ECE 219: Large Scale Data Mining: Models and Algorithms Project 4: Regression Analysis

Sarah Madsen, Connor Roberts, Edwin Calderon

19 March 2021

1 Introduction

Common regression practices are explored throughout the following project. The analysis of Linear, Neural, and Boosted tree methods are explored and evaluated for three datasets, "Bike", "Suicide", and "Video Transcoding". Throughout the training of these algorithms, feature engineering is implemented to explore the most salient features in the datasets, and the influence they induce in the machine learning estimators.

Further analysis is done within subsections of algorithms and the results able to be obtained though adjacent models, such as linear, ridge, and lasso linear regression methods.

In addition to traditional methods, the group explores modern libraries in the analysis of the data. Gradient boosting methods are implemented through the use of CatBoost and LightGBM.

2 Bike Sharing Dataset

2.1 Data Inspection

Question 1 for Bike Dataset:

In the figure shown below we can see the absolute correlation between our three target variables, casual, registered, and count, and their features. For the most part we see all three of these target variables share high correlations with season, year, month, and temperature. What this is telling us is when we see a positive correlation between these variables and our target variables increasing. This means that these are likely the features most responsible for changes in our targets.

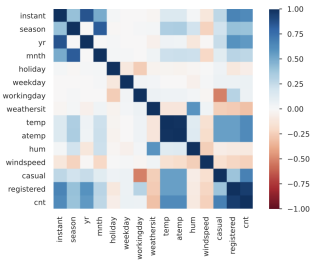


Figure 1: Pearson Correlation Matrix

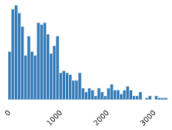


Figure 2: Histogram of casual riders

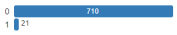


Figure 3: Histogram of holidays

Question 2 for Bike Dataset:

Shown above are the histograms for the number of casual riders and the number of holidays. We can see that both of these are relatively skewed towards one direction. In the case of the casual users we will be using this in combination with the registered users for our target variable so we do not need to worry about adjusting it. In terms of the histogram of the holidays, because this is a binary variable there is also no reason for us to adjust this when using the dateaset with our models.

Question 3 for Bike Dataset:

The bike dataset doesn't contain any categorical features so shown below are the boxplots of the target variables

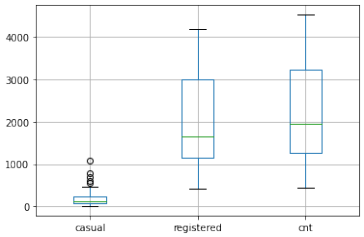


Figure 4: Boxplots of Numerical Target Variables

Question 4 for Bike Dataset: Shown below is just one of the months plotted against total count. To save space we limited the image to just one but in practice all the months looked quite similar. Interestingly, there seemed to be an uptick in count towards the end of each month we looked at.

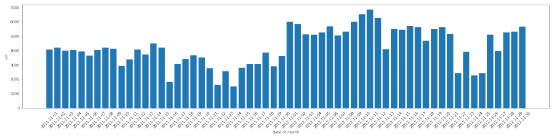


Figure 5: One Month vs Count

Question 9 for Bike Dataset:

Looking at the mutual information and f-score values shown below we can see that there are some features that are correlated with other features as well as features not incredibly related to our target so we would presume we would get similar or better scores on the feature selected dataset in comparison to the full dataset.

```
Mutual Information between count and features:

season :  0.2145429300245203
yr :  0.2773231994903549
mnth :  0.37862837263313187
holiday :  0.01151265746593055 4
weekday :  0.0448600426812793
workingday :  0.021816751837866 91
weathersit :  0.0649471208926089
temp :  0.38876797679497 83
atemp :  0.4644173617084324
hum :  0.04543535262581688
windspeed :  0.05534508900120061


 F Score and P-Value:

0 .  season , F score: 143.9676525909154 ,     P Value:  2.133996684343797e-30
1 .  yr , F score: 344.8905855356845 ,     P Value:  2.4835399044501753e-63
2 .  mnth , F score: 62.00462454833198 ,     P Value:  1.2431117778656476e-14
3 .  holiday , F score: 3.421441039972211 ,     P Value:  0.064759357926115
4 .  weekday , F score: 3.331091365174518 ,     P Value:  0.06839080695470057
5 .  workingday , F score: 2.7367422831913517 ,     P Value:  0.0984949616002635
6 .  weathersit , F score: 70.72929782920825 ,     P Value:  2.150975821424902e-16
7 .  temp , F score: 473.4717105349773 ,     P Value:  2.8106223975901415e-81
8 .  atemp , F score: 482.4543105289931 ,     P Value:  1.8545041252824314e-82
9 .  hum , F score: 7.461939996345346 ,     P Value:  0.006454143325437774
10 .  windspeed , F score: 42.437841593463865 ,     P Value:  1.3599586778863725e-10
```

