Large-Scale Data Mining: Models and Algorithms ECE 219 Winter 2023
Prof. Vwani Roychowdhury
UCLA, Department of ECE

Project 4: Regression Analysis and Define Your Own Task!

Sudeeksha Agrawal, Tazeem Khan, Vamsi Krishna Pamidi

UID: 305928941,105946724,805945580

Introduction:

Regression is a technique used to find the relationship between a dependent variable and one or more independent variables. In this project, we explored different feature engineering and model selection methods that are specific to regression.

QUESTION 1: Standardize feature columns and prepare them for training

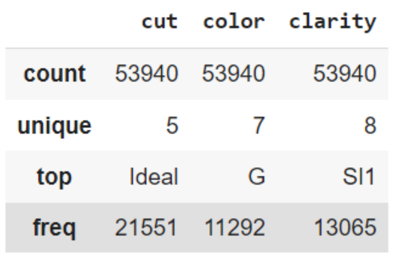
Using the Diamonds dataset:

df = pd.read_csv("C:/Users/Tazeem Khan/Downloads/diamonds.csv")

Dataset Information:

- 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 = z / mean(x, y) = 2 * z / (x + y) (43–79)
- table: width of top of diamond relative to widest point (43–95)

Column wise data inspection:
For Categorical columns:



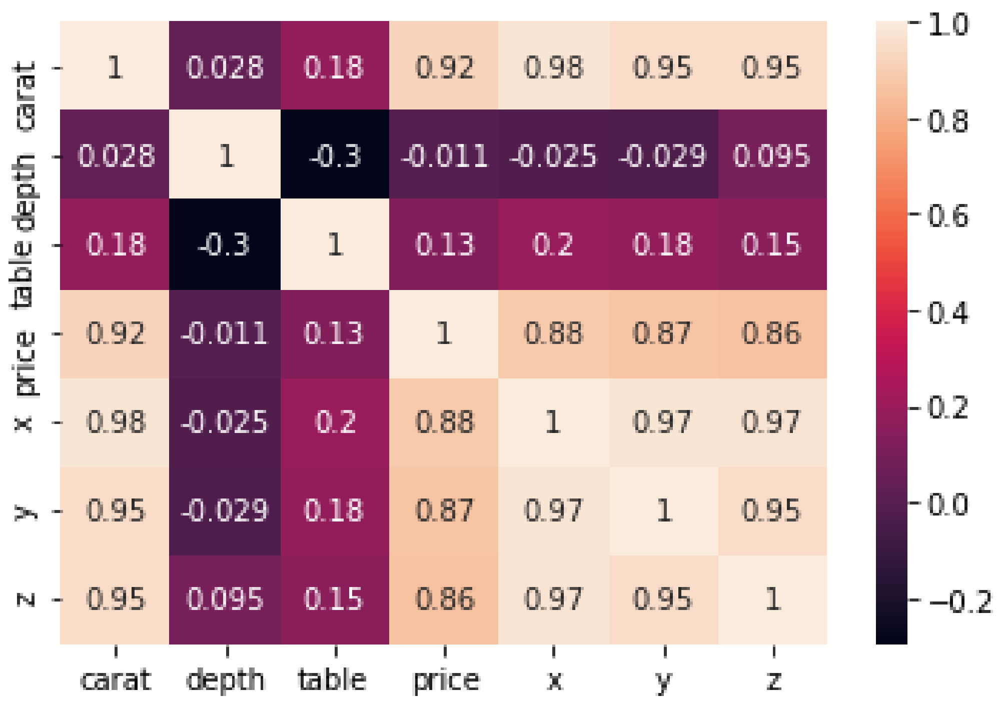
	cut	color	clarity
count	53940	53940	53940
unique	5	7	8
top	Ideal	G	SI1
freq	21551	11292	13065

For Numeric columns:

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3934.801557	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.442321	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	327.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	952.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2403.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5327.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

Label encoding is done for the columns cut, color, clarity as the values are continuous and it doesn’t make sense to use one- hot encoding.

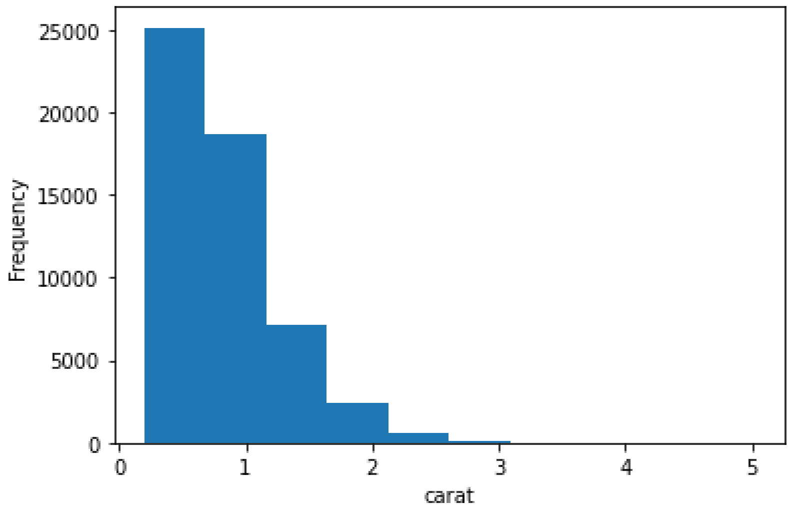
Question 1.1:
Plot a heatmap of the Pearson correlation matrix of the dataset columns. Report which features have the highest absolute correlation with the target variable. In the context of either dataset, describe what the correlation patterns suggest.

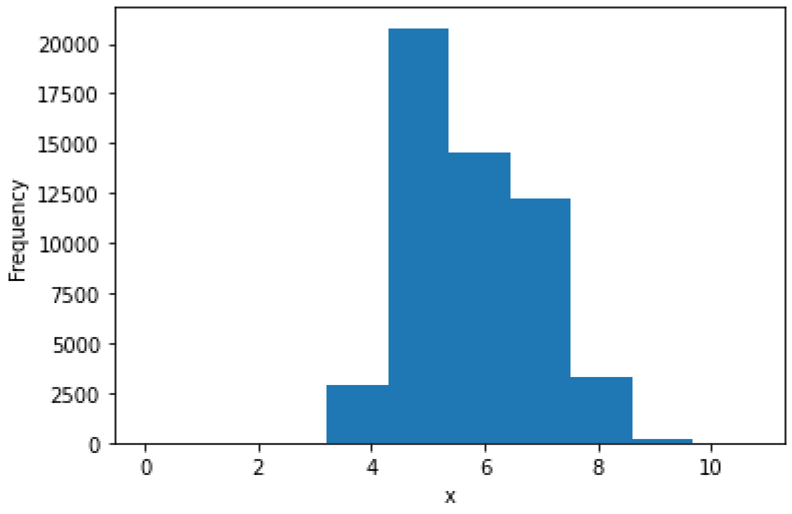


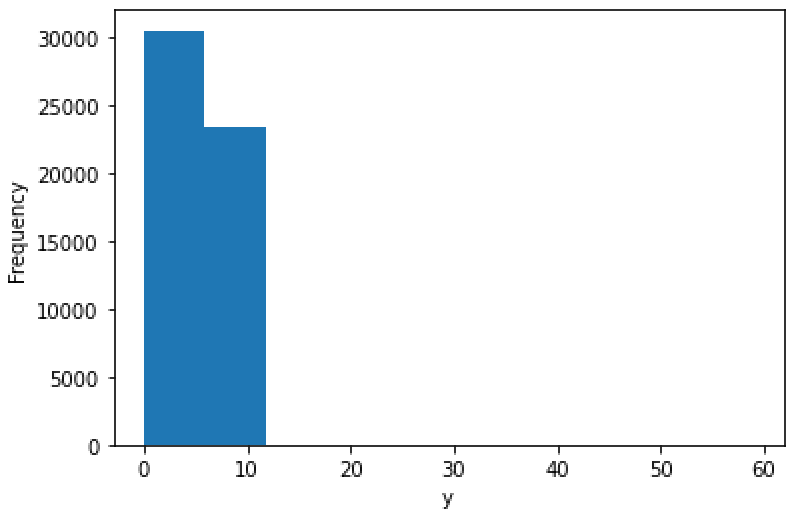
The target variable price is most correlated to the variable carat and then to the variables x, y, and z. The correlation pattern suggests that the variables carat, x, y, and z are more correlated to each other and to the target variable when compared to the variables table and depth.

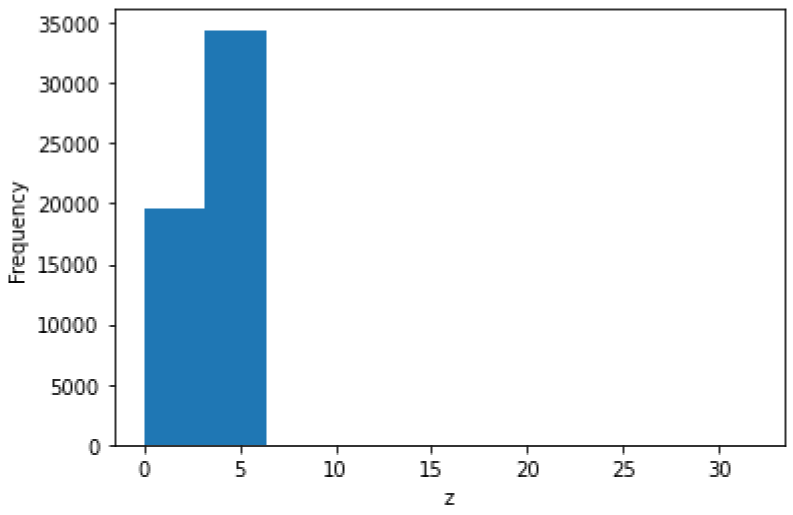
Question 1.2:
Plot the histogram of numerical features. What preprocessing can be done if the distribution of a feature has high skewness?

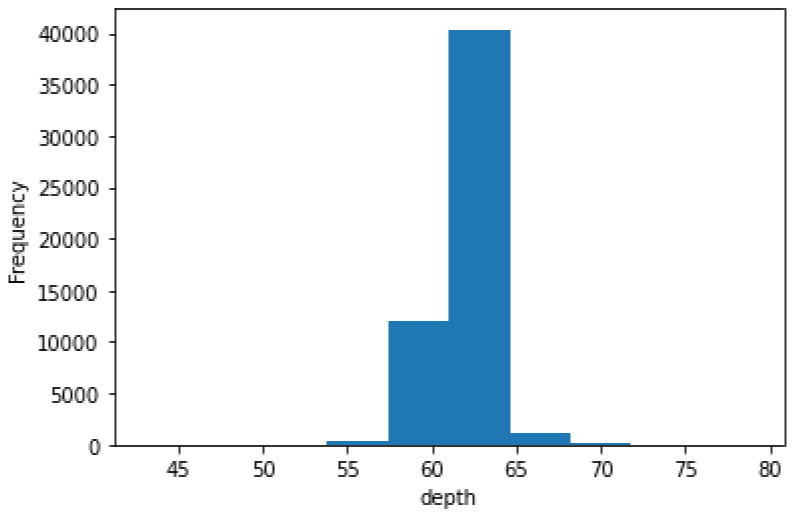
The figures below show the histogram of the values in each column.

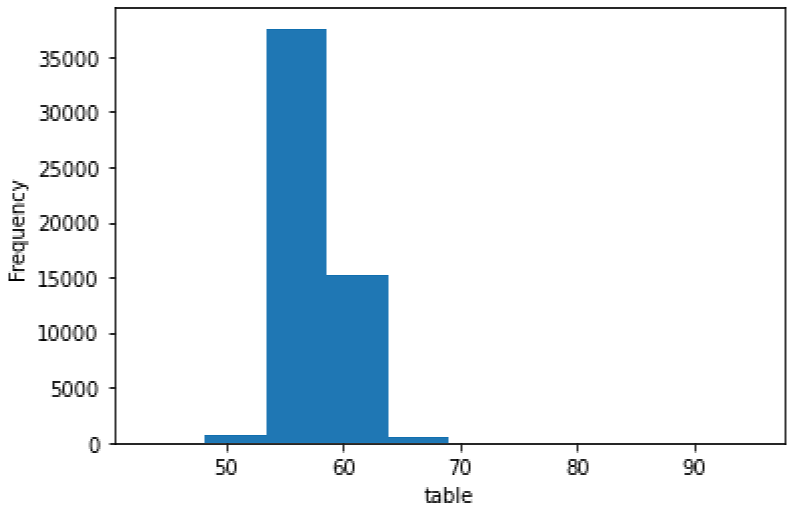





30000
25000
20000
15000
10000
5000
0
Frequency
0
10
20
30
40
50
60
y


35000
30000
25000
20000
15000
10000
5000
0
Frequency
0
5
10
15
20
25
30
z


40000
35000
30000
25000
20000
15000
10000
5000
0
Frequency
45
50
55
60
65
70
75
80
depth

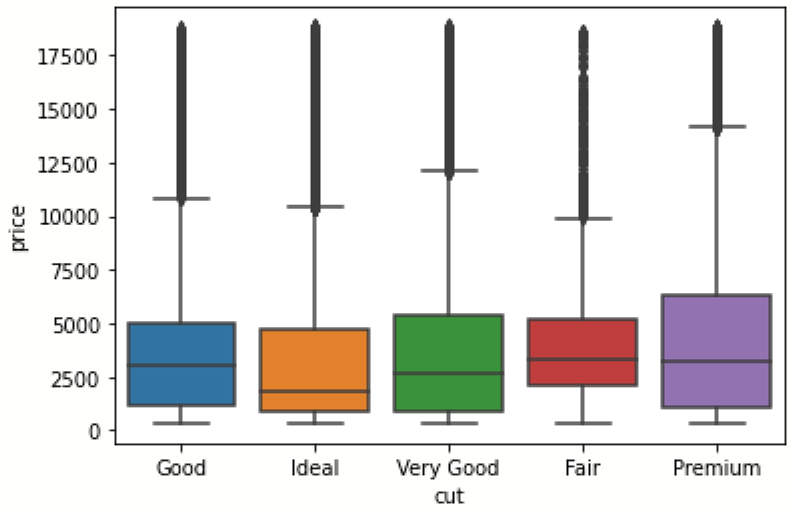


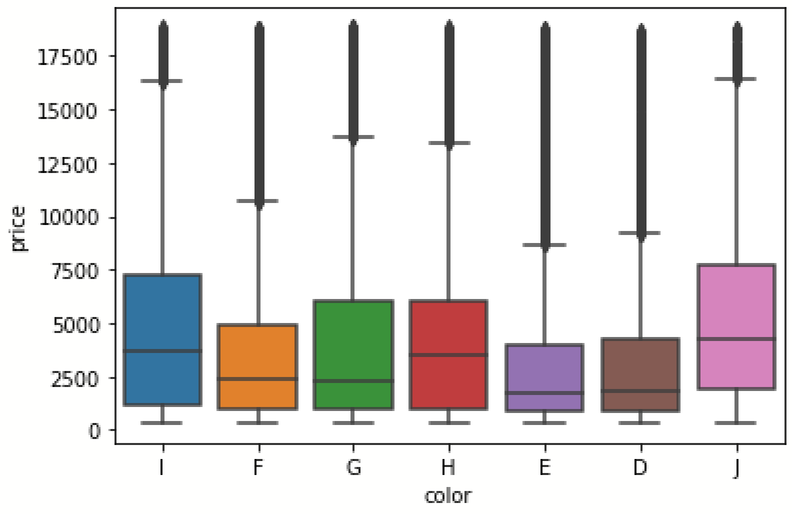
- From the figures presented above, it can be observed that the histograms corresponding to the table depth appear to be nonskewed. And the features carat, x, y, and z appear to be skewed.
- To tackle the skewness problem in the variable of the data set, we can use the exponential or logarithmic transformations on each of these variables depending upon what side the variables are skewed towards.

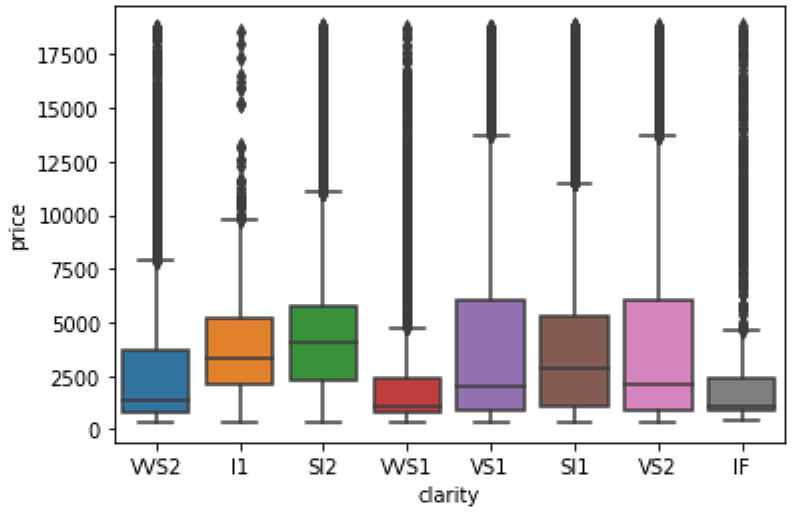
Question 1.3:

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

The figures below show the box plot of categorical features vs target variable.







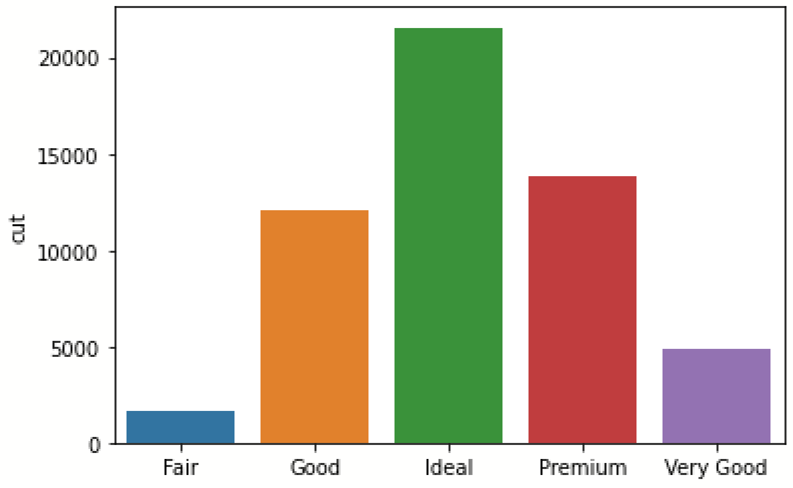
From the box plots presented above, the below observations were made:

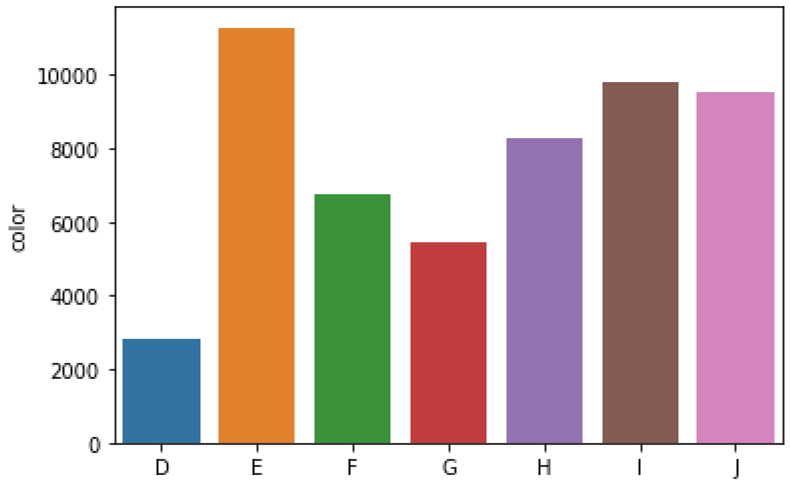
- For both the cut and clarity columns, the median price of the diamonds is higher for the categories like fair and SI2, whereas that should have been higher for Ideal. And there is no linear relationship between the median value and the category type.
- Then coming to the outliers, the number of outliers varies by a more significant extent in the clarity variable, where there are more outliers for the class WS1, and IF1 when compared to the other classes.
- And in the color box plot, it can be observed that F, G, E, and D are skewed more than the other colors.

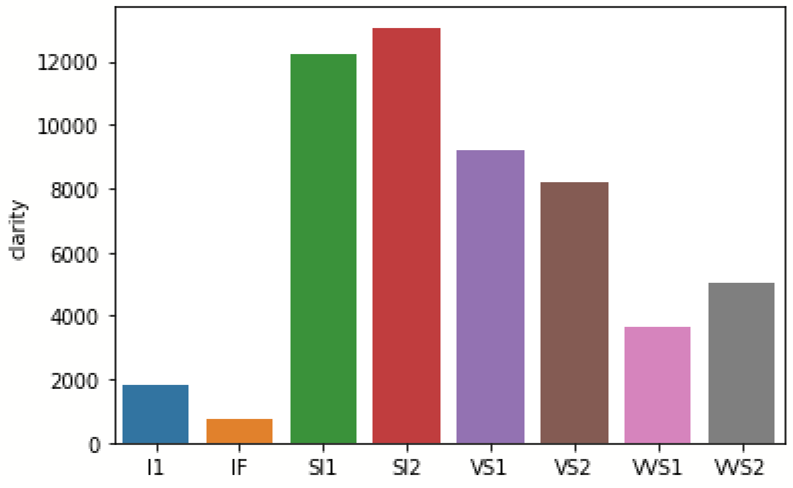
Question 1.4:

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

Plot showing the counts by color, cut and clarity are shown below.







Question 2.1:

Standardize feature columns and prepare them for training.

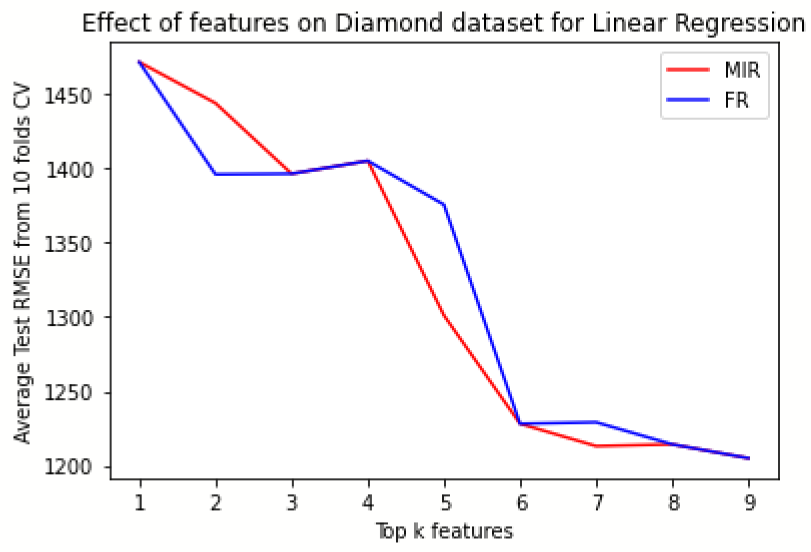
The data set was standardized using `StandardScaler` method from the sklearn.preprocessing library.
The sample of the data set after standardizing is shown below.

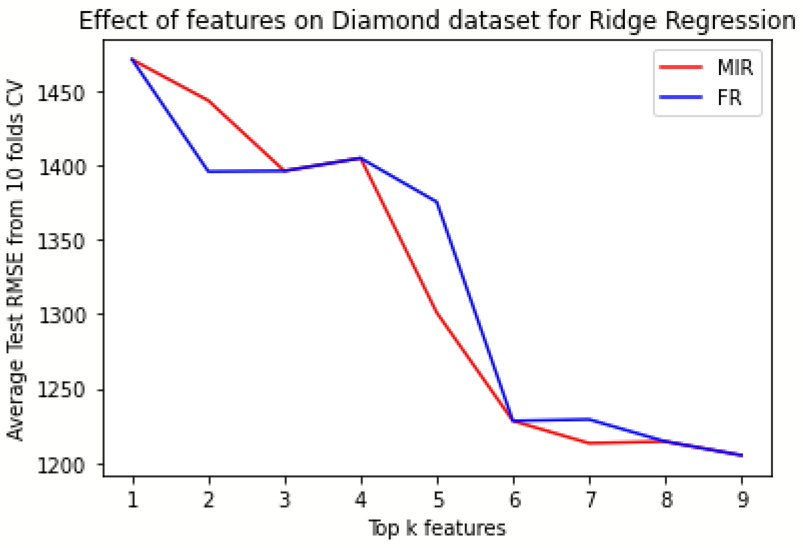
	cut	color	clarity	price	std_price	carat	depth	table	x	y	z	cut_encoded	color_encoded	clarity_encoded
0	Ideal	E	SI2	330	-0.903594	-1.198168	-0.174092	-1.099672	-1.587837	-1.536196	-1.571129	0.981473	0.937163	-1.245215
1	Premium	E	SI1	327	-0.904346	-1.240361	-1.360738	1.585529	-1.641325	-1.658774	-1.741175	0.085889	0.937163	-0.638095
2	Good	E	VS1	328	-0.904095	-1.198168	-3.385019	3.375663	-1.498691	-1.457395	-1.741175	-1.705279	0.937163	0.576145
3	Premium	I	VS2	337	-0.901839	-1.071587	0.454133	0.242928	-1.364971	-1.317305	-1.287720	0.085889	-1.414272	-0.030975
4	Good	J	SI2	338	-0.901588	-1.029394	1.082358	0.242928	-1.240167	-1.212238	-1.117674	-1.705279	-2.002131	-1.245215
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
53935	Ideal	D	SI1	2758	-0.294982	-0.164427	-0.662711	-0.204605	0.016798	0.022304	-0.054888	0.981473	1.525021	-0.638095
53936	Good	D	SI1	2759	-0.294731	-0.164427	0.942753	-1.099672	-0.036690	0.013548	0.100988	-1.705279	1.525021	-0.638095
53937	Very Good	D	SI1	2760	-0.294480	-0.206621	0.733344	1.137995	-0.063434	-0.047741	0.030135	-0.809695	1.525021	-0.638095
53938	Premium	H	SI2	2757	-0.295232	0.130927	-0.523105	0.242928	0.373383	0.337506	0.285204	0.085889	-0.826413	-1.245215
53939	Ideal	D	SI2	2761	-0.294230	-0.101137	0.314528	-1.099672	0.088115	0.118616	0.143499	0.981473	1.525021	-1.245215

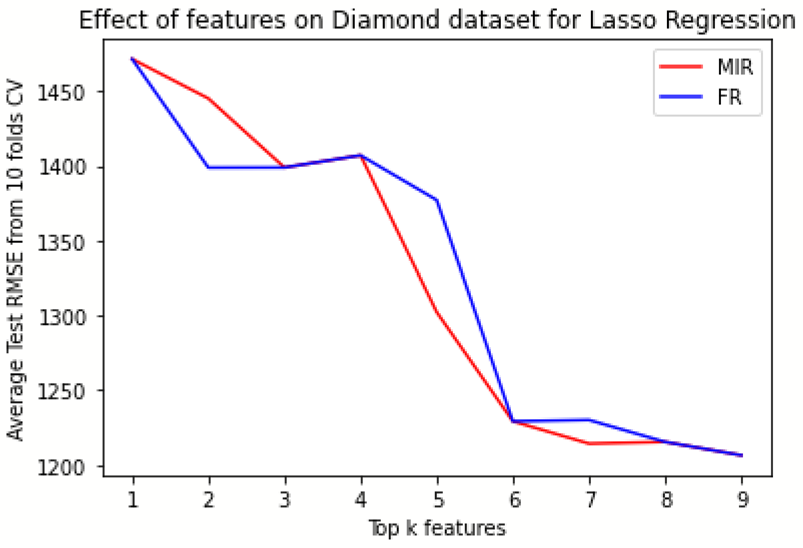
53940 rows × 14 columns

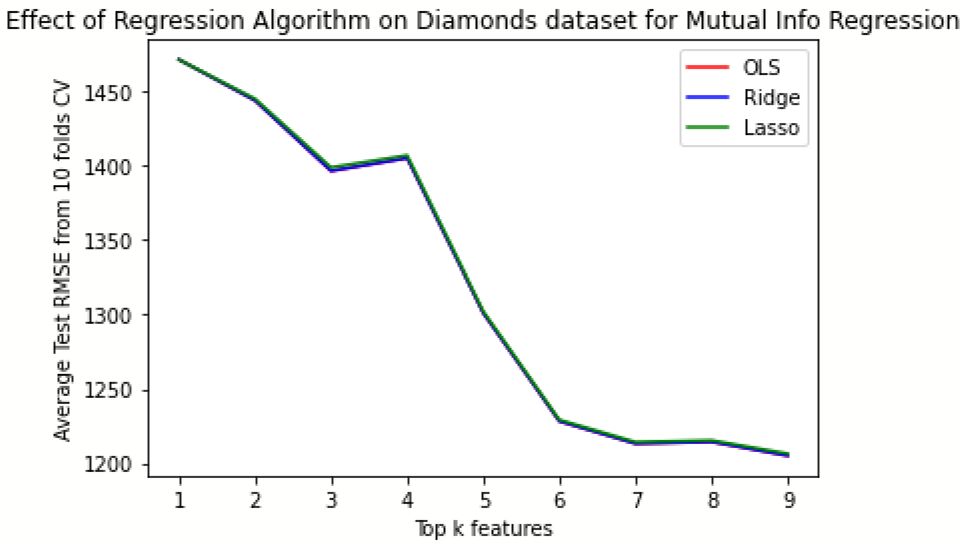
Question 2.2:
You **may** use these functions to select features that yield better regression results (especially in the classical models). Describe how this step qualitatively affects the performance of your models in terms of test RMSE. Is it true for all model types? Also, list two features for either dataset with the lowest MI w.r.t to the target.

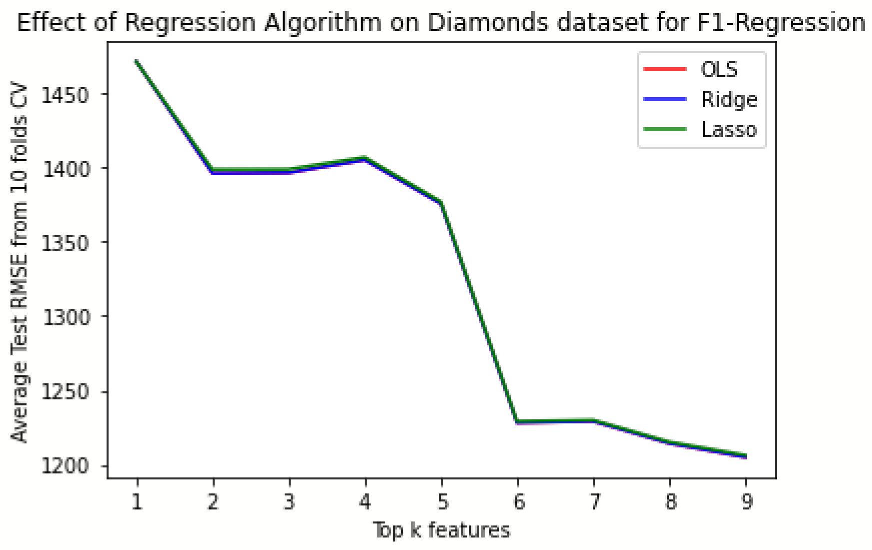
Here we are asked to compare two feature extraction methods, mutual information-based and f score based. The mutual information-based method calculates the dependency between two features, and the f score-based model compares the f scores with different features selected.




Effect of features on Diamond dataset for Ridge Regression
MIR
FR
Average Test RMSE from 10 folds CV
Top k features


Effect of features on Diamond dataset for Lasso Regression
MIR
FR
Average Test RMSE from 10 folds CV
Top k features


Effect of Regression Algorithm on Diamonds dataset for Mutual Info Regression
OLS
Ridge
Lasso
Average Test RMSE from 10 folds CV
Top k features



- Here, it can be observed that if the number of features being considered increases, then the RMSE decreases.
- And there is not much difference in the effect of the regression algorithm that is being chosen. All three algorithms appear to be performing nearly the same.
- The drop in the RMSE is steep until the number of features is 6. Then it decreases at a lower rate.

Question 3:
For random forest model, explain what OOB error and R2 score means given this link.

OOB Error and R2 score:
Random forest is an ensemble of many decision trees, and it uses bootstrapping to generate data which is given as a different input to each decision tree (DT). For every row in the main sample, it is run through each decision tree (DT) that did not include the row in its bootstrap training data. The predictions made by each DT are recorded, and the most frequent prediction for each row is identified. The OOB (out of bag) error is then determined by counting the number of rows incorrectly predicted from the OOB sample. For this OOB error we don't need to set aside a part of the training data we have validation. And R2 score coefficient of determination is used to assess a linear regression model's effectiveness. One can measure the proportion of variation in the output dependent variable that can be predicted from the input independent variable(s). This evaluation checks how well the model can reproduce observed results by comparing the total deviation of the results explained by the model.

OOB error in the given link is the R2 score for the independent variable and the predicted values averaged over each n_output_.

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

a) Ordinary Least Squared: In this scheme the weights are those that fits the given data best by minimizing the error between the predicted values and the true values.
b) Lasso Regression:
Lasso regression is a regularized linear regression where the regularization effect is provided by using the values of the parameters learned. The function that finds the value to be used is the L1 norm or the modulus value of the weights magnitude.
So, the goal of this regularizartion technique is to find a way to minimize the values of the L1 norm of the parameters along with minimizing the values of the error. And the ratio in which L1 regularization term compares to error term is known as regularization strength. So, the final function to be minimized is the summation of the L1 norms of the parameters multiplied by the regularization strength plus the error term.
This Lasso regularization forces the parameters to become 0.
c) Ridge Regression:
Ridge regression is a regularized linear regression where the regularization effect is provided by using the values of the parameters learned. The function that finds the value to be used is the L2 norm of the weights magnitude.
So, the goal of this regularizartion technique is to find a way to minimize the values of the L2 norm of the parameters along with minimizing the values of the error. And the ratio in which L2 regularization term compares to error term is known as regularization strength. So, the final function to be minimized is the summation of the L2 norms of the parameters multiplied by the regularization strength plus the error term.
This ridge regularization forces the parameters values to become closer to 0.

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

RMSE values for each regression type:
Standardized case:

Linear regression:	Ridge Regression:	Lasso Regression:
F1 - `-1206.2805967946167`	F1 - `-1206.2805967946183`	F1 - `-1206.2805967953695`
MI - `-1206.2805967946167`	MI - `-1206.2805967946183`	MI - `-1206.2805967953695`

Non standardized case:

Linear regression:	Ridge Regression:	Lasso Regression:
F1 - -1206.2805967946167	F1 - -1206.2805967946392	F1 - -1206.2805967957875
MI - -1206.2805967946167	MI - -1206.2805967946392	MI - -1206.2805967957875

We done a grid search for 10 cross validations with different values of the regression strength (0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0) for both the ridge and the lasso regression models. And we found out that for both these models we found that the lambda value of 0.0001 gives the lowest RMSE error. And out of all these models the basic linear regression model was the best with lowest RMSE value and very close to the ridge regression. And the top 9 features are used for regression which is the case with the lowest RMSE as found in the previous question.

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

Comparing the values of the RMSE given above, there was a slight improvement in the model after standardization for both the ridge regession and the lasso regression. The values of the RMSE are better after standardization when compared to those before.

Question 4.4:
Some linear regression packages return p-values for different features. What is the meaning of these p-values and how can you infer the most significant features?

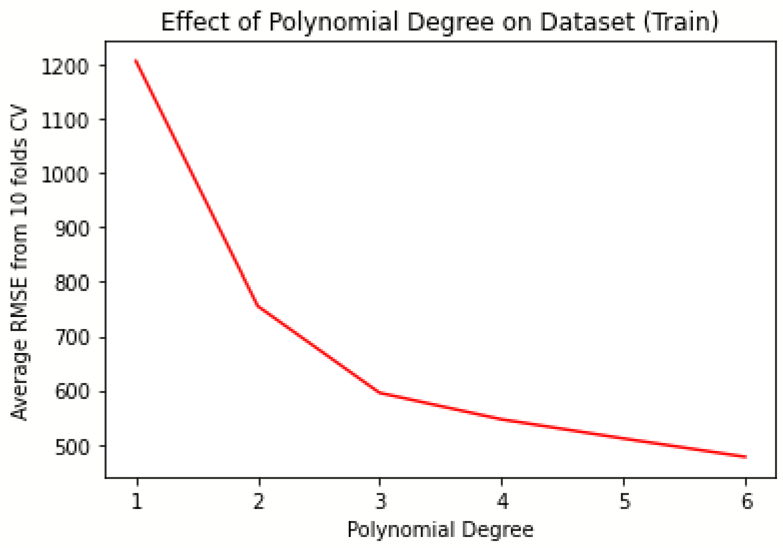
For linear regression models, the p - value of each coefficient gives the likelihood that there is no relationship (or coefficient being 0) between the independent variable and the dependent variable. So, the lower the value of this probability the higher the certainty that there exists a relationship between the independent and dependent variable. But the p - value doesnot give the strength of this relationship.
Some linear regression packages give the p - values for each feature, this p value gives the certainty of the relationship. So, the lower the value the more significant the features are. So, p values are helpful in deciding which features are significant.
To decide on which features are significant we generally set a bar on the p - value below which the features are considered as significant. This is called as the alpha value, generally kept at 0.05. In our case all the features except the ‘z’ are found to be significant for linear regression with p values less than 0.05 in each case.

Question 5.1:
What are the most salient features in polynomial regression? Why?

A 10 times cross validated linear regression model with F- 1 score based model with 9 features and ridge regression was used on polynomial degrees of (1, 2, 3, 4, 5, 6) and lambda values of (0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0).
On this model the top 5 features were found as **['carat', 'x', 'clarity_encoded', 'color_encoded', 'cut_encoded'].**
This is because the polynomial model oforder 1 was the best model and in these model these were the top 5 features that explained the data well.

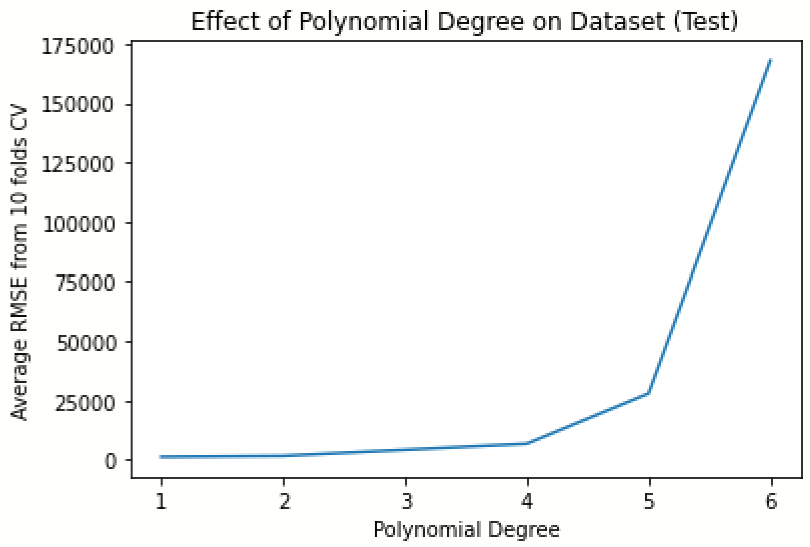
Question 5.2:
What degree of polynomial is best? How did you find the optimal degree? What does a very high-order polynomial imply about the fit on the training data? What about its performance on testing data?

On this data the polynomial with degree 1 was found to be the best. This was based on the RMSE value of the models based on the test data. The lower the RMSE value, the better the model is, and it was found that the model with degre ne has the lowest RMSE.
The trend of the RMSE values with the polynomial degree for the training data is shown below:



It can be observed that as the degree of polynomial is increasing the RMSE on the training data is decreasing. This always happens as the complexity of the model increases the training error decreases.

The trend of the RMSE values with the polynomial degree for the training dat is shown below:



It can be observed that although the training error was decreasing, the test error increased with increasing polynomial degree. This means that there is overfitting occurring with the increasing degree, and the model with degree 1 fits the data best.

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

Using ten times cross-validated grid search to create, fit and validate a neural network by using the following hyperparameters, L2 regularization strength : [0.001,0.1,10], model activation: ['tanh', 'relu'], the number of hidden layers is five different patterns which are obtained by using combination with replacement from a list between [10,100].
So, the best model that was obtained had the following hyperparameters:
Model activation: 'relu,' L2 regularization strength: 10, model hidden layer sizes: (100, 100).
For this model, the training RMSE was -548.3416401412477, whereas the test RMSE was -622.6319133312167.

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

The test RMSE for each best model of each model type is given below.
Best Lasso Regression = -1205.0451378935275
Best Ridge Regression = -1205.0450594357058

Best Linear Regression = -1205.0450508509552
Best Neural Network = -622.6319133312167

The Test RMSE of the best neural network is nearly half of the value of the test RMSE of the regression techniques. This is because regression is a linear technique that can only deal with linear dependencies, whereas neural networks can deal with both linear and non-linear dependencies. So these nonlinear dependencies could have caused the neural networks to perform better than regression.
Apart from these, the number of trainable parameters in neural networks is many more than that of a linear regression algorithm (9 variables in our case). So, these make the model complexity more and thus a good fit for the data.

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

We used the identity function(default one) as the output activation function, as there are no restrictions on the output variable values except that it cannot be zero as it is a price. But we were not encountering any negative values, as it is automatically handled by the loss function to keep the values close to their originals. So, we did not specifically use any output activation functions.

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

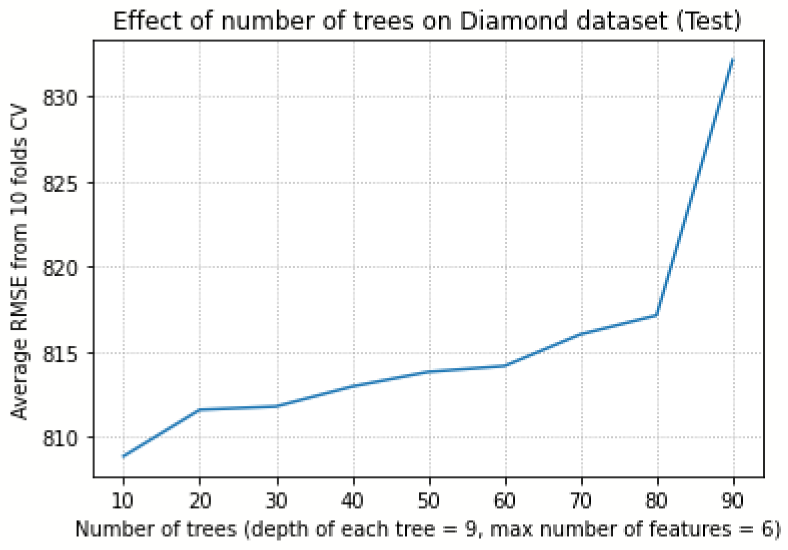
Many risks prevent us from using deep neural networks:

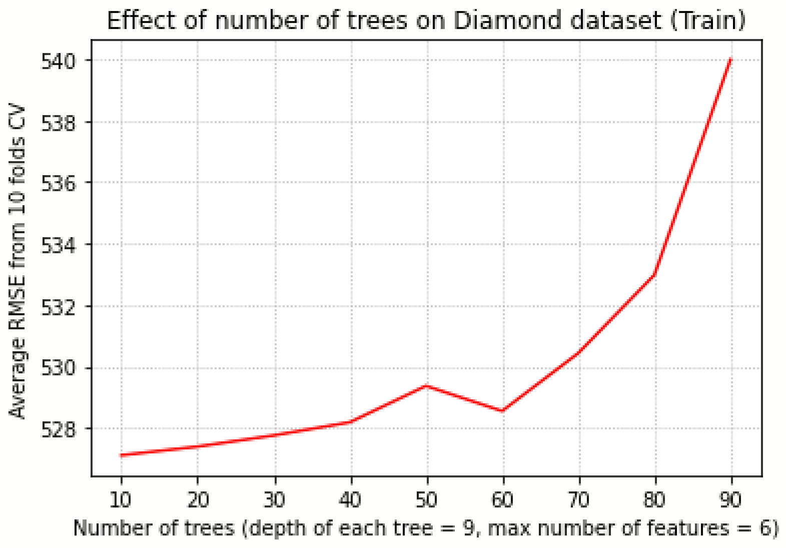
- Vanishing gradients:
  When there are more layers in the network, the value of the product of derivative decreases until at some point the partial derivative of the loss function approaches a value close to zero, and the partial derivative vanishes. We call this the vanishing gradient problem.
- Exploding gradients:
  Exploding gradients are a problem when large error gradients accumulate, resulting in very large updates to neural network model weights during training. Gradients are used during training to update the network weights, but this process typically works best when these updates are small and controlled.
- Overfitting:

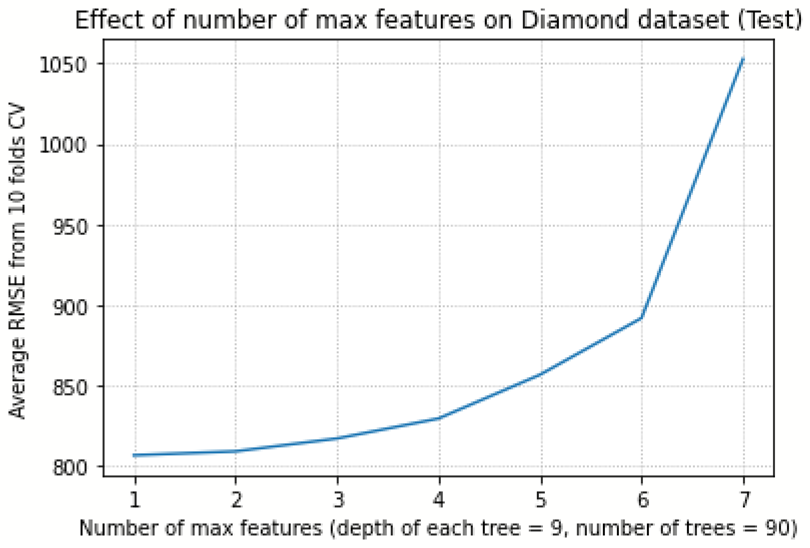
An increase in the number of layers means an increase in the complexity of the model and the number of parameters of the model. Thus the model might become overfit to the data that is being trained on.

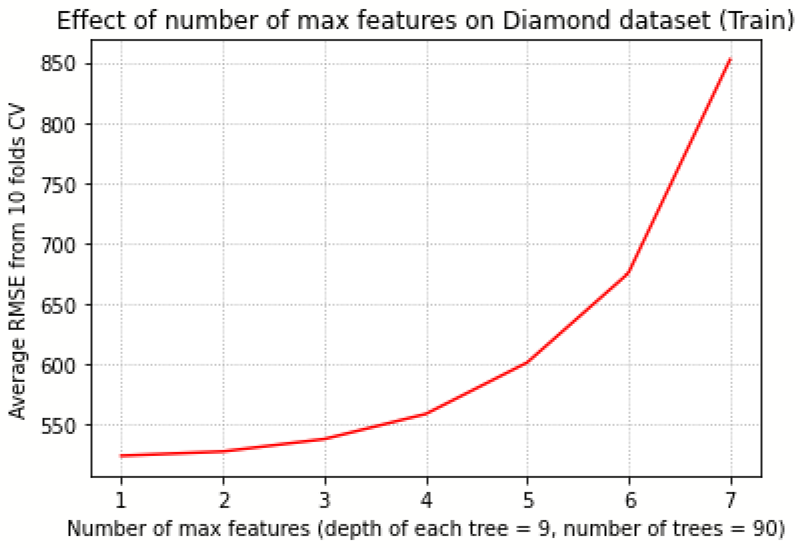
Question 7.1:
Explain how these hyper-parameters affect the overall performance. Describe if and how each hyper-parameter results in a regularization effect during training.

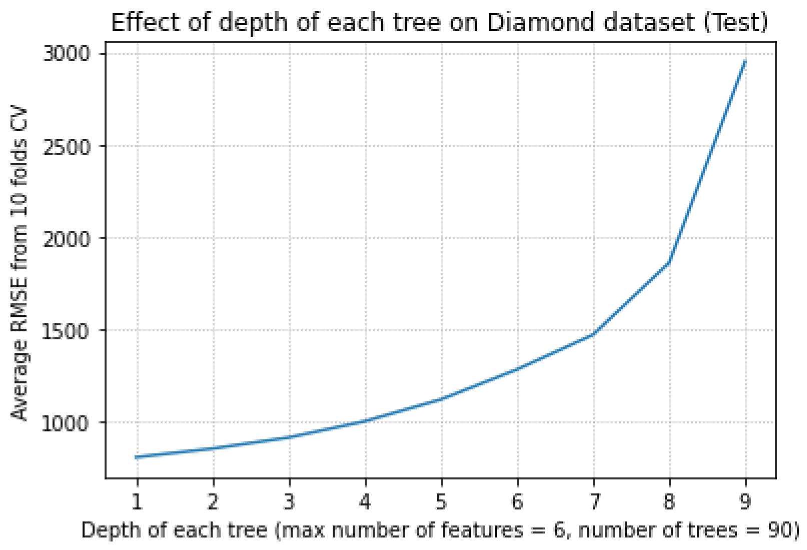
- Increasing max_features generally improves the performance of the model as at each node; now, we have a higher number of options to be considered. However, this is not necessarily true, as this decreases the diversity of individual trees.
- The main advantage of using a large number of simple models, or decision trees, in your random forest model is that predictive performance tends to increase as the number of trees increases. It is important to note that after a certain point, you start to hit a point of diminishing returns where the scale of the performance gains you see from adding more trees gets smaller and smaller. Random forest models made with many trees also encode more complexity than random forest models made with a small number of trees.
- Random forest Decision trees that have a large max depth are more likely to overfit to the data they were trained on. Shallow trees with small max depth models that have many trees may overfit the dataset they were trained on.

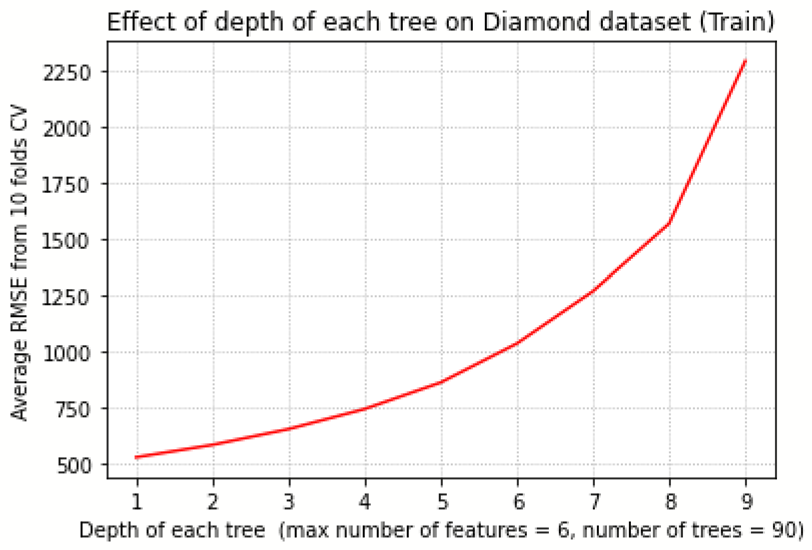



Effect of number of trees on Diamond dataset (Train)
Average RMSE from 10 folds CV
540
538
536
534
532
530
528
10
20
30
40
50
60
70
80
90
Number of trees (depth of each tree = 9, max number of features = 6)


Effect of number of max features on Diamond dataset (Test)
Average RMSE from 10 folds CV
1050
1000
950
900
850
800
1
2
3
4
5
6
7
Number of max features (depth of each tree = 9, number of trees = 90)


Effect of number of max features on Diamond dataset (Train)
Average RMSE from 10 folds CV
850
800
750
700
650
600
550
1
2
3
4
5
6
7
Number of max features (depth of each tree = 9, number of trees = 90)



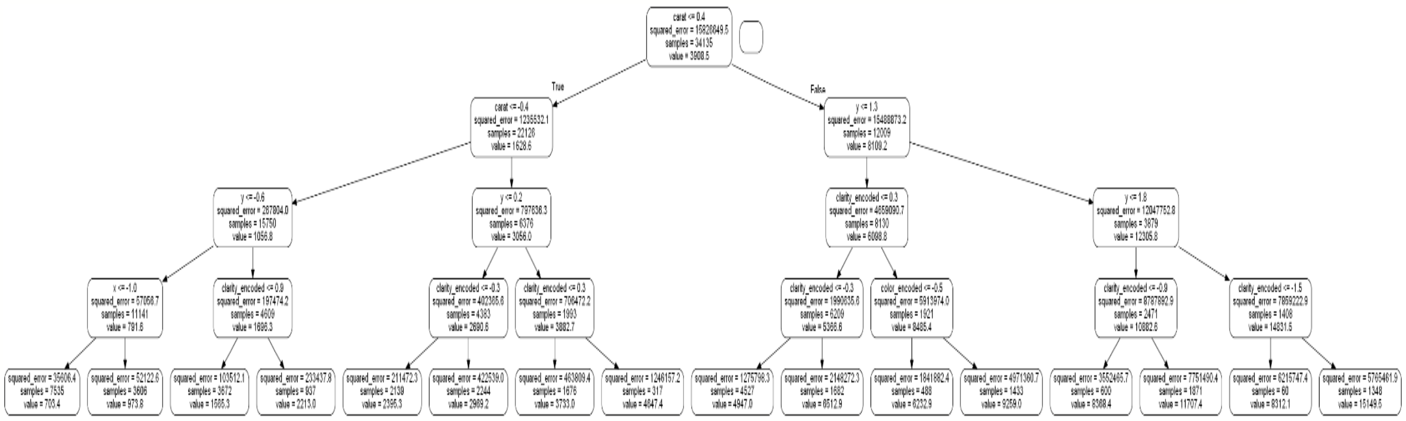


The graph above shows us the effect of each parameter in the random tree on the performance of the random forest model.

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

Random Forest is considered a non-linear model due to its lack of assumptions regarding a linear relationship between predictor and target variables. The non-linear nature of Random Forest, as well as a single decision tree, is evident in the box-like decision boundary that arises from model training. This means that a decision tree can use the same feature in various splitting criteria across different levels. Consequently, the model can learn intricate relationships within the data structure without the limitations and assumptions that a linear model would impose.

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



A tree from the random forest with a maximum depth of 4 is shown above. And the feature for branching at the root node is a carat variable with values for the threshold is 0.4. And with this, we can say that the carat is the most critical variable. We also found that the carat is one of the crucial variables from the part 3.3.1. Then the important features are y and clarity_encoded.

Question 7.4:
Measure “Out-of-Bag Error” (OOB). Explain what OOB error and R2 score means.

The out-of-bag error for the random forest model `0.9784945892811334.` And the terms OOB error and R2 score are already explained in question 3.

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

Light GBM is one of the types of boosted trees. The important hyperparameters present in the lightGBM model are given below:

- Boosting Type:
  This hyperparameter selects which kind of algorithm to be used. There are four choices, gbdt: traditional Gradient Boosting Decision Tree, rf: random forest, dart: Dropouts meet Multiple Additive Regression Trees, goss: Gradient-based One-Side Sampling
  - gbdt: traditional Gradient Boosting Decision Tree
    In the gbdt method we have a lot of decision trees(weak learners). Those trees are built sequentially:
    First tree learns how to fit to the target variable.
    Second tree learns how to fit to the residual (difference) between the predictions of the first tree and the ground truth.
    The third tree learns how to fit the residuals of the second tree, and so on.
    All those trees are trained by propagating the gradients of errors throughout the system.
    The main drawback of gbdt is that finding the best split points in each tree node is time-consuming and memory-consuming operation other boosting methods try to tackle that problem.
  - dart: Dropouts meet Multiple Additive Regression Trees
    One issue with GBDT is over-specialization, where the trees added in later iterations only impact a small number of instances and have little effect on the remaining instances. However, incorporating dropout makes it harder for the later trees to specialize on those few samples and can lead to improved performance.
  - rf: random forest
    This is a general random forest-based model, which is an ensemble of many trees.
  - goss: Gradient-based One-Side Sampling
    Goss proposes a sampling method based on the gradient to avoid searching through the entire search space. When the gradient is small for a data instance, it indicates that the data is well-trained. Conversely, a large gradient suggests that the data should be retrained. Therefore, there are two groups of data instances: those with small and large gradients. With One-Side Sampling, goss retains all data with a large gradient and randomly samples from the data with a small gradient. This approach reduces the search space, allowing goss to converge more quickly.
- Task:
  What task is to be performed on the data, that is to train or predict
- Application:

What is the main application of this model? Is it classification or regression? If it is classification, is it binary or multi-class?

- Learning Rate:
  This process assesses the contribution of individual trees to the final outcome. GBM operates by commencing with an initial estimate that is modified using each tree's output. The extent of the modification in the estimates is determined by the learning parameter.
- Number of booting rounds or iterations:
  The number of iterations to be performed.
- Early stopping round:
  Model will stop training if one metric of one validation data doesn’t improve in last early_stopping_round rounds. This will reduce excessive iterations.
- Number of leaves:
  The number of leaves in a full tree.
- Max depth:
  This parameter defines the maximum depth of the tree and is utilized to address model overfitting.
- Regression lambda:
  The strength of the regularization that is being used.
- Regression alpha:
  The strength of the regularization that is being used.
- Bagging fraction:
  The fraction of the training data that is to be bagged during every bagging.
- Min data in leaf:
  Minimal number of data in one leaf. Can be used to deal with over-fitting
- Bagging frequency:
  Perform bagging at every k specified iterations.
- N_estimators:
  Number of boosted trees to fit.

The search space that will be used for the analysis is given below:

- 'num_leaves': np.arange(1, 3000, 100)
- 'min_data_in_leaf' : np.arange(1, 1000, 100)
- 'max_depth': np.arange(1, 100, 10)
- 'n_estimators': np.arange(10, 4000, 100)
- 'reg_alpha': [10.0**x for x in np.arange(-3, 3)]
- 'reg_lambda': [10.0**x for x in np.arange(-3, 3)]

- 'subsample': np.arange(0.1, 1, 0.1)
- ‘subsample_freq': np.arange(0, 100, 10)
- 'min_split_gain': [10.0**x for x in np.arange(-3, 0)]
- 'bagging_fraction' : [0.4, 0.6, 0.8]
- "bagging_freq" : [1, 2]
- "bagging_seed" : [42]
- 'boosting_type': ['gbdt', 'dart', 'rf']

Question 8.2:
Apply Bayesian optimization using skopt.BayesSearchCV from scikit-optmize to find the ideal hyperparameter combination in your search space. Report the best hyperparameter set found and the corresponding RMSE.

After applying bayesian optimization using skopt.BayesSearchCV from scikit-optmize with 10 times cross-validation the ideal hyperparameter combination was found as given below:

- bagging_fraction', 0.6
- 'bagging_freq', 2
- ‘bagging_seed', 42
- ’boosting_type', 'dart'
- 'max_depth', 71
- 'min_data_in_leaf', 301
- 'min_split_gain', 0.01
- 'n_estimators', 2110
- 'num_leaves', 2301
- 'reg_alpha', 1.0
- 'reg_lambda', 1.0
- 'subsample', 0.6
- 'subsample_freq', 0

And the values of the train, and test RMSE corresponding to this ideal hyperparameter combination are -481.436866, -587.279550020244.

The top 10 models (hyperparameters that are tuned) and their corresponding train and test RMSE obtained from the BayesSearchCV method with 10 fold cross-validation are given below in the table below:

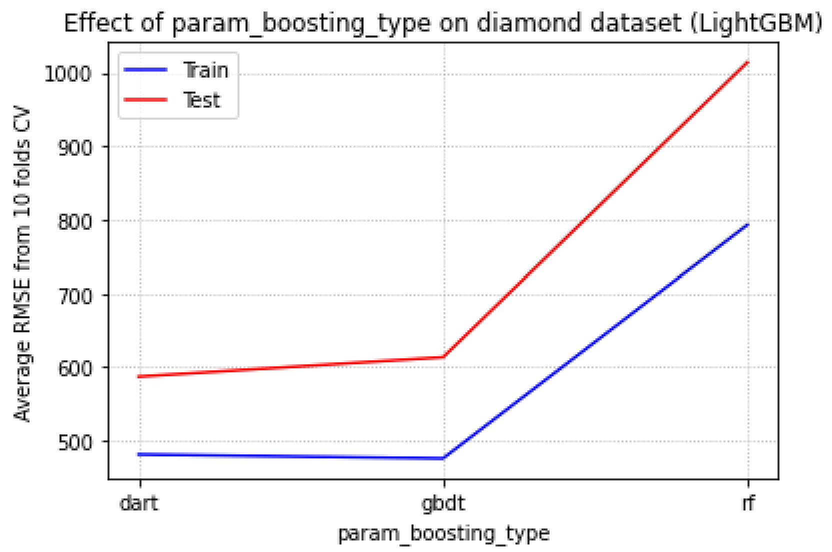
test_score	train_score	boost_type	num_leaves	max_depth	n_estimators	reg_alpha
-587.27955	-481.436866	dart	2301	71	2110	1
-602.259912	-393.834249	gbdt	1701	31	1410	0.01
-603.629489	-499.249012	gbdt	2601	51	610	10
-609.218378	-476.747092	gbdt	1601	21	2910	100
-609.592765	-475.639951	gbdt	901	61	2910	1
-643.383115	-534.462088	gbdt	1001	81	2510	1
-798.558392	-442.066426	rf	1101	31	110	10
-893.464982	-723.815155	rf	2601	51	3410	0.1
-915.197533	-741.230163	rf	1801	81	1410	100
-1450.4157	-1265.96093	rf	2101	51	1610	1

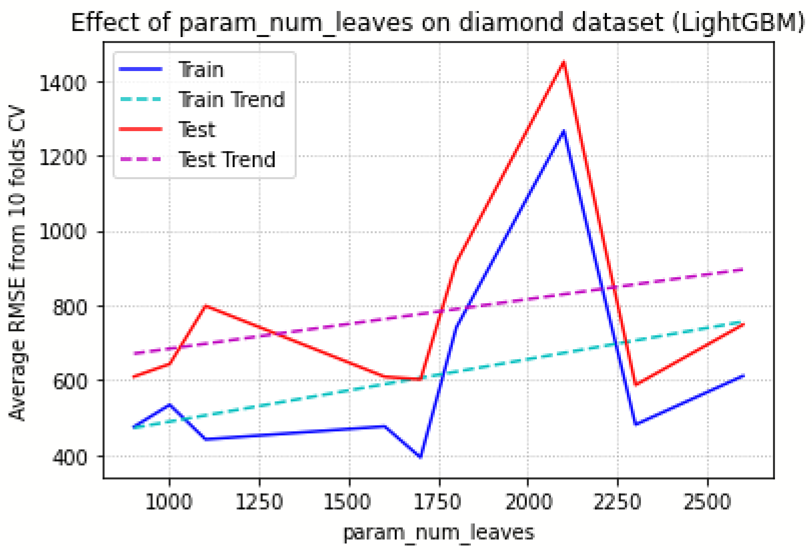
reg_lambda	subsample	subsample_freq	bagging_fraction	bagging_freq	min_data_in_leaf	min_split_gain
1	0.6	0	0.6	2	301	0.01
10	0.5	10	0.8	2	201	0.001
10	0.1	20	0.4	2	201	0.01
10	0.4	20	0.6	2	401	0.001
1	0.2	20	0.6	2	401	0.01
1	0.8	20	0.8	1	601	0.01

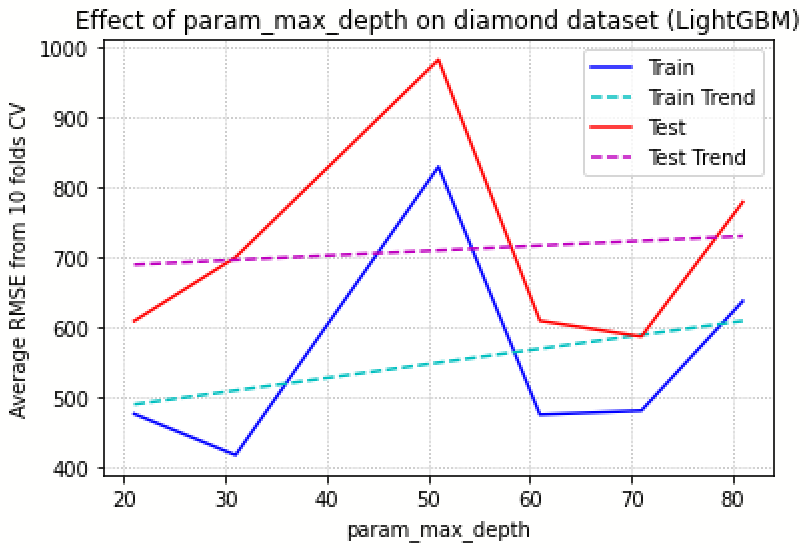
0.1	0.6	60	0.6	2	1	0.01
0.1	0.8	50	0.8	1	101	0.1
1	0.9	70	0.8	2	101	0.001
100	0.6	60	0.6	2	501	0.01

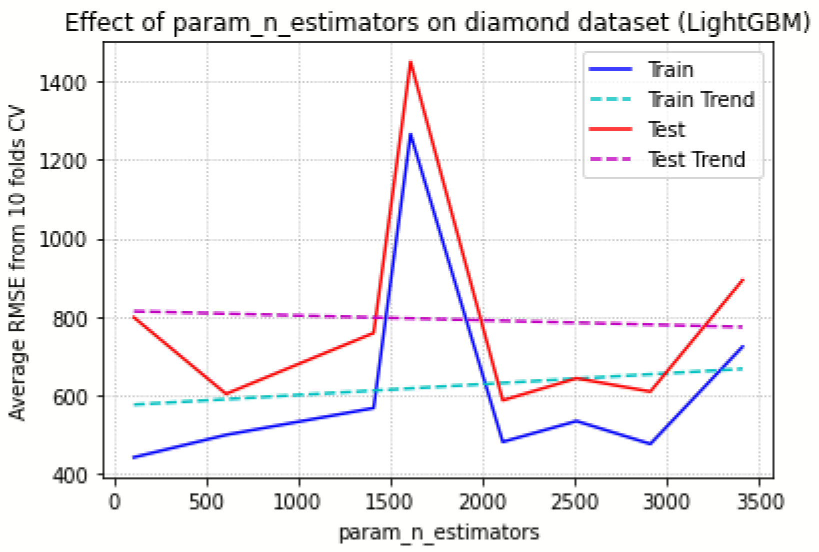
Question 8.3:
Qualitatively interpret the effect of the hyperparameters using the Bayesian optimization results: Which of them helps with performance? Which helps with regularization (shrinks the generalization gap)? Which affects the fitting efficiency?

The graphs showing the variation of the test and train mse vs each of these parameter values are given below:

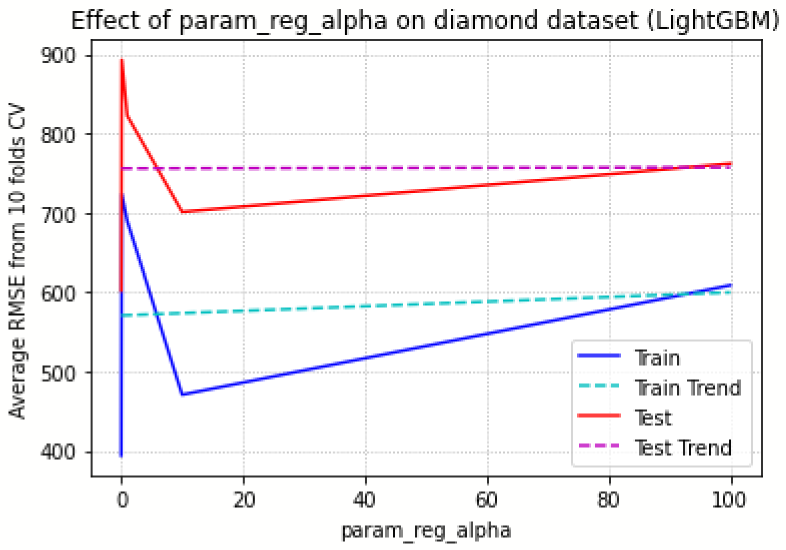



Effect of param_num_leaves on diamond dataset (LightGBM)
Train
Train Trend
Test
Test Trend
Average RMSE from 10 folds CV
param_num_leaves
400
600
800
1000
1200
1400
1000
1250
1500
1750
2000
2250
2500

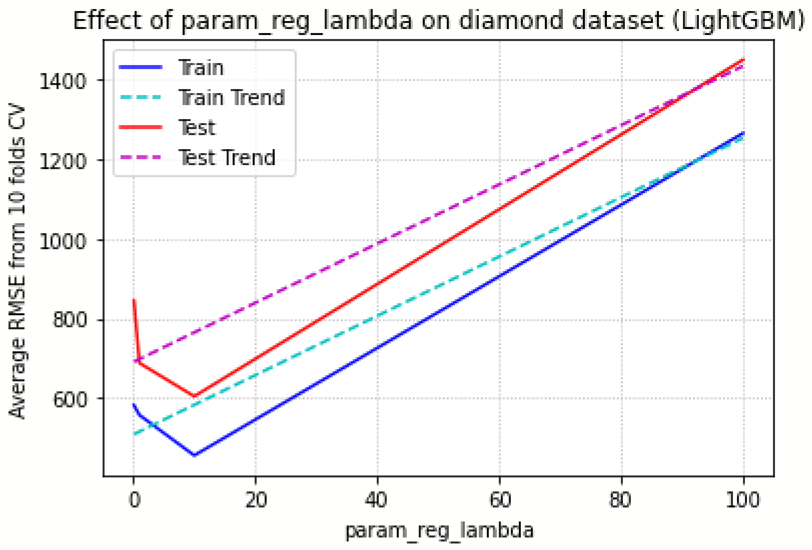

Effect of param_max_depth on diamond dataset (LightGBM)
Train
Train Trend
Test
Test Trend
Average RMSE from 10 folds CV
param_max_depth
400
500
600
700
800
900
1000
20
30
40
50
60
70
80

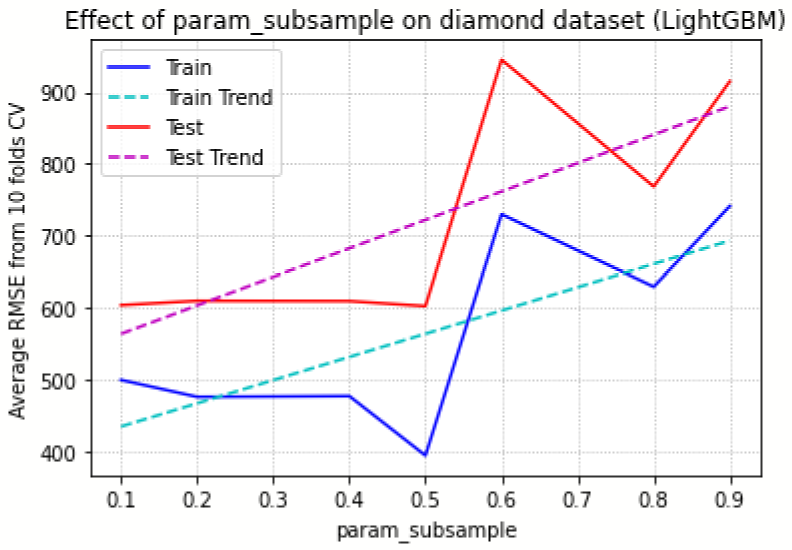


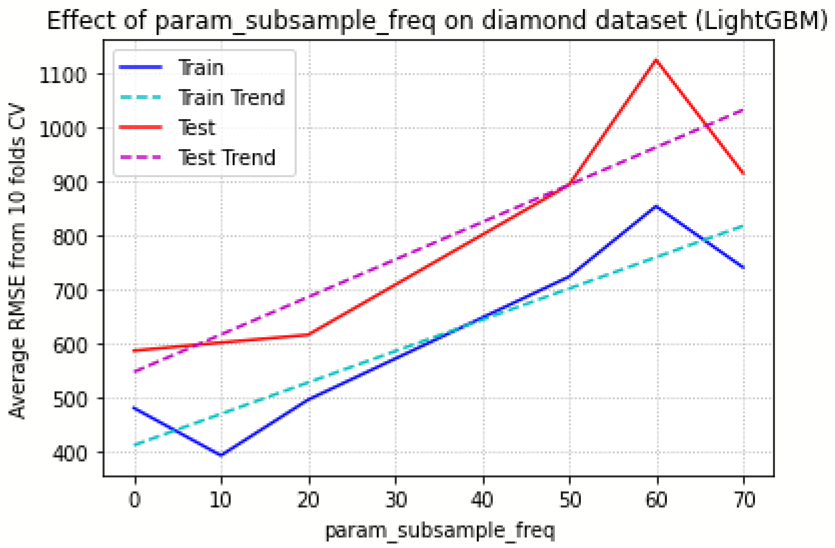
Effect of param_n_estimators on diamond dataset (LightGBM)
Average RMSE from 10 folds CV
param_n_estimators
Train
Train Trend
Test
Test Trend



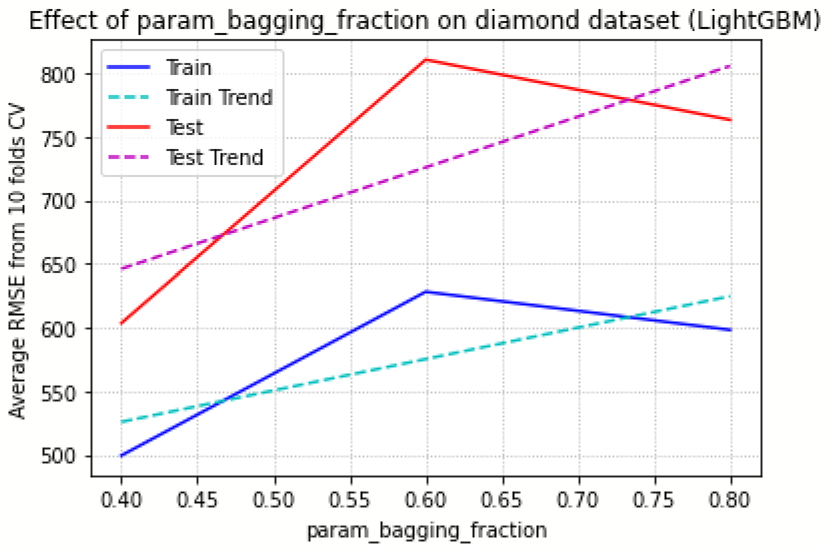
Effect of param_reg_alpha on diamond dataset (LightGBM)
Average RMSE from 10 folds CV
param_reg_alpha
Train
Train Trend
Test
Test Trend


Effect of param_reg_lambda on diamond dataset (LightGBM)
Train
Train Trend
Test
Test Trend
Average RMSE from 10 folds CV
param_reg_lambda
1400
1200
1000
800
600
0
20
40
60
80
100

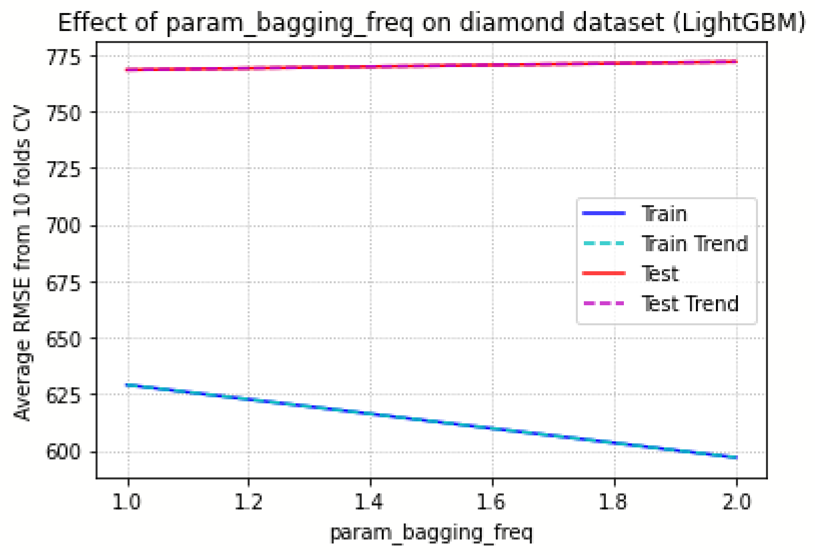

Effect of param_subsample on diamond dataset (LightGBM)
Train
Train Trend
Test
Test Trend
Average RMSE from 10 folds CV
param_subsample
900
800
700
600
500
400
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

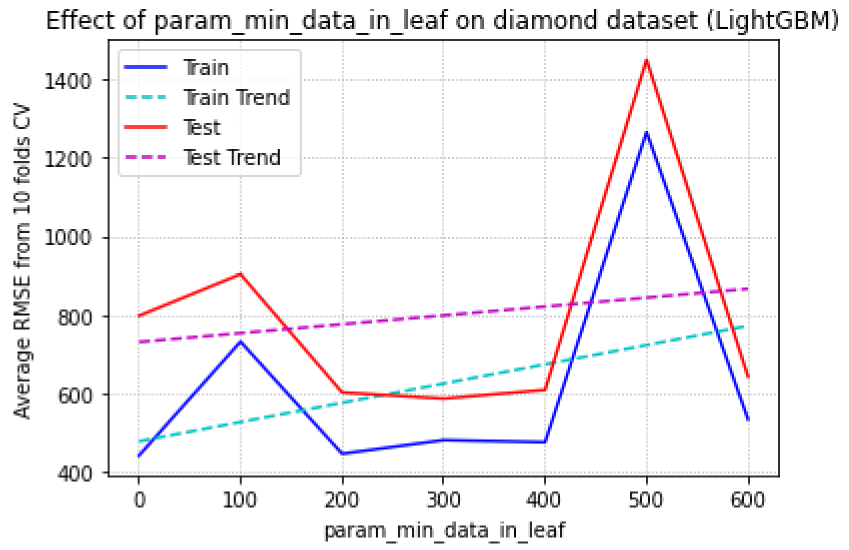


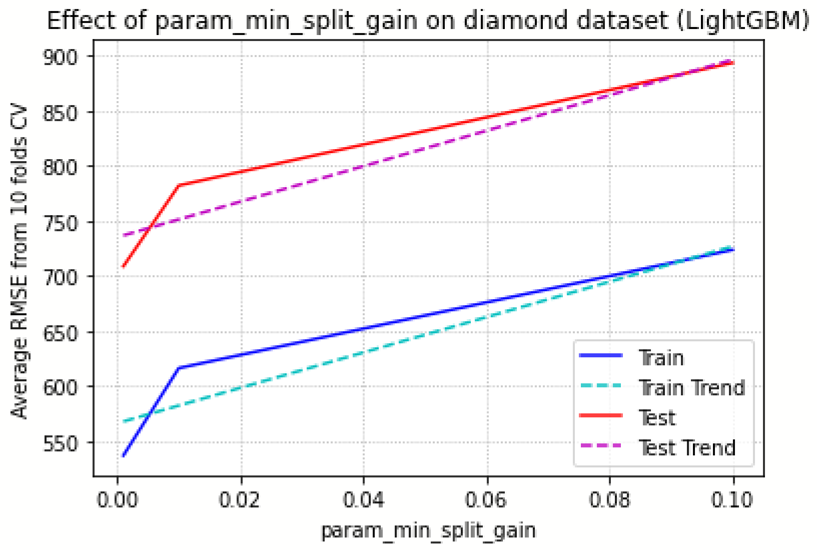
Effect of param_subsample_freq on diamond dataset (LightGBM)
Train
Train Trend
Test
Test Trend
Average RMSE from 10 folds CV
param_subsample_freq



Effect of param_bagging_fraction on diamond dataset (LightGBM)
Train
Train Trend
Test
Test Trend
Average RMSE from 10 folds CV
param_bagging_fraction


Effect of param_bagging_freq on diamond dataset (LightGBM)
Average RMSE from 10 folds CV
param_bagging_freq
Train
Train Trend
Test
Test Trend
775
750
725
700
675
650
625
600
1.0
1.2
1.4
1.6
1.8
2.0


Effect of param_min_data_in_leaf on diamond dataset (LightGBM)
Average RMSE from 10 folds CV
param_min_data_in_leaf
Train
Train Trend
Test
Test Trend
1400
1200
1000
800
600
400
0
100
200
300
400
500
600



- Boosting Type:
  It can be seen that the gradient boosting algorithms dart and gdbt perform better than the normal random forest, and as expected the test value of the RMSE is lower for the dart when compared to the gdbt because of drop outs in dart that generalize to the data well. Where are the train RMSE is nearly same for both the cases.
- Number of leaves:
  It can be seen that for the train and test RMSE decreases until a value of number of leaves is reached and then it again increases then decrease. This happens in both the train and test showing that it is not due to overfitting.
- Max depth:
  There is a weird trend in the train and test RMSE values of the model with increasing max depth. It increases in the beginning then decreases and then increases. So, there is no explainable trend.
- Number of estimators:
  The effect of the number of estimators on the train and test RMSE is not pretty much high. There was nearly a same value of the RMSE's except for a single value in between.
- L1 regularization term: (alpha)
  By examining the graphs, it is apparent that the train and test RMSE demonstrate an improvement at first with moderate values of Lasso regularization. However, the performance deteriorates when the regularization becomes more severe. When the regularization value is small, there is no regularization, which can lead to overfitting if other hyperparameters are large. On the other hand, larger values of regularization enhance generalization but may adversely affect the model's performance.

- L2 regularization term: (Lambda)
  We observe that both train and test RMSE initially improves with finite values of Tikhonov regularization, but performance drops when aggressive regularization is performed.
- Subsample:
  The test and train RMSE values increase with an increase in the value of the subsample. It was nearly constant until a value of subsample of 0.5 and then it started increasing.
- Subsample frequency:
  There is a clear linear relation between the values of the RMSE and the value of the subsample frequency. So, if subsampling is done very often there is a decrease in the value of the RMSE then subsampling not being done very often.
- Bagging fraction:
  There is a slight positive relation between the value of the RMSE and the value of the bagging fraction. But this doesnot make intuitive sense as the fraction of data that is used if increased will increase the data that is present for the model to learn. So, the RMSE should be decreasing.
- Bagging frequency:
  There is a negative correlation between the train RMSE and the value of the bagging frequency. But there is a very very slight positive correlation between the values of the test RMSE and the value of the bagging frequency.This is anticipated since bagging functions as a regularization technique. When bagging is performed more frequently, i.e., when samples are dropped, certain subsets of the training data are dropped more frequently as well. This may lead to crucial training samples being disregarded during training.
- Min data in leaf:
  There is not a clear relation between the values of the min data in a leaf vs the model performance.
- Min split gain:
  Augmenting this ratio acts as a regularization mechanism by directing the ensemble to disregard marginal improvements in the training performance that do not significantly affect the model's generalizability. It is worth noting, however, that the graph provides a completely opposite picture since the Bayesian agent did not cover the entire range of ratios. If aggressive bagging is applied excessively, the model's performance is likely to suffer due to the regularization effects.

For higher performance:

1. Grow shallower trees, which can be done by tweaking max_depth, no of leaves, min split gain
2. Grow less trees, which can be done by tweaking number of iterations

For good regularization tweaking num of leaves, max_depth, n_estimators, subsampling ratio can be useful. For good fitting using the dart model helps.

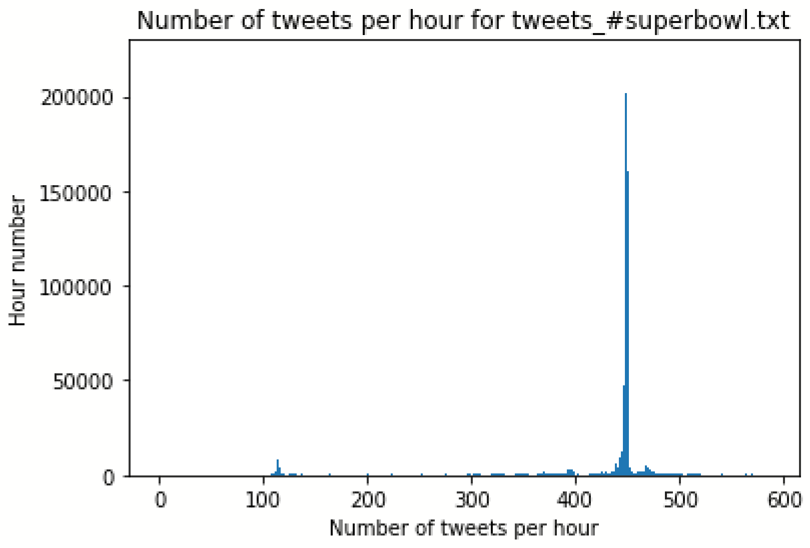
Question 9.1:
Report the following statistics for each hashtag, i.e. each file has:
• Average number of tweets per hour
• Average number of followers of users posting the tweets per tweet (to make it simple, we average over the number of tweets; if a users posted twice, we count the user and the user's followers twice as well)
• Average number of retweets per tweet

The required statistics are given in the table below:

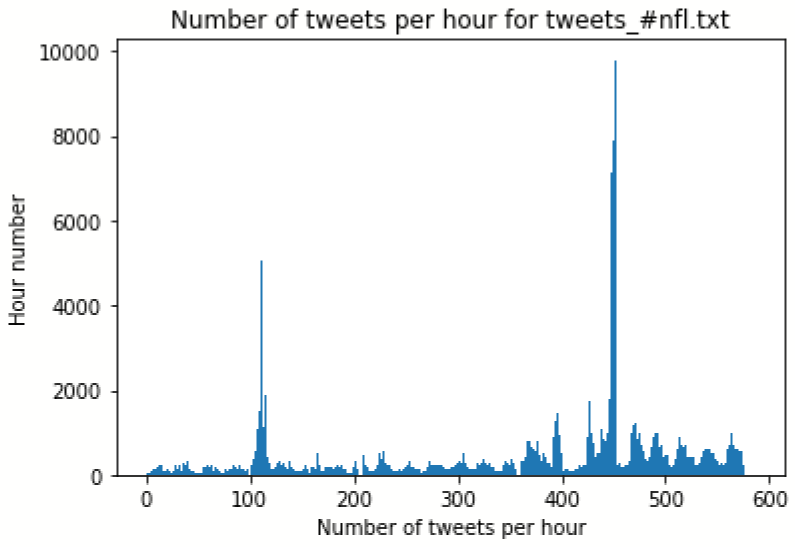
#	Avg. no. of tweets per hour	Avg. no. of followers of users posting	Avg. no. of retweets per tweet
gohawks	292.487850621736	2217.92373552819	2.01320939913198
gopatriots	40.9546980060619	1427.25260516354	1.40819191016970
nfl	397.021390181984	4662.37544523693	1.53446026555432
patriots	750.89426460689	3280.46356165502	1.78528712884769
sb49	1276.85705986804	10374.160292019	2.52713444111402
superbowl	2072.11840170408	8814.96799424623	2.39118958192077

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

The plot showing the "number of tweets in han our" over time for #SuperBowl is shown below:



The plot showing the “number of tweets in han our” over time for #nfl is shown below:



For both these plots the x-axis represents the hour number from the starting time of the tweets in the data. And the y-axis represents the number of tweets in that specific hour. And it can be observed that for the super bowl, there is only one peak, which might have happened during the super bowl game time. But in the NFL hashtag graph, you can see two of those peaks and small peaks in between showing for the individual matches present in that NFL tournament, where the larger one is the super bowl.

Question 10:

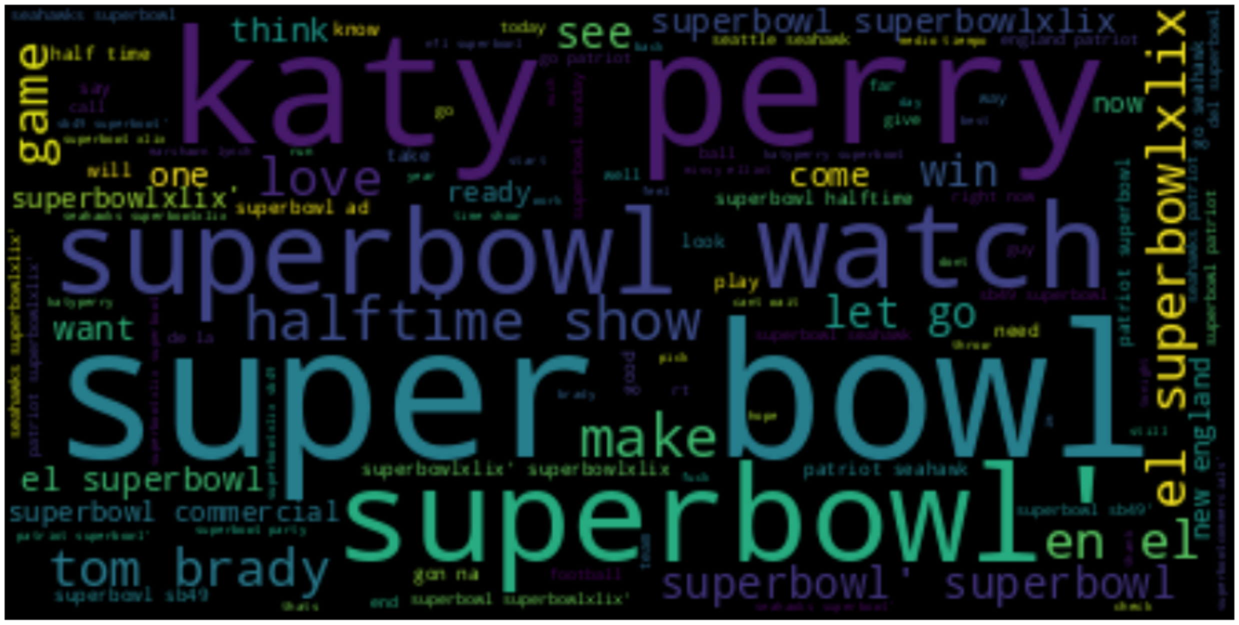
Describe your task.

Title:
Predict the winning team in the Superbowl match based on the tweet text and the time of the tweet.
Project Details:
We are going to do three parts of this project. The first part is exploratory data analysis. Then the second part uses the nfl_data_py package to introduce the time domain into the data and to know the real-time scores when the tweets were being posted. Then the third part is building a model and tuning the hyperparameters such that the model performs well and the best model is made.

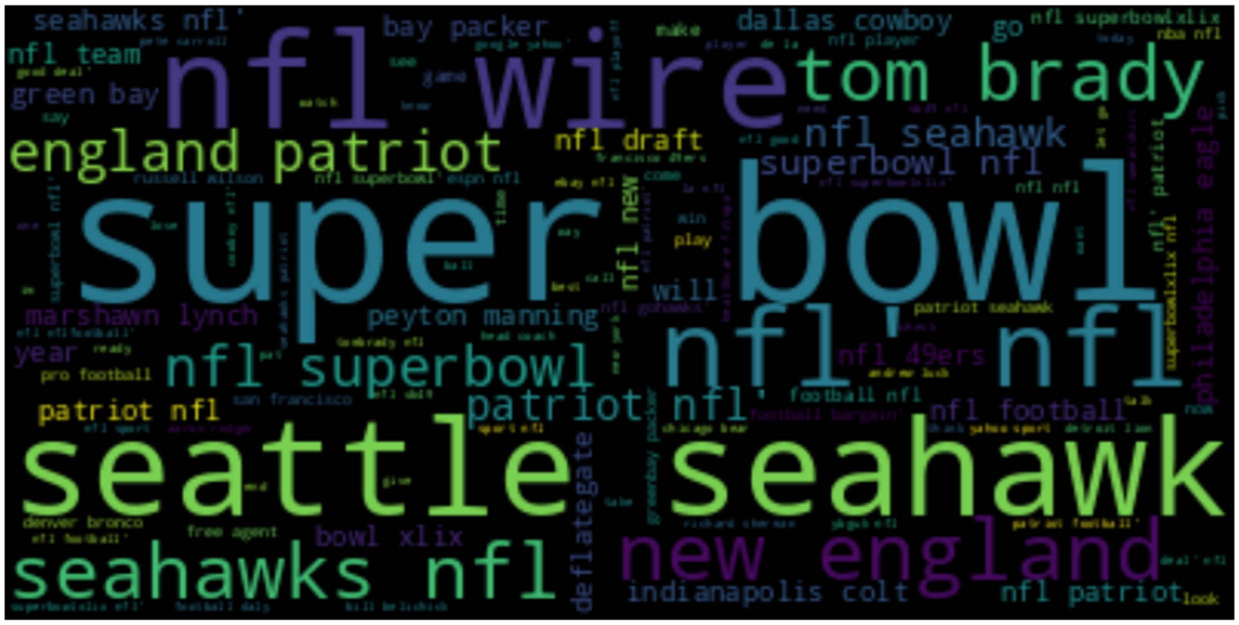
Part :1
Exploratory Data Analysis.
Explore the data and any metadata (you can even incorporate additional datasets if you choose).

Word Cloud:
The tweets data about each of the each of the six hashtag files was used to read the tweet text, tweet time, follower count, and retweet count into the memory. Then the tweet text was cleaned for each of the tweet for all the six hashtags, which were then further lemmatized.
Then these lemmatized tweets were fed into a word cloud model to look at the famous words that are being present in the tweets. The results of these analysis for each of the hash tags are given below:

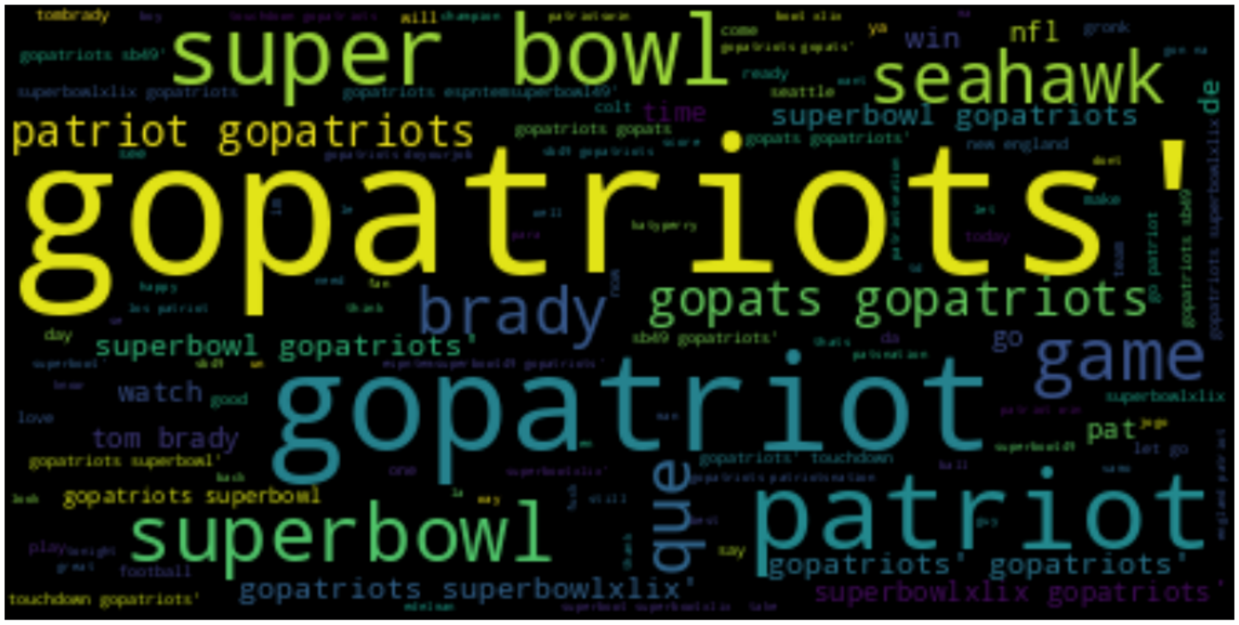
#SuperBowl:



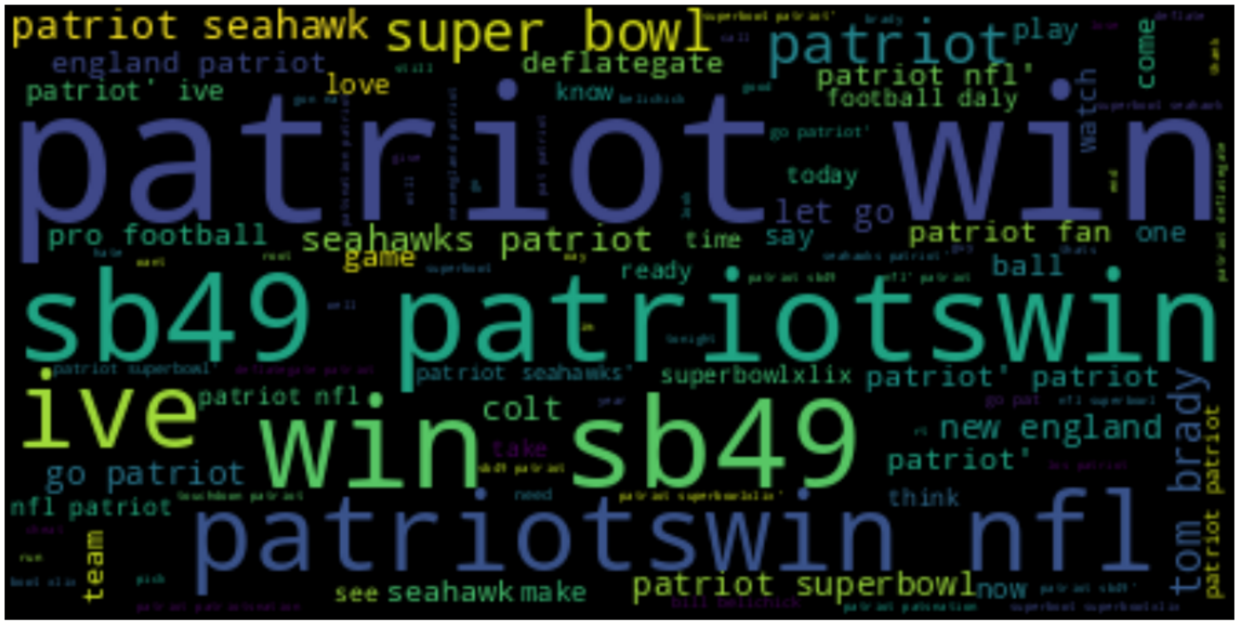
#nfl:



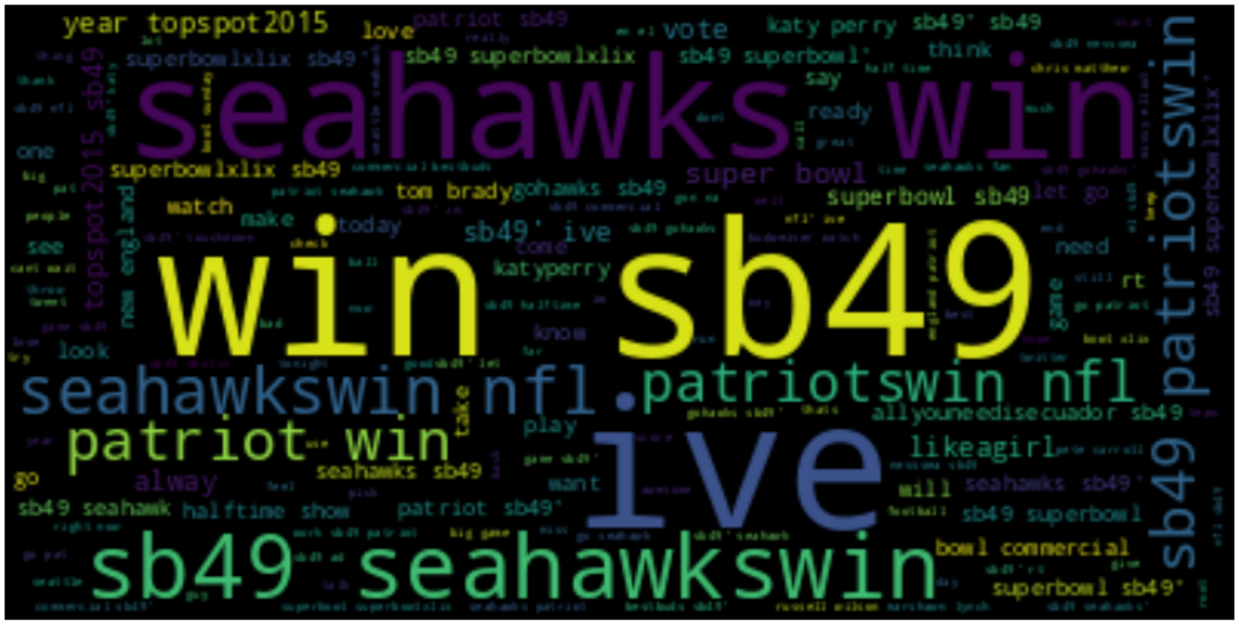
#gopatriots:



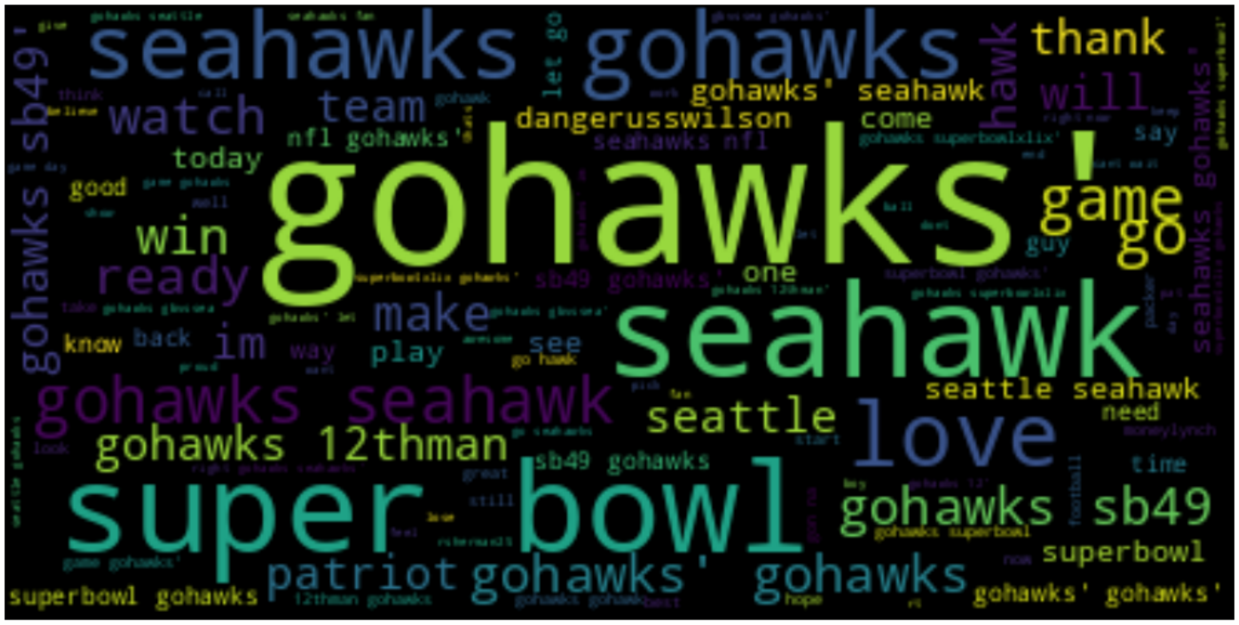
#patriots:



#sb49:



#gohawks:



It can be observed that in the Super Bowl hashtag, the singer Katy Perry was mentioned more times than any other player and many more times than even nfl word. So, this shows that if player-wise matching is to be made then careful observation of the words has to be made.
And It can also be observed that the player Tom Brady is mentioned more than any other player in both the nfl and other hashtags. Then halftime is another important word that is also present in most of the hashtags.

One important observation that can also be made is that in the nfl tweets, the team names are also preset and repeated the most.

Part: 2
Feature Engineering
Describe the feature engineering process. Implement it with reason: Why are you extracting features this way - why not in any other way?

After performing this exploratory data analysis, the data about tweet text, time, number of followers, and retweet count for each tweet of each hashtag was made into a single data frame. So, this data frame contains the data from all the files irrespective of the hashtag it has. Now, this is done so that our model should not be biased based on data from a hashtag like gopatriots might have data biasing patriots. So, this gives us two benefits.
Then nfl_data_py module was imported, which contained the data of the nfl, these contain a huge amount of data about the teams, the players, the matches, etc.

Exploring the nfl_data_py data:
`nfl.import_pbp_data()`
This command will import the play-by-play data which is a transcript of all tracked events in a game. So, this contains the data about all the events that have occurred in nfl. The 2014 nfl module was imported, some of whose columns are given below:
play_idgame_id, old_game_id, home_team, away_team, season_type, week, posteam, posteam_type, defteam, side_of_field, yardline_100, game_date, quarter_seconds_remaining, half_seconds_remaining, game_seconds_remaining, game_half, quarter_end, drive, sp, qtr ,down, goal_to_go, time, yrdln, etc.,

`nfl.import_rosters()`
This command will give the different players details like what team they belonged to, what is their jersey number, how many points did they score, etc, that played in a given year in the nfl. So, this contains the details about different players.

`nfl.import_team_desc()`
This command gives the details about each nfl team like what their short form is what state they belong to what is their logo etc.,

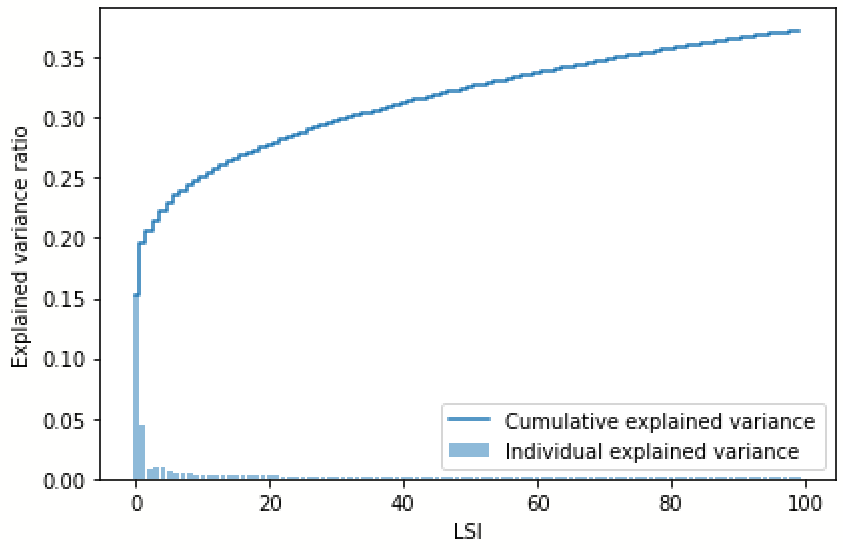
Then from the imported play-by-play data for the year 2014, the match day of the super bowl was chosen, and the data of the super bowl was filtered based on this date. And the following columns was chosen from the entire data frame.

Game_date, play_id, game_id, home_team, away_team, time_game, total_home_score, total_away_score, posteam, defteam, posteam_score, defteam_score, passer_player_id, passer_player_name, receiver_player_name, receiver_player_id, rusher_player_name, rusher_player_id, start_time, time_of_day, play_clock, play_type_nfl
These features were selected because, this features contained critical data about the match situation like what the score of eachteam was, what was the time, what was the player name for each activity if something important like touch down was going on.

Then the time for the PBP data was converted into the PST time, then again the tweet data time was also converted into the PST time. Then both of these data frames are merged using pd.merge_asof function which has the option to merge data frames based on the time column with the option to merge those times that are nearby. So, this was done to merge the data into a single data frame. Then again the player-wise data present in a data frame was also merged to this main data frame to the details about each player. This will give us the data about the players.

Then the tweet text data was vectorized using a count vectorizer then fed into TF-IDF vectorizer to convert the text data into numbers. Then the data was split into train and test with 25% as the percentage of data in the test case.

Part : 3
Modelling
The target data was labeled encoded data where it is 1 if at the tweet posting time, the total home score is less than the total away team score, and 0 otherwise.
The shape of TF-IDF train matrix: (2117803, 72498)
Shape of TF-IDF test matrix: (705935, 72498)
LSI (latent semantic indexing) was used to decrease the number of features present in the vectorized output version. And it was run on different values of n_components to fit this hyperparameter that best suits the model.
The following values of n_components are used [1,10,100,200,500,1000,2000,50].
For each value of the LSI the percentage of the total variance explained was obtained and is shown in the graph below:



The best value of the number of the latent vectors was selected from the graph above, and then it was used for input.

This latent input(a case is also present where the LSI was not used, the naive case, this was also used) was then fed into a logistic regression model which is also in a hyperparameter tuning process with the following cases:

1) No regularization penalty case.
2) L1 regularization penalty case with c values of [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000,100000]
3) L2 regularization penalty with c values of [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000,100000]

The performances of these models were compared using ROC curves, Accuracy, recall, precession and F-1 scores that comes from the grid search algorithm.

The best model was selected.

For us our model kept on crashing while we were training our model.

The code for the normal no regularization penalty case is:

```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)

lrmodel = LogisticRegression(penalty='none',random_state=42,max_iter=100000)
pred_lr_model = lrmodel.fit(train_LSI,y_train).predict(test_LSI)

# Train logistic regression model
# lrmodel = LogisticRegression()
# lrmodel.fit(train_tfidf,y_train)

# # Evaluate model on test set
# pred_y = lrmodel.predict(test_tfidf)
print(classification_report(y_test, pred_lr_model))
```

Code for the L-1 regularization case:

```python
from sklearn.model_selection import GridSearchCV
clf_lr_l1 = LogisticRegression(penalty='l1',random_state=42,solver='liblinear',max_iter=100000)
param_grid = {'C': [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000,100000]}
grid_l1 = GridSearchCV(clf_lr_l1,param_grid,cv=5,scoring='accuracy')
grid_l1.fit(train_LSI,y_train)

pred_cv_lr_l1 = grid_l1.best_estimator_.predict(test_LSI)
print('Best Value of L1 Regularization Parameter:',grid_l1.best_params_['C'])
for l, n in zip(param_grid['C'],grid_l1.cv_results_['mean_test_score']):
    print(f'L1 Reg. Param.: {l}\t',f'Avg. Validation Accuracy: {n}')

print("Accuracy (best logistic classifer with L1 regularization):", accuracy_score(y_test,pred_cv_lr_l1 ))
print("Recall (best logistic classifer with L1 regularization):", recall_score(y_test,pred_cv_lr_l1 ))
print("Precision (best logistic classifer with L1 regularization):", precision_score(y_test,pred_cv_lr_l1 ))
print("F1-Score (best logistic classifer with L1 regularization):", f1_score(y_test,pred_cv_lr_l1 ))
```

Code for the L-2 regularization case:

```python
clf_lr_l2 = LogisticRegression(penalty='l2',solver='liblinear',random_state=42)
grid_l2 = GridSearchCV(clf_lr_l2,param_grid,cv=5,scoring='accuracy')
grid_l2.fit(train_LSI,y_train)
pred_cv_lr_l2 = grid_l2.best_estimator_.predict(test_LSI)

print('Best Value of L2 Regularization Parameter:',grid_l2.best_params_['C'])
for l, n in zip(param_grid['C'],grid_l2.cv_results_['mean_test_score']):
    print(f'L2 Reg. Param.: {l}\t',f'Avg. Validation Accuracy: {n}')

print("Accuracy (best logistic classifer with L2 regularization):", accuracy_score(y_test,pred_cv_lr_l2 ))
print("Recall (best logistic classifer with L2 regularization):", recall_score(y_test,pred_cv_lr_l2 ))
print("Precision (best logistic classifer with L2 regularization):", precision_score(y_test,pred_cv_lr_l2 ))
print("F1-Score (best logistic classifer with L2 regularization):", f1_score(y_test,pred_cv_lr_l2))
```