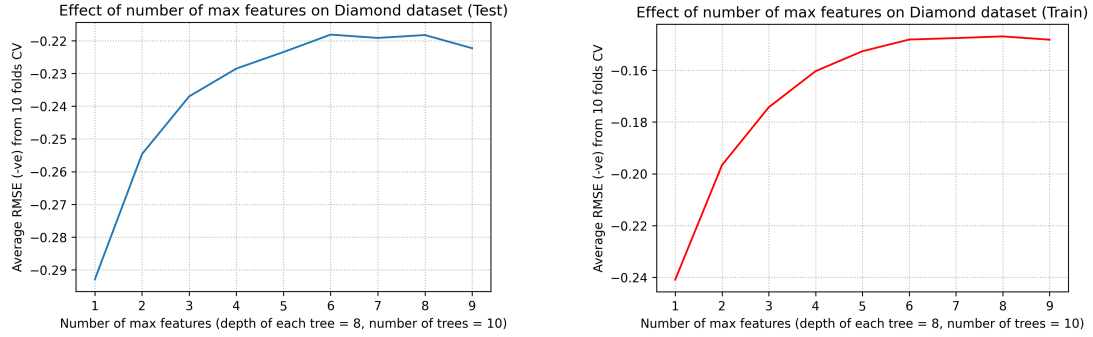


Figure 17: Effect of maximum number of features on diamond dataset (train and test)

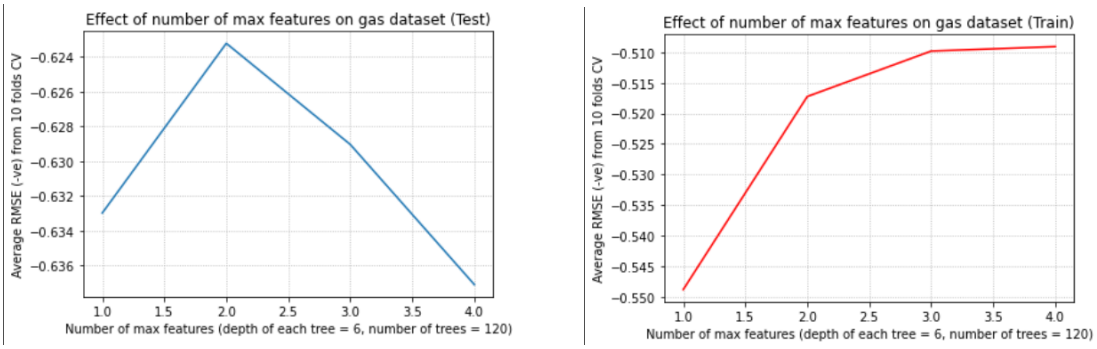


Figure 18: Effect of maximum number of features on CO- emission dataset (train and test)

From the above figures we can observe that increasing the maximum number of features improves

the training RMSE monotonically, converging to a mean value, while the test RMSE starts to degrade after showing improvement until a critical point. This indicates that the number of features also acts as a regularizer like depth of tree, with very large values leading to overfitting on the training set and poor generalization on the test set. This is because increasing the number of features for each tree improves the model capacity of individual trees, but also increases the correlation between each tree, breaking the notion of independent trees within the random forest. Selecting a fraction of the features aims to decorrelate the individual trees and improve generalization error rates, increasing the strength of the random forest.

## 20 Question 20

In a random forest, each tree along with its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN) as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Thus, according to WLLN, the target variable and tree-decision has finite variance, leading to the overall decision (and any other statistic) of the random forest to converge towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). In other words, a random forest contains a large number of uncorrelated models, with the final target variable being a majority decision fusion from all of the trees. Bagging from independent models outperform individual models because of the following reasons:

- Each decision tree is trained on a different and independent random subspace, forcing variation, decorrelation and diversification among individual trees. This ensures that all the features within the superspace are properly utilized by the random forest to form a decision.
- Mitigates errors or overfitting effects of individual models, assuming most of the trees are making correct decisions.
- By ensembling, we are essentially getting the "mean of means", receiving a robust central estimate of the target variable from a collection of slightly different but overlapping weak learners, which combined, provides a robust model.
- Does not require scaling or standardization of features

# 21 Question 21

For this question, we trained a random forest of depth 4, with 3 maximum features and 10 trees on the diamond dataset. We used export_graphviz() from Scikit-Learn and pydot library to visualize the one of the trees in that random forest model. The resulting tree is shown in Figure below.

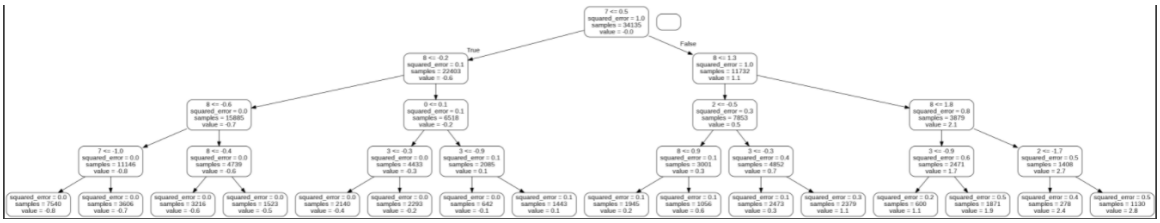

Figure 19: Sample tree from a random forest model trained on diamond dataset

For diamond dataset, from branching at the root node, 'price' is selected. This means that feature 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: 'z'
- Level 2: 'x'
- Level 3: 'y', 'carat', 'color'

'carat', 'clarity', 'x', 'color', 'z' were found to be most salient features in the previous questions. We can say that the most important features for both datasets are similar to the ones whose p-values were significant for linear regression in the previous sections. 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 dataset.

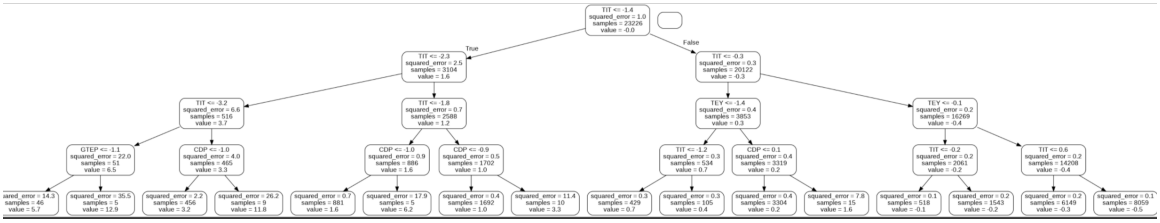

Figure 20: Sample tree from a random forest model trained on CO emission dataset

Similarly for CO emission dataset from the tree diagram most important features are found to be as follows:

- Level 1: 'TIT'
- Level 2: 'TEY'
- Level 3: 'CDP', 'TEY'

During the previous sections we found the important features to be 'GTEP', 'TIT', 'TEY', 'CDP'. Again, there is a large overlap between the important features.
Hence looking at both the datasets we conclude that 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 both datasets.

## 22 Question 22

For lightGBM, we select the following hyperparameters along with associated proper search space:

- Boosting type ('boosting_type'): gradient boosting decision tree (GBDT), Dropouts meet Multiple Additive Regression Trees (DART), Random Forest (RF). 'boosting_type' selects one of four gradient boosting algorithms (we omitted Gradient-based One-Side Sampling (GOSS) as it was throwing out errors during training):
  - GBDT (XGBoost): Default and most widely known boosted decision tree algorithm due to its accuracy, reliability and efficiency without requiring considerable effort for hyperparameter tuning. GBDT treats individual decision trees within an ensemble as weak learners, with the first tree aiming to fit the feature set to target variables and the succeeding trees aiming to reduce the residual error between the predicted target variable and ground truth target variable of preceding trees, with the entire ensemble trained via backpropagation of error gradients. The drawback of GBDT is the associated memory and compute requirements for finding the optimal split points for each individual tree, thereby not scaling well when the number of trees is very large. GBDT also suffers from over-specialization, with the latter trees making meaningful predictions for only a few samples in the dataset (sensitive to a few instances) and not generalizing well for the majority of the other samples.
  - DART: DART solves the overspecialization problem in GBDT by introducing dropout, which is used in neural networks to drop weights for regularization and improving generalization for unseen test data.
  - RF: Random forest, which was introduced in previous questions. In a random forest, each tree along with its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN) as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Thus, according to WLLN, the target variable and tree-decision has finite variance, leading to the overall decision (and any other statistic) of the random forest to converge towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). In other words, a random forest contains a large number of uncorrelated models, with the final target variable being a majority decision fusion from all of the trees.
- Depth of each tree ('max_depth'): 1 to 90 in steps of 10. Increasing the depth of each tree improves the error rate for both the training and test sets, converging to a constant value. For the test set, the error rate might start 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 but increases variance.
- Number of leaves ('num_leaves'): 20 to 990 in steps of 10. 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. LightGBM adds leaf nodes to trees based on performance gains from adding those leaves, regardless of depth of each tree. As a result, it is necessary to tune both the maximum number of leaves as well as the depth of each tree to control model complexity.
- Number of trees ('n_estimators'): 10 to 3900 in steps of 100. Specifies the number of boosted trees to fit. Increasing the number of trees improves and stabilizes 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.

- L1 regularization (alpha) penalty parameter ('reg_alpha'): $[10^{-4}, 10^{4}]$ . 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'): $[10^{-4}, 10^{4}]$. 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 but may hurt model performance.

- Subsampling ratio or bagging fraction ('subsample'): [0.1,1]. Specifies 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_by tree') 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 45 in steps of 5. Controls how often bagging is performed, in terms of epochs.

- Minimal gain to perform split ('min _split _gain'): . 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.

For CatBoost, we select the following hyperparameters along with associated proper search space:

- Feature fraction ('colsample_by level' or 'rsm'): $[0.1, 1]$. 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 1000 in steps of 100. Specifies the number of boosted trees to fit. Increasing the number of trees improves and stabilizes 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.

- Number of leaves ('num _leaves' or 'max _leaves'): 20 to 990 in steps of 10. This is used only for lossguide growing policy. Similar 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 leaf in the tree.

- L2 regularization penalty parameter ('l2 _leaf _reg' or 'reg _lambda'): $[10^{-4}, 10^{4}]$. 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 but may hurt model performance

- 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 training and test sets, converging to a constant value. For the test set, the error rate might start 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 but increases 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.

# 23 Question 23

For this question, we find the optimal hyperparameters in terms of train and test RMSE (average negative test RMSE, higher is better) of boosted trees (LightGBM and CatBoost) using Bayesian optimization on the chosen dataset (diamond dataset). In Bayesian optimization, a Gaussian process is used to approximate the objective function (called surrogate model), which allows priors on the distribution of moments (mean and uncertainty) to propagate forward as the search progresses. The acquisition function decides the next set of hyperparameters to sample from the design space using Monte Carlo sampling with Bayesian Upper-Confidence Bounds (UCB), also known Thompson sampling, which balances exploration and exploitation. Thompson sampling ensures that the acquisition function does not get stuck in a local optimum early in the search, with the exploration parameter decreased as the confidence in the Pareto-frontier grows. We used 10 iterations for the CatBoost regressor, and 20 iterations for the LightGBM regressor, coupled with 10-fold cross-validation. Table below outlines the best set of hyperparameters obtained for both the regressors. From the table, we see that lightGBM performs slightly better than CatBoost in terms of average train and test RMSE on the diamond dataset. However, there is no significant 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.

Table 6

| Regressor | Boosting type | Depth of each tree | number of trees | number of leaves | Min. split gain | Bagging Fraction | Bagging Frequency | L1 reg | L2 Reg | Frature Fraction | Grow Policy, score func | Bagging Temp | Train RMSE(-ve) | Test(-ve) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LightGBM | GBDT | 21 | 710 | 40 | 0.0001 | 0.5 | 35 | 0.0001 | 0.0001 | default | - | - | -0.05959 | -0.2896 |
| Catboost | - | 9 | 210 | N/A | - | - | - | - | 0.01 | 0.8 | ST, Cosine | 2.1 | -0.08968 | -0.2701 |

# 24 Question 24

Figure 21 shows the effects of the 9 hyperparameters of LightGBM on the average negative train and test RMSE for diamond dataset obtained from Bayesian hyperparameter tuning. Since Bayesian optimization is stochastic, we also plot trend lines to showcase the overall tendency of each hyperparameter.

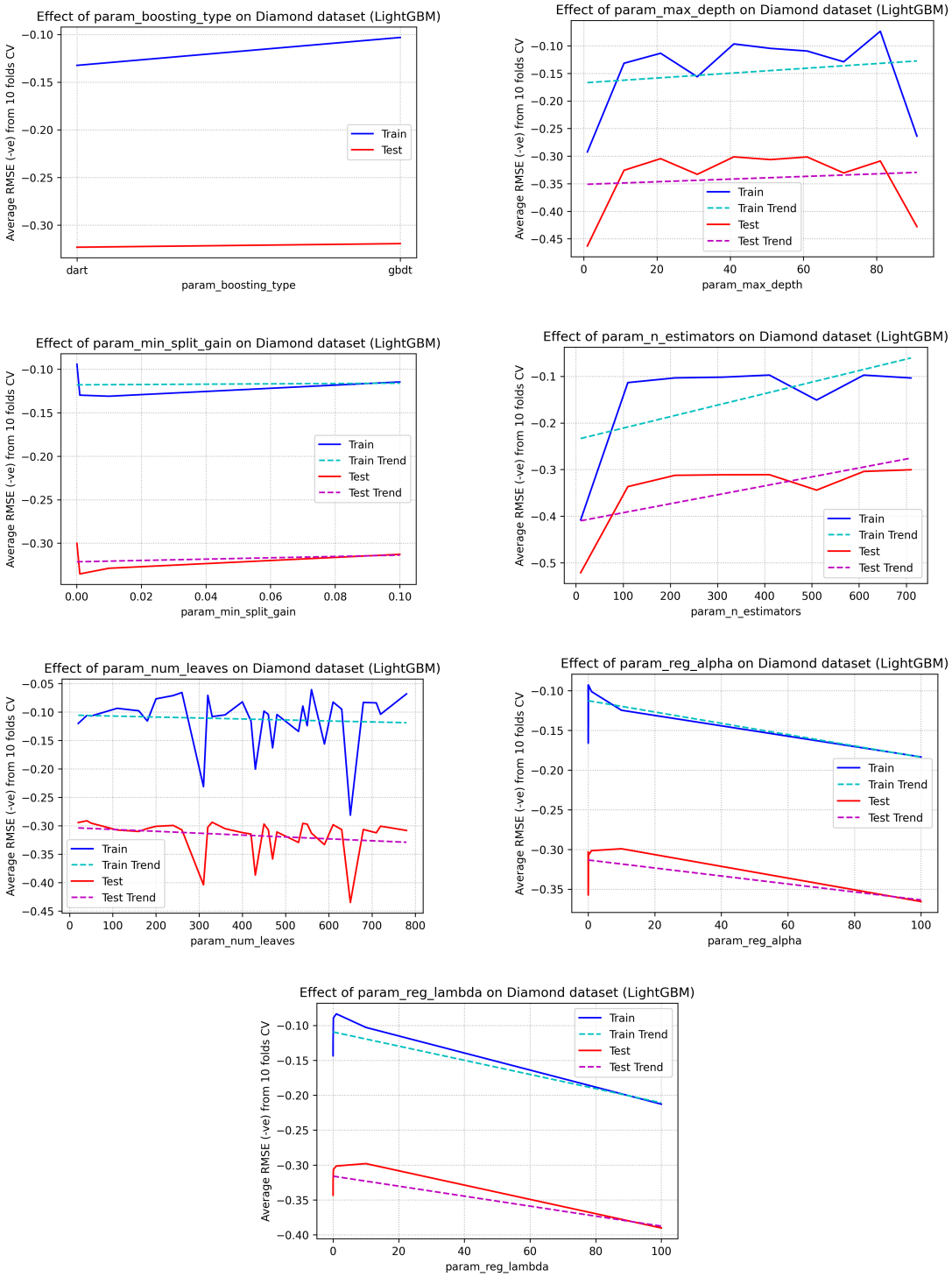


Figure 21: Effect of the hyperparameters of LightGBM individually on diamond dataset

1. Boosting Type: From Figure, we see that GBDT and RF perform similar to each other, while both jointly outperform DART. Theoretically, DART should improve generalizability of GBDT

by introducing dropout. However, DART also has implicit hyperparameters that require tuning, such as model seed, the probability of skipping dropout during an iteration, dropout rate and whether to use XGBoost dropout or uniform dropout, which we did not tune due to compute constraints. Too many floating parameters can result in unexpected performance of DART.

2. Depth of each tree: From Figure, 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.

3. Minimal gain to perform split: From Figure, we see that the both training and test RMSE improves slightly with an increase in the gain ratio. 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. However, the graph does not show the full picture as the Bayesian agent did not cover the full range of the ratio. More aggressive bagging is likely to cause drop on model performance due to regularization effects.

4. Number of trees: From Figure, we do not see any performance gains or losses 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.

5. Number of leaves ('num_leaves' or 'max _leaves'): 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 shows slight improvement in performance as the number of leaves increases.

6. L1 regularization term: From Figure, 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.

7. L2 regularization term: From Figure, 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.

8. Bagging frequency: From Figure, 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.

9. Bagging fraction: From Figure, we see 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

The following figure shows the effects of the 8 hyperparameters of CatBoost on the average negative train and test RMSE for diamond dataset obtained from Bayesian hyperparameter tuning. Since Bayesian optimization is stochastic, we also plot trend lines to showcase the overall tendency of each hyperparameter.

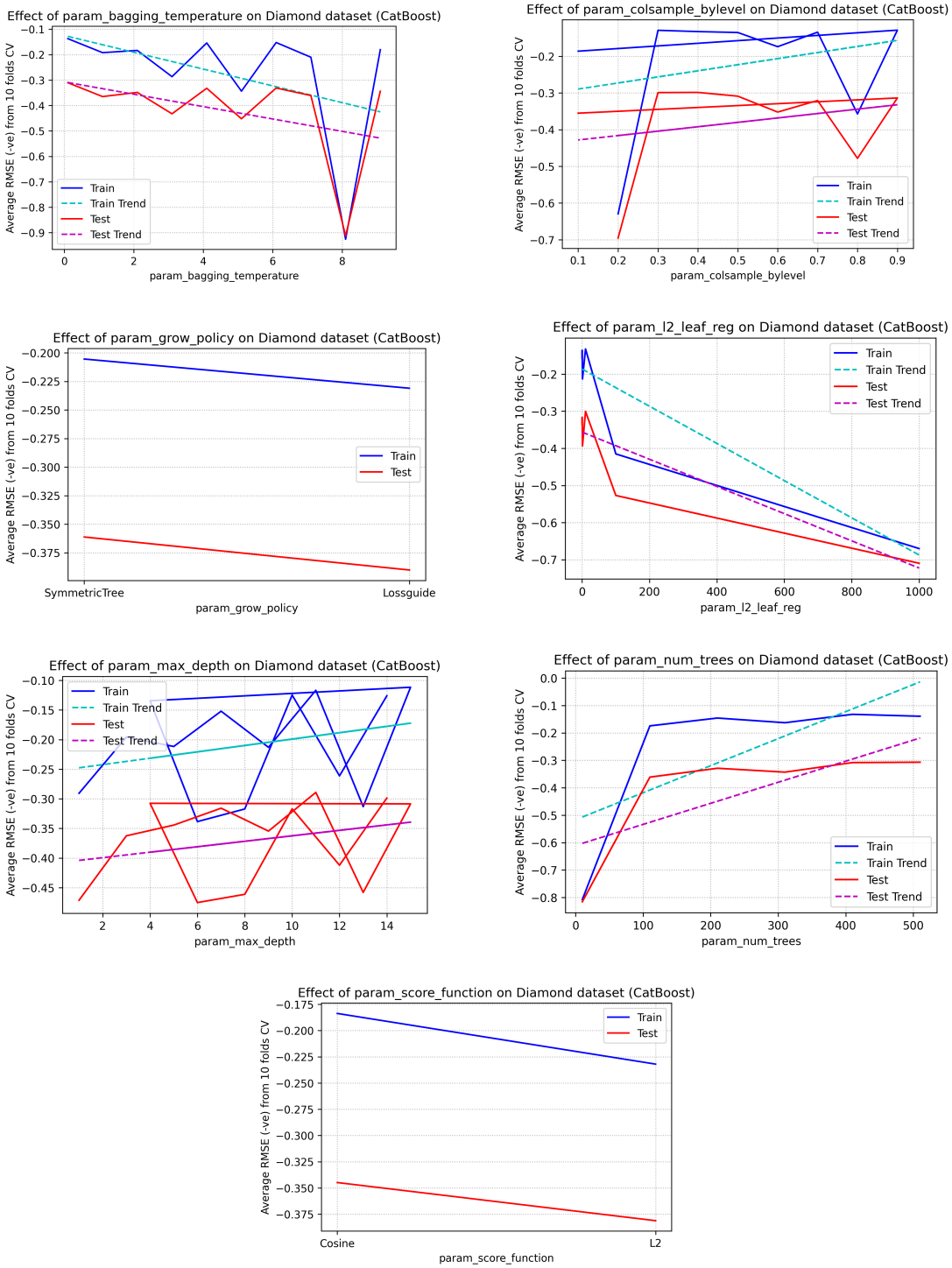


Figure 22: Effect of the hyperparameters of Catboost individually on diamond dataset

1. Bagging temperature: From the figure(1), we see that as bagging temperature increases, the tran and test RMSE degrades. 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. Since, we are already supplying

top 10 salient features to CatBoost, assigning extreme weights to these features can hurt model performance.

2. Feature fraction: From Figure, 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.

3. Grow policy: From Figure, 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.

4. L2 regularization: From Figure, 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.

5. Depth of each tree: From Figure, 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.

6. Number of leaves (only for lossguide): 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, 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.

7. Number of trees: From Figure , 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.

8. Score function: From Figure, 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

## 25 Question 25

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 in earlier questions. Here, we report the performance of the best model (hyperparameter tuned either using 10-fold grid search cross validation or Bayesian optimization) found via 10-fold grid-search cross validation for each dataset in the following table. The best model for diamond dataset was MLP whereas for CO emission dataset it was polynomial regression.

| Model | Best average train RMSE | | Best average test RMSE | |
|---|---|---|---|---|
| - | Diamond | CO-Emission | Diamond | CO-Emission |
| Linear | -0.302 | -0.66 | -0.302 | -0.687 |
| Lasso | -0.302 | -0.65 | -0.303 | -0.685 |
| Ridge | -0.3 | -0.66 | -0.301 | -0.687 |
| Polynomial | -0.13 | -0.59 | -0.15 | -0.61 |
| MLP | -0.1261 | -0.4703 | -0.1420 | -0.6166 |
| Random Forest | -0.1195 | -0.2611 | -0.1948 | -0.6232 |
| LightGBM | -0.05959 | - | -0.2896 | - |
| Catboost | -0.08968 | - | -0.2701 | - |

We see that usually, the negative training RMSE is higher (absolute 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.

## 26 Question 26

In this question, we report the OOB error for the random forest model in previous question. The OOB error for the best random forest models on the two datasets are as follows:

- OOB, diamond: 0.8980
- OOB, CO emission: 0.6640

In a random forest, each tree along with its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN) as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Each decision tree is trained on a different and independent random subspace, forcing variation, decorrelation and diversification among individual trees. OOB score is a technique for validating random forest models, defined as the number of correctly predicted samples (rows) from the out of bag sample rows via majority decision fusion. An out-of-bag sample is defined rows in the original dataset that was not present in the bootstrap samples of a particular decision tree. Since OOB is being calculated on unseen test data, it serves as a form of validation score for the ensemble. $R^2$ (coefficient of determination), on the other hand, is defined as the ratio of the variance in the target variable that can be predicted by the features in the dataset (also known as goodness of fit). It includes all the data in the training (and validation) set regardless of whether a tree within an ensemble saw some of the samples or not, along with decision from all the decision trees. For $R^2$ score calculation, separate steps are required for training and validation score calculation.

## 27 Question 27

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.

Table 7: Statistics for each hashtag file

| Hashtag | Average number of tweets | Average number of followers | Average number of retweets |
|---|---|---|---|
| #gohawks | 292.48 | 2217.92 | 2.013 |
| #gopatriots | 40.954 | 1427.25 | 1.408 |
| #nfl | 397.02 | 4662.375 | 1.534 |
| #patriots | 750.89 | 3280.463 | 1.785 |
| #sb49 | 1276.85 | 10374.16 | 2.527 |
| #superbowl | 2072.11 | 8814.96 | 2.39 |

The code snippet for calculating that is -

```
def report_statistics(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
        max_time = 0
        min_time = np.inf
        total_followers = 0
        total_retweets = 0
        total_tweets = len(lines)
        for line in lines:
            json_obj = json.loads(line)
            if json_obj['citation_date'] > max_time:
                max_time = json_obj['citation_date']
            if json_obj['citation_date'] < min_time:
                min_time = json_obj['citation_date']
            total_followers += json_obj['author']['followers']
            total_retweets += json_obj['metrics']['citations']['total']
        avg_tweets_per_h = total_tweets * 3600 / (max_time - min_time)
        avg_followers_per_tweet = total_followers / total_tweets
        avg_retweets_per_tweet = total_retweets / total_tweets
        print(filename)
        print('Average number of tweets per hour: ', avg_tweets_per_h)
        print('Average number of followers of users posting the tweets per tweet: ',
              avg_followers_per_tweet)
        print('Average number of retweets per tweet: ', avg_retweets_per_tweet)
        print('-' * 50)
```

# 28 Question 28

Plot “number of tweets in hour” over time for #SuperBowl and #NFL (a bar plot with 1-hour bins).

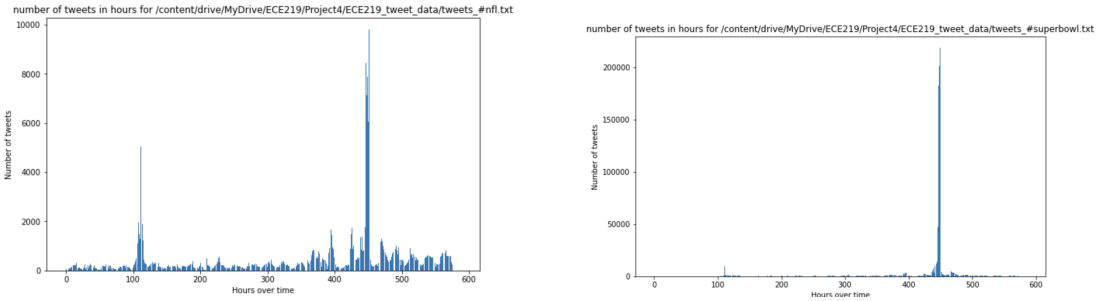


Figure 23: “Number of tweets in hour” over time for #NFL and #SuperBowl

Our inference from these plots is that the number of tweets surges around hour 450th index for both hashtags. Also the distribution of tweets for #NFL is kind of bimodal normal.

# 29 Question 29

We have chosen our task - Developing a Library of Prediction Tasks given a tweet:

- Predict how likely it is that a tweet belongs to a specific team fan
- Predict the number of retweets from a tweet
- Predict positive/negative tweets in a specified time frame, with some analysis using Wordcloud.

## 29.1 Predict the tweet belongs to a specific team fan

### 29.1.1 Task Description

We are using the data from tweets_#superbowl.txt dataset of 5.2GB for this purpose. We read the file as a json object, and it has multiple keys like firstpost_date, title, url, tweet, hashtags, retweet_count, followers_count, citations.

For the purpose of this prediction task we are interested in finding which fan team does the tweet belong to. We have formulated the task as a binary classification problem for prediction between the teams - Hawks and Patriots (teams that have entered the superbowl).

### 29.1.2 Data Pre-processing

Before performing our actual feature selection and prediction task we have performed some pre-processing on the tweets dataset as explained below :

- Tweet is extracted from the tweets dataset using the referencing json_obj['tweet']['text'].
- We then proceed to build the labels for each tweet, as this is not inherently present in the dataset. Labels over here refer to which team's fan is the tweet inclining to.
- We make use of regex and find all the hashtags in the tweet, and then scan them to see if they match the pattern of hawks or patriots.
- One-hot label encoding is employed where [0,1] would mean the tweet is from a Patriots fan, and [1,0] would mean it's from a Hawks fan.
- We also remove the # from the tweet text before feature selection because we want the words associated with the hashtags to also be considered by our feature selection models.
- The tweets which have labels belonging to both patriots and hawks or neither are removed from the dataset. So, the size of the dataset now would be 158153 tweets.
- Train_test_split function is used on this processed dataset, with a test_size of 0.2.

### 29.1.3 Feature Selection

- We employ the feature selection methods learnt in Project 1 through 3 for this task.
- Each tweet in the dataset is first cleaned. Cleaning involves removing the HTML artefacts. This cleaned data is then passed through a lemmatization function which uses pos_tagging.
- A TfidfVectorizer(stop_words='english', min_df=3) is then fit to the lemmatized data features. This will assign weights to each word in our dataset and build a model out of it.
- To reduce the number of features we select the top 50 features using SVD decomposition, Truncated SVD() function.
- The above feature selection is projected on both x_train and x_test. So, the final shape of x_train is (126522, 50), and x_test is (31631, 50). This will reduce the processing time, and chances of model overfitting as we are only retaining the important features for the task.

### 29.1.4 Team Classification

- The first model used for Team Classification is LogisticRegression().
- A parameter grid search is performed using GridSearchCV() for a 4 fold cross validation. 'L2' and no penalty are the penalty parameters, along with C ranging from 0.01 to 100.
- The best model from this grid search is - LogisticRegression(C=0.01, penalty='none').
- The second model employed for comparison purpose is a RanfomForestClassifier().
- A parameter grid search is performed here as well with max_depth varying from 10 to 200.
- The best model found from grid search is - Random Forest Classifier with max_depth = 50.

### 29.1.5 Model Evaluation

The results of the above described models are summarised in the table below.

Table 8: Evaluation Results for Team Prediction

| Model | Accuracy | Recall | Precision | F1-Score |
|---|---|---|---|---|
| Logistic Regression | 0.980 | 0.987 | 0.978 | 0.982 |
| Random Forest | 0.980 | 0.980 | 0.984 | 0.982 |

The confusion matrix for the prediction tasks are -

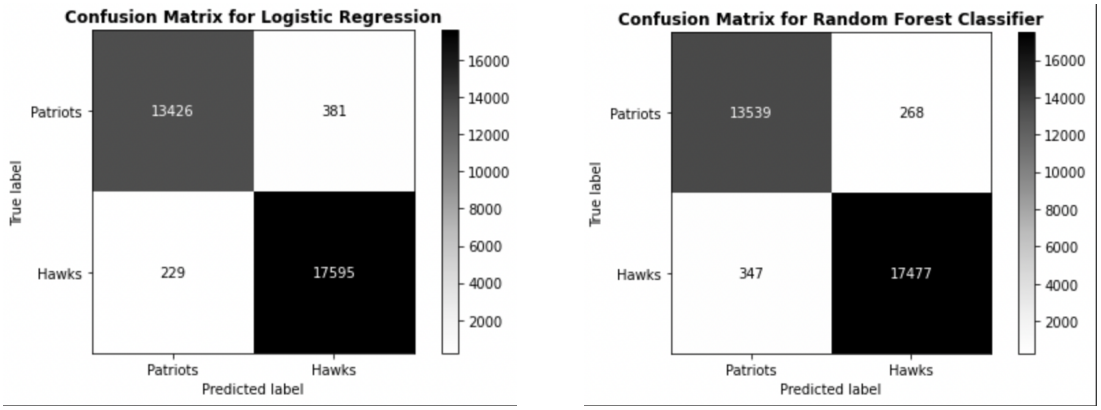


Figure 24: Confusion Matrix

Our inference from these results is that the model does pretty well for the task of classifying between the fan teams of Patriots and Hawks from the superbowl dataset. This inference is backed by the accuracy metrics and the confusion matrix where a clear binary classification between 2 classes is observed.

## 29.2 Predict the number of retweets from a tweet

### 29.2.1 Task Description

We used two approached for this task. For one approach we used all six datasets. We trained our model on five datasets namely 'gohawks', 'gopatriots', 'patriots', 'sb49', 'superbowl' and then tested it on 'nfl'. We also implemented models using just superbowl dataset which is 5.2GB in size. The goal was to predict number of retweets after the tweet was given as a input.

### 29.2.2 Data Preprocessing

Before the actual feature extraction and prediction of number of tweets we cleaned the data and converted it to a format which could be fed to all the networks.

- Tweet was extracted from the dataset by referencing the json object.
- Retweet count was also extracted using similar procedure from the json object which is our label.
- We removed hashtags from the input data.
- The output was standardized.

### 29.2.3 Feature Extraction

- Each tweet is cleaned. This cleaned data was passed through a lemmatization function which used pos tagging.
- We used tfidf to get our data features then further employed svd to select top 50 features.
- Then we divide the dataset in x_train and x_test. In case of one approach x train was 5 datasets namely 'gohawks', 'gopatriots', 'patriots', 'sb49', 'superbowl, whereas x_test was 'nfl'. Whereas in the other case we used only superbowl dataset where the data distribution was x_train with 126522, 50) samples and x_test with (31631, 50) samples.

### 29.2.4 Models

- MLP Regressor : The MLP from Scikit learn was trained on four datasets and tested on one dataset with relu activation function and adam optimizer. We used the following combinations of hidden units=[50,100,200,300,500,600] to build our model consisting of three layers. After the gridsearch we chose the model which gave us the best RMSE value.The results for this have been summarized in the coming section.
- Gradient Boosting Regressor: We used sklearn's GB regressor to train on the data. We again, trained on four datasets and tested on one dataset. We used the following parameters to build our model:
  - max depth: [10, 20, 50, 100]
  - max features: ['auto', 'sqrt']
  - min samples leaf: [1, 2, 4],
  - min samples split: [2, 5, 10]
  - n estimators: [5,10,50,100,200, 400, 600, 800]
- Linear Regression: We used Statsmodels.api's Linear regression on superbowl dataset.
- Support Vector Machine: We used SVR on superbowl dataset which is used for regression applications.

### 29.2.5 Model Evaluation

The results of the models used on all six datasets are summarized in the table below.

Table 9: Evaluation Results for Predicting Number of Retweets

| Model | hyperparameters | RMSE |
|---|---|---|
| MLP Regressor | Hidden Layers: (300, 600, 600) | 22768.8339 |
| Gradient Boosting Regressor | max depth= 50<br>max features= auto<br>min samples leaf= 2<br>min samples split= 2<br>n estimators= 10 | 38721.7846 |

From the RMSE it can be seen that MLP regressor works better than the gradient boosting regressor even though we expected the gradient boosting regressor to outperform neural network. A decision tree is a greedy algorithm that can only make progress when each base features carries some reasonable amount of information. If they do, it can progress one feature at a time. In this sense MLP is not a greedy algorithm. They do not make choices one parameter independent of all others. When they make a gradient step, they are updating parameters that effect all parts of the image at all scales. And hence in this case the neural network performs better than the decision tree.

Table 10: Evaluation Results for Predicting Number of Retweets

| Model | MSE |
|---|---|
| Linear Regression | 0.9992 |
| SVM | 1.02041 |

As can be seen from MSE values of the both algorithms work similarly on the given dataset. Standardized output was given to these algorithms which reduced the MSE value significantly. Also, these were tested on only one dataset instead of all six datasets.

For large amount of data MLP regressors supersede the performance of all other algorithms and hence are preferred over other algorithms.This conclusion is backed by RMSE value of the dataset.

## 29.3 Predict Positive/Negative tweets in a specified time frame

### 29.3.1 Task Description

We used the data from #gopatriots, #gohawks and #nfl datasets for this task. The goal was to classify tweets according to their sentiment in a specified duration. We read the file as a json object then using TimeBlob found the target variable.
For this classification task we formulated binary classifiers to predict between positive and negative tweets. We used Logistic Regression and Random Forest for the classification. The models were tested on each dataset separately.

### 29.3.2 Data Preprocessing

Before performing our actual feature selection and prediction task we have performed some preprocessing on each of the tweets dataset as explained below :

- Tweet is extracted from the tweets dataset using the referencing json_obj['tweet']['text'].

- We then proceed to build the labels for each tweet, as this is not inherently present in the dataset. Labels over here refer to whether the tweet is positive or negative.
- We use TextBlob library for assigning labels to our data. Textblob is a text processing library. The sentiment analysis section gives two outputs polarity and subjectivity. We use the term polarity to assign the label to each tweet. If polarity is less than equal 0 then we assign a zero value to the label, whereas we assign the label one for polarity greater than zero.
- The function taken into account also the time at which the tweets were posted and hence a good conclusion can be made regarding the game in that specific duration.
- We consider 3403 samples from gohawks dataset, 412 samples from gopatriot dataset and 4994 samples from the nfl dataset.
- Train_test_split function is used on each dataset, with a test_size of 0.2.

#### 29.3.3 Feature Selection

- We employ the feature selection methods learnt in Project 1 through 3 for this task.
- Each tweet in the dataset is first cleaned. Cleaning involves removing the HTML artefacts. This cleaned data is then passed through a lemmatization function which uses pos_tagging.
- A TfidfVectorizer(stop_words='english', min_df=3) is then fit to the lemmatized data features. This will assign weights to each word in our dataset and build a model out of it.
- To reduce the number of features we select the top 50 features using SVD decomposition, Truncated SVD() function.
- The above feature selection is projected on both x_train and x_test.

#### 29.3.4 Classification

- The first model used for Team Classification is LogisticRegression().
- A parameter grid search is performed using GridSearchCV() for a 4 fold cross validation. 'L2' and no penalty are the penalty parameters, along with C ranging from 0.01 to 10
- The second model employed for comparison purpose is a RanfomForestClassifier().
- A parameter grid search is performed here as well with max_depth varying from 10 to 200.

#### 29.3.5 Model Evaluation

The results of the above described models for Gohawks dataset are summarised in the table below.

Table 11: Evaluation Results for Gohawks

| Model | Accuracy | Recall | Precision | F1-Score |
|---|---|---|---|---|
| Logistic Regression | 0.8781 | 0.9500 | 0.8913 | 0.9198 |
| Random Forest | 0.8678 | 0.9660 | 0.8689 | 0.9149 |

The best model for logistic regression for the given dataset was found to be C=0.01, penalty='none', randomstate=42. Whereas for Random forest it was maxdepth=30, randomstate=42 The confusion matrix for the prediction tasks are -

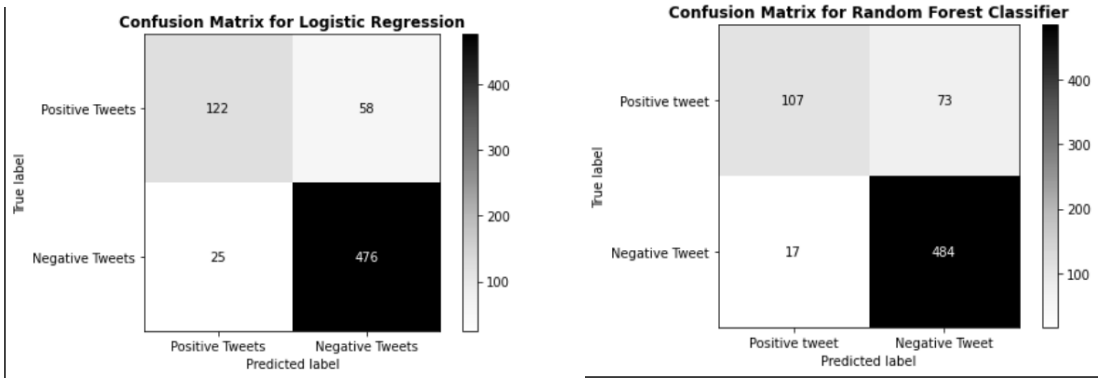


Figure 25: Confusion Matrix for gohawks dataset

The results of the above described models for Gopatriots dataset are summarised in the table below.

Table 12: Evaluation Results for Gopatriots

| Model | Accuracy | Recall | Precision | F1-Score |
|---|---|---|---|---|
| Logistic Regression | 0.8674 | 0.9692 | 0.875 | 0.9197 |
| Random Forest | 0.8795 | 1 | 0.8666 | 0.9285 |

The best model for logistic regression for the given dataset was found to be C= 10, penalty='none', randomstate=42. Whereas for Random forest it was maxdepth = 10, randomstate=42 The confusion matrix for the prediction tasks are -

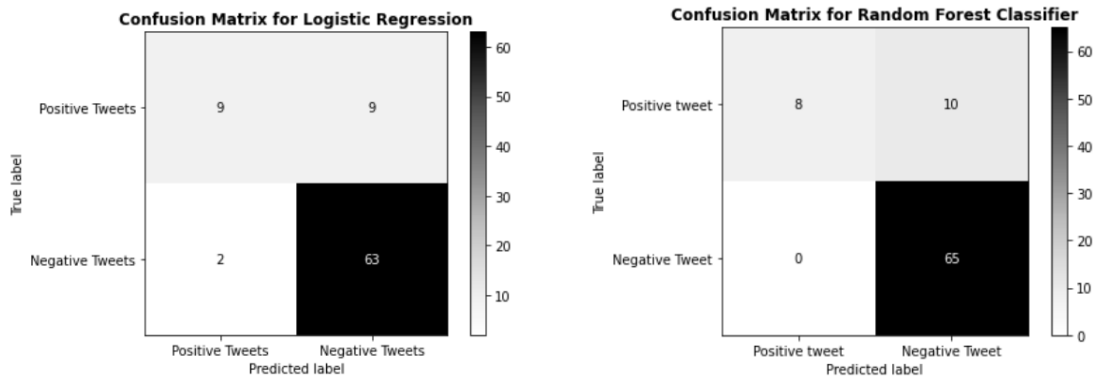


Figure 26: Confusion Matrix for gopatriots dataset

The results of the above described models for NFL dataset are summarised in the table below.

Table 13: Evaluation Results for NFL

| Model | Accuracy | Recall | Precision | F1-Score |
|---|---|---|---|---|
| Logistic Regression | 0.8598 | 0.9697 | 0.8631 | 0.9133 |
| Random Forest | 0.8708 | 0.9829 | 0.8657 | 0.9206 |

The best model for logistic regression for the given dataset was found to be C = 10, penalty='none',

randomstate=42. Whereas for Random forest it was maxdepth= 50, randomstate=42 The confusion matrix for the prediction tasks are -

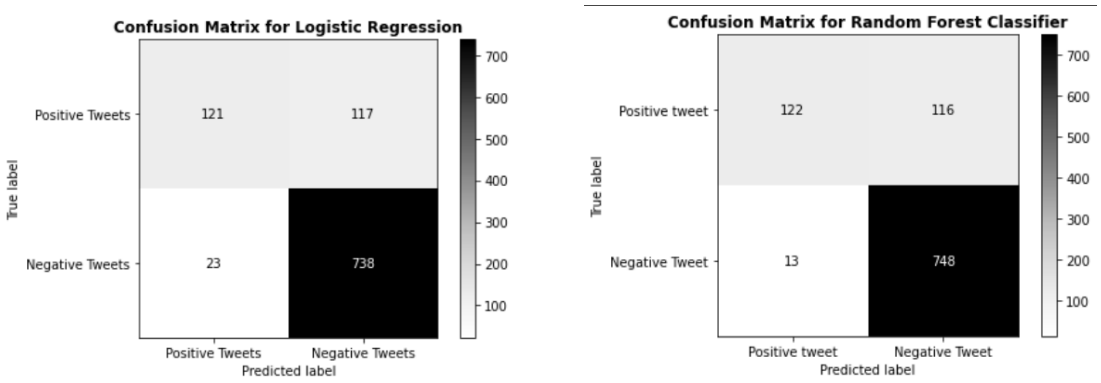


Figure 27: Confusion Matrix for nfl dataset

Our inference from these results is that the model does pretty well for the task of classifying between the positive and negative sentiment from the three dataset. This inference is backed by the accuracy metrics and the confusion matrix where a clear binary classification between 2 classes is observed. From the values the trend of the game for specified time duration can also be observed.

### 29.3.6 Sentiment Analysis with respect to time

Analysis in particular time frames has been done on the #gohawks and #gopatriots dataset. We notice an interesting trend which explains why our classifiers perform so well on these datasets.

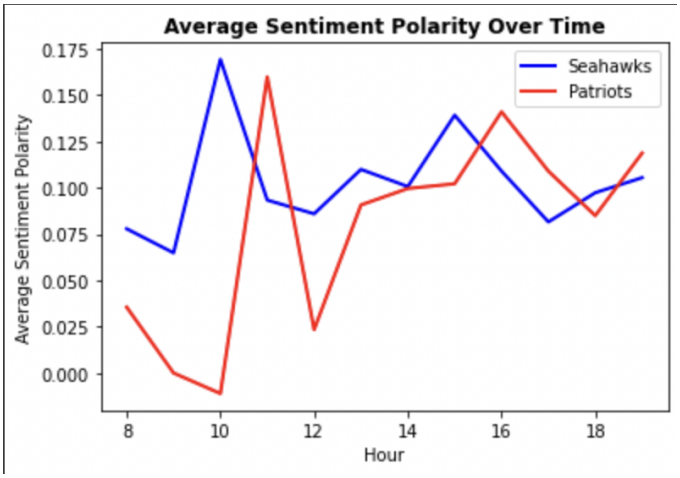


Figure 28:
Average sentiment polrity over time

In the figure above we plot the average sentiment polarity over time for the seahawks and patriots. If we look at the peaks in both datasets we notice that when Seahawks has a high polarity, Patriots has a very low polarity roughly around the 10th hour. And at the 11th hour when patriots peaks to a high polarity, seahawks is at its lowest polarity. Also, to start off with Seahawks in general has a very high average polarity compared to Patriots. This says that the Seahawks fans are more happy with their team's performance throughout till the 14th hour, when Patriots also catches up and both teams have a similar polarity.

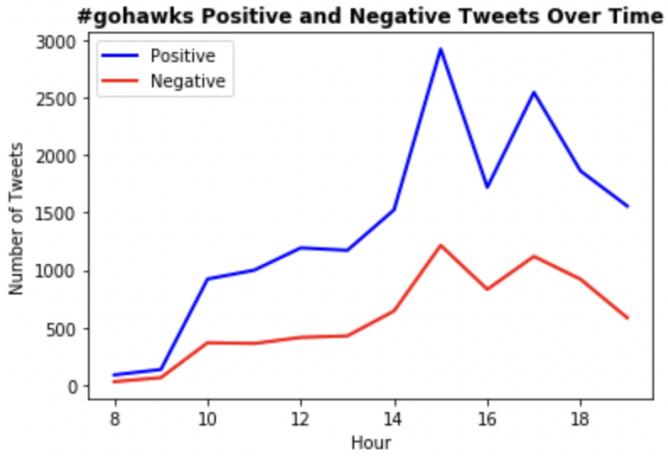


Figure 29:
#gohawks positive and negative tweets over time

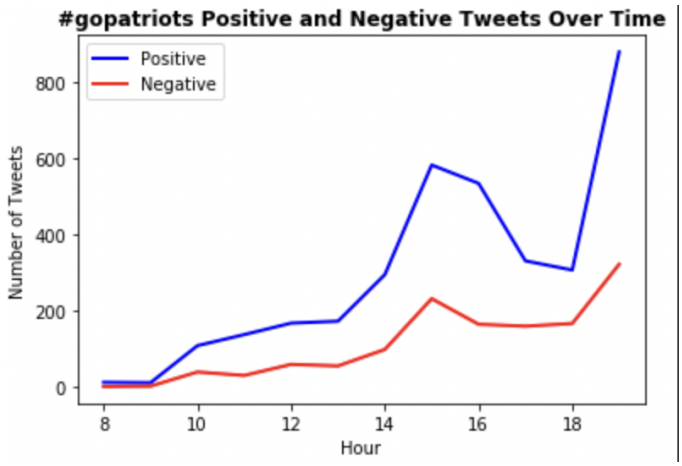


Figure 30:
#gopatriots positive and negative tweets over time

From the figures above we can conclude that the positive and negative tweets are well separable with respect to time in both datasets. And on an average, the number of positive tweets is higher than the number of negative tweets.

### 29.3.7 Analysis using Wordcloud

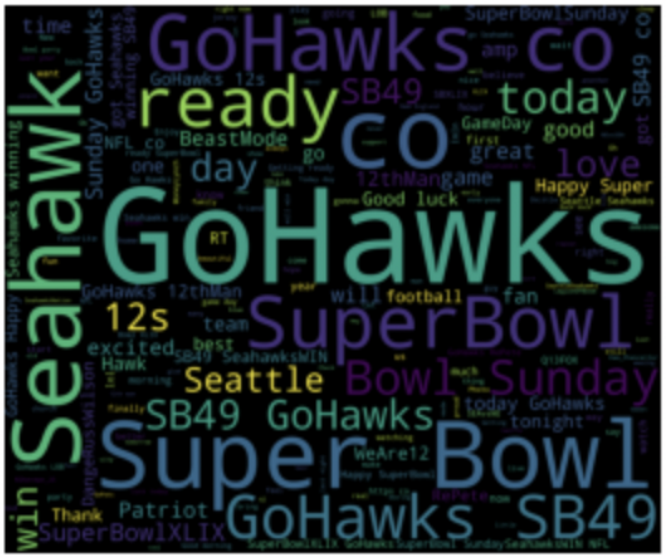


Figure 31:
Wordcloud analysis for positive tweets in #gohawks


Figure 32:
Wordcloud analysis for negative tweets in #gohawks

A word cloud (also called tag cloud or weighted list) is a visual representation of text data. Words are usually single words, and the importance of each is shown with font size or color. Python fortunately has a wordcloud library allowing to build them.

Obviously the words taking up more space in the cloud are the ones common to both kinds of tweets - GoHawks, SeaHawk, Superbowl. However if we look closely there are a lot of positive words associated with higher polarity tweets like - good, great , excited, good luck, happy, love, first. Whereas in the cloud for negative tweets we don't find such words present and it is shadowed with words like hate, little, mean, sorry. This is a great visualisation tool that aids us in understanding our dataset better.