Figure 6: Mutual Information and F Score

3 Suicide Dataset

3.1 Data Inspection

Below are the pictures for the data inspection of the target ”suicides no’.
Question 1 for Suicide Dataset:

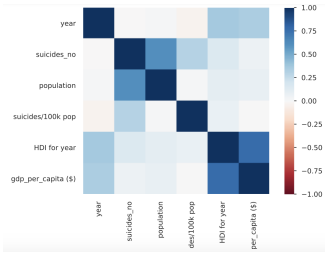


Figure 7: Pearson correlation matrix

Question 2 for Suicide Dataset:

These histograms and Pearson correlation matrix are extracted from the pandas profiling scheme, and further illustrates some of the same concerns regarding the distribution of the categorical features discussed in the following section.

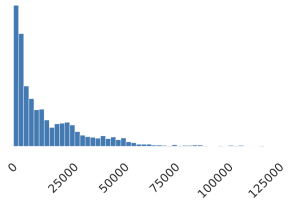


Figure 8: Suicides/100k pop histogram

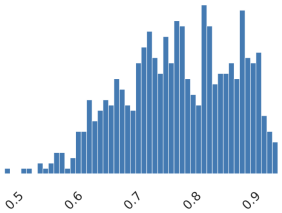


Figure 9: HDI histogram

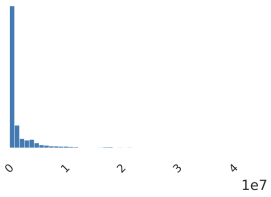


Figure 10: Population histogram

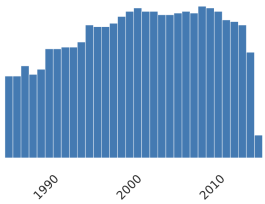


Figure 11: Year histogram

Question 3 for Suicide Dataset:

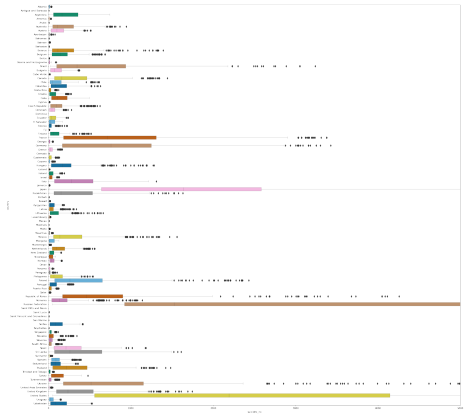


Figure 12: Feature: Country

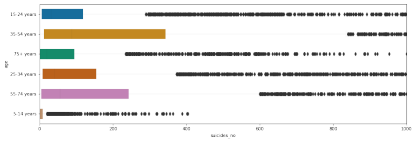


Figure 13: Feature:Age

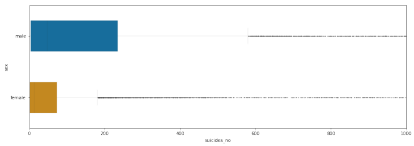


Figure 14: Feature: Sex

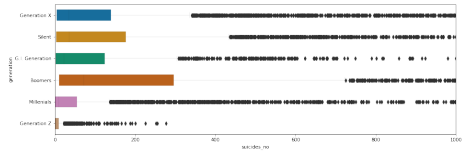


Figure 15: Feature: Generation

In the boxplots above, the information of the countries' boxplots shows a large variability between the country itself and regions. This illustrates that in the classification methods, the large non-uniformity of the countries should be addressed, and an initial attempt is implemented by partitioning the data. The generation boxplots shows a large number of outliers, as does the sex boxplots. Outliers such as these are an obstacle in classification models, since they typically skew the results in an unfavorable way.

Below are the pictures for the data inspection of the target "suicides/100k pop'.

The boxplots below further illustrate the variance contained in the data as it has many of the same distributions seen in the previous boxplots. This implies a great skewness that needs to be further addressed in the project.

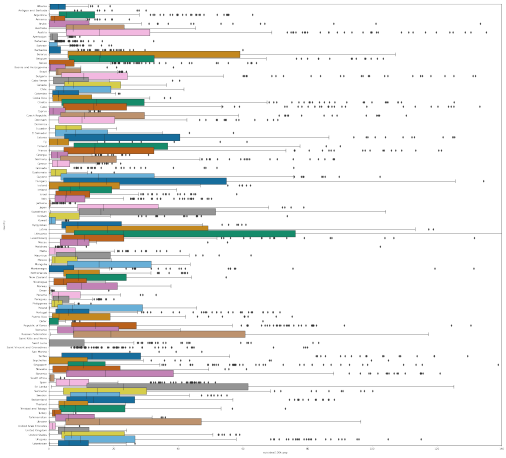


Figure 16: Feature: Country

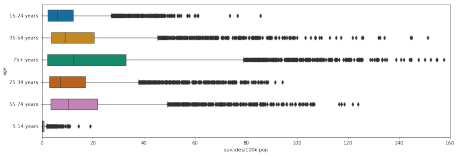


Figure 17: Feature: Age

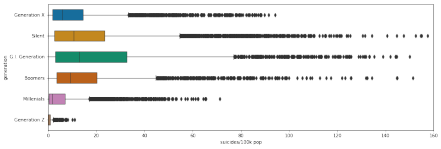


Figure 18: Feature: Generation

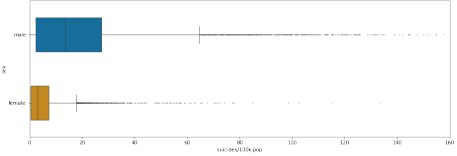


Figure 19: Feature: Sex

Question 5 for Suicide Dataset:

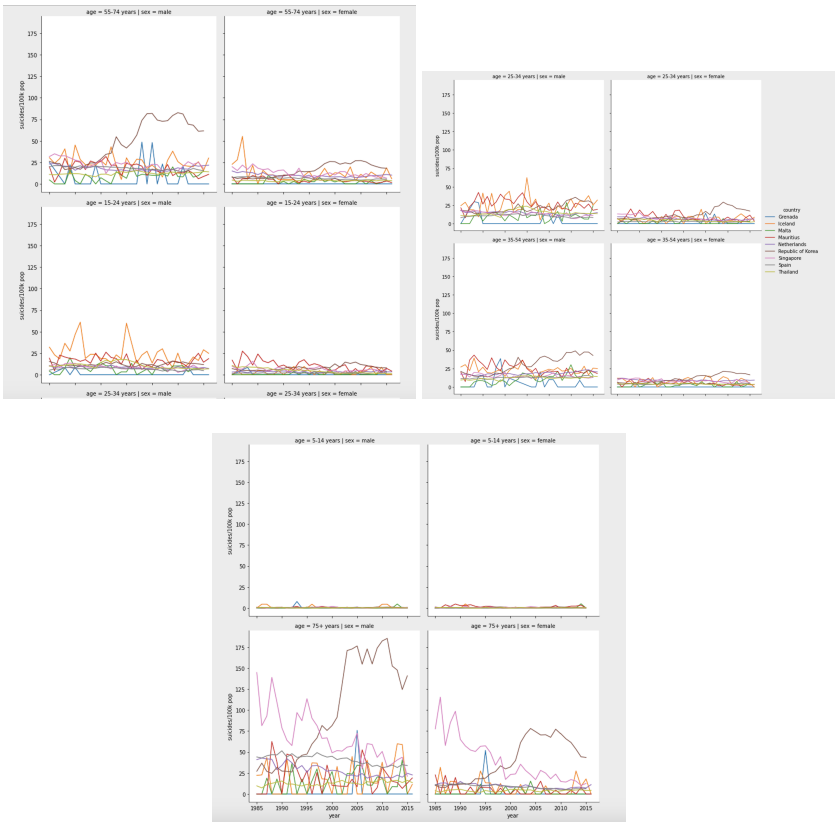


Figure 20: 'Suicides/100k pop" against time for different ages/genders

Above, the plots give us a greater look into the data and how feature selection may be used to partition the features. It is seen that a subset of the data is somewhat much more uniform when smaller subsets are isolated.

Question 7: In applications of one-hot encoding, information of the labels is distorted or discarded as the feature space grows. Scalar encoding holds a numerical hierarchy.

Question 8: Feature Selection Process of Suicide Dataset:

```
Mutual Information between suicides/100k and features:

Africa :  0.004579478576135276
Asia :  0.003715747251587409
Europe :  0.04126274377876005
North America :  0.050240122554849886
Oceania :  0.006494170736068394
South America :  0.013313197231321183
Female :  0.13431597132849138
Male :  0.1377052435064554
15-24 :  0.02579154277047646
35-54 :  0.01841160385215579
5-14 :  0.017993053120596514
55-74 :  0.22738611448572543
75+ :  0.020500511442396174
Boomers :  0.041139768727162185
G.I. Generation :  0.023490744204065273
Generation X :  0.027135513966892955
Generation Z :  0.01501621226396388
Millenials :  0.05179906343800633
Silent :  0.04313692934279789
Population :  0.026651937324447594
GDP Per Capita :  0.5877271834010536
Number of Suicides :  0.19018664161743715
Year :  0.721067844069327
```

Figure 21: MI Scores

```
 F Score and P-Value:

0 .  Africa , F score: 66.93963431264717 ,     P Value:  2.9178246134584683e-16
1 .  Asia , F score: 66.18741307664381 ,     P Value:  4.2696414590000695e-16
2 .  Europe , F score: 1222.647181532954 ,     P Value:  3.4814323124447e-262
3 .  North America , F score: 656.456248139557 ,     P Value:  4.036575288863593e-143
4 .  Oceania , F score: 4.407559194330083 ,     P Value:  0.0357889872516574
5 .  South America , F score: 19.069537289838756 ,     P Value:  1.2649570815338951e-05
6 .  Female , F score: 5035.427899106122 ,     P Value:  0.0
7 .  Male , F score: 5035.427899106153 ,     P Value:  0.0
8 .  15-24 , F score: 233.90850746820163 ,     P Value:  1.3713970108595773e-52
9 .  35-54 , F score: 6.1364218826676922 ,     P Value:  0.013248396840543672
10 .  5-14 , F score: 70.57193326529135 ,     P Value:  4.646500819275748e-17
11 .  55-74 , F score: 2490.576486714397 ,     P Value:  0.0
12 .  75+ , F score: 173.89594422337152 ,     P Value:  1.3714946624158368e-39
13 .  Boomers , F score: 2065.611860765473 ,     P Value:  0.0
14 .  G.I. Generation , F score: 62.875561990211516 ,     P Value:  2.2839457691318597e-15
15 .  Generation X , F score: 1089.9982012836413 ,     P Value:  1.661602407534953e-234
16 .  Generation Z , F score: 118.69386675648262 ,     P Value:  1.3894620205580866e-27
17 .  Millenials , F score: 654.7693474710669 ,     P Value:  9.2135873457456627e-143
18 .  Silent , F score: 1185.0732848023233 ,     P Value:  2.3381110030483988e-254
19 .  Population , F score: 739.5625042761219 ,     P Value:  9.468085850175482e-161
20 .  GDP Per Capita , F score: 1.909580251300788 ,     P Value:  0.167020989064028
21 .  Number of Suicides , F score: 0.08864797749177898 ,     P Value:  0.7659052836271877
22 .  Year , F score: 2886.4074133700137 ,     P Value:  0.0
```

Figure 22: F-Scores

The scores calculated above were used throughout the iterative process of feature selection and model implementation. It is known that the higher the F-score of a feature/lower p-value corresponds to a greater significant relationship between that feature and a target variable. These scores are calculated by partitioning features further, such as the age group of the main feature - ”generation”. Decisions to keep the feature in the modeling are guided by the scores seen above.

4 Video Transcoding Time Dataset

4.1 Question 1

The heatmap for the video transcoding dataset is shown in Figure 23. The features with the highest correlation to the target variable are umem, o-width, and o-height. High correlation between features suggests they are related in some way. This implies these features will be the most powerful during the regression tasks.

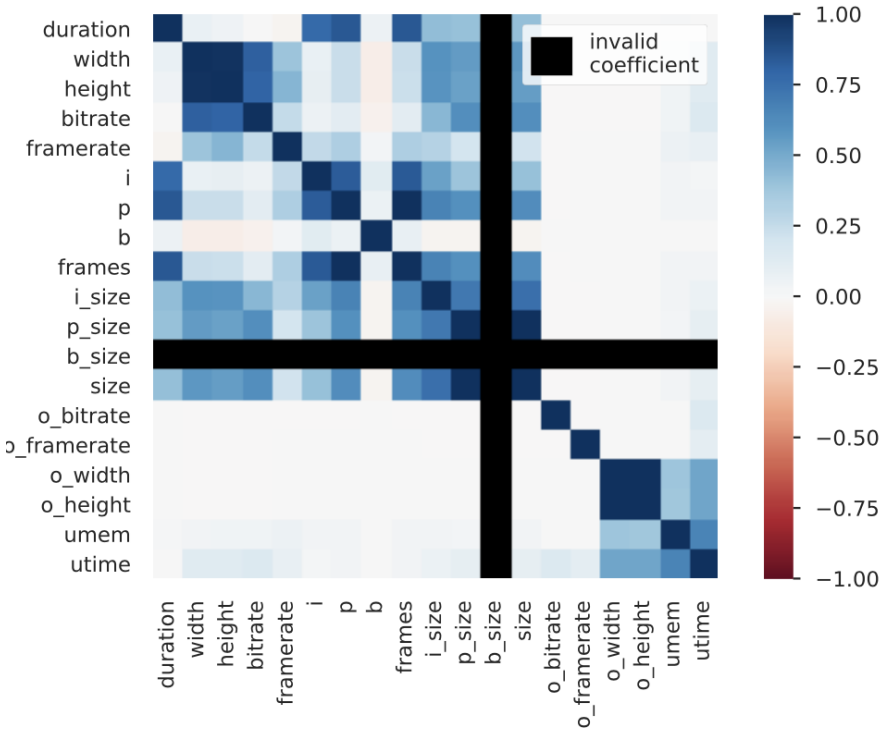


Figure 23: The Pearson correlation Correlation Matrix for the video transcoding dataset.

4.2 Question 2

The histograms for the video transcoding dataset can be found in the attached video transcoding report pdf. The historgram for the feature i, which is the number of frames in a video, is shown in Figure 24 as an example. The distribution of this feature is skewed towards the y-axis; most of the values of i are below 200. To handle skewness in data we can normalize data by subtracting the mean and dividing by the standard deviation. This forces the distribution of values of the feature to follow the standard normal distribution.

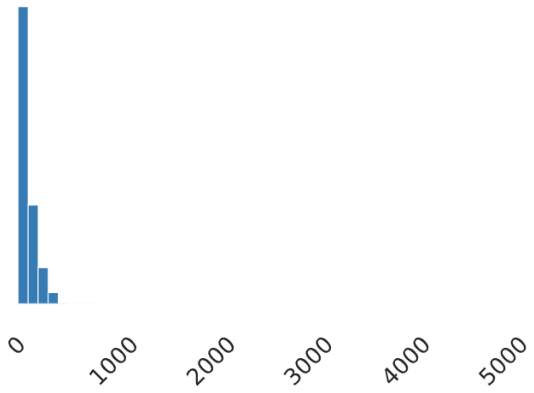


Figure 24: The histogram for the number of frames in a video for the video transcoding dataset. The distribtuion is skewed to the right: the median is less than the mean.

4.3 Question 3

The boxplots for the categorical features versus the target variable for the video transcoding dataset are shown in Figures 25 and 26. From these boxplots, we can see that the codec features mostly fall into the same utime ranges; the codec feature does not tell us much about how the data sample is classified. To contrast, the o-codec feature has a lot more variability. For instance, we can see that if a data sample has an o-codec of flv, it is likely to have a very low utime, whereas an o-codec of h264 is more liekly to have a larger utime. Therefore, the o-codec feature gives more information about the video data sample classification.

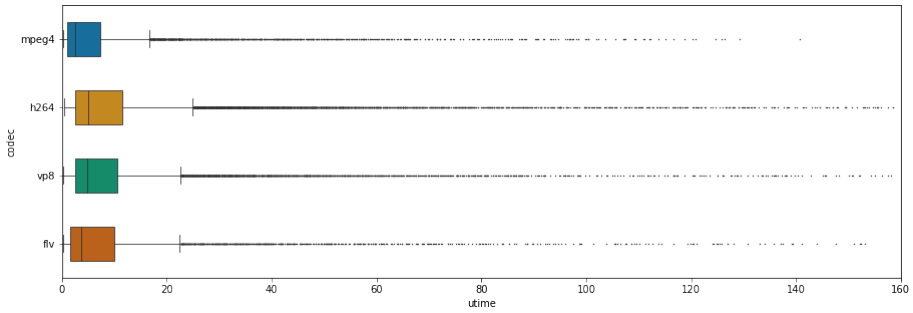


Figure 25: The boxplot for the cateogical feature codec plotted against the target variable utime, for the video transcoding dataset.

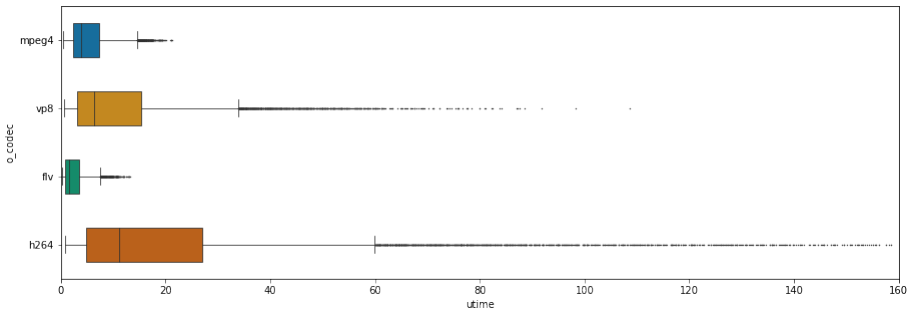


Figure 26: The boxplot for the cateogical feature o-codec plotted against the target variable utime, for the video transcoding dataset.

4.4 Question 6

The distribution of the target variable, utime, is shown in Figure 27. There is a wide range to the data; most of the data are in the range 0 to 20, but some of the data are above 100. Additionally, there is right-handed skewness of the variable: the median of 4.408 is less than the mean of 9.996.

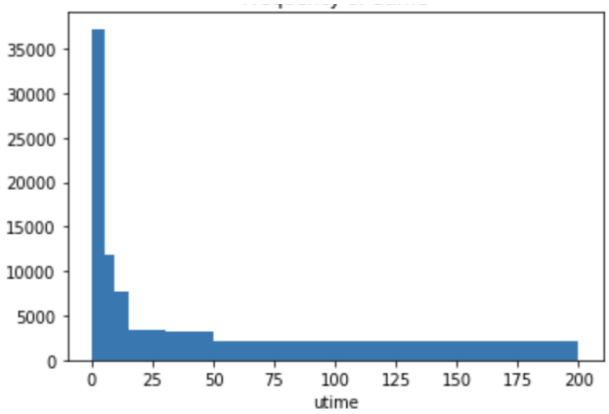


Figure 27: The histogram of the target variable, utime, for the video transcoding dataset.

4.5 Question 7

The categorical features for the video transcoding dataset are codec and ocodec, which are the coding standards used for each video in the dataset. One-hot encoding leads to an issues called multicollinearity. This occurs when there is a dependency introduced between otherwise independent features. The output of one of the features can easily be predicted from the other features. This is a problem with linear and logistic regression

models. On the other hand, numerical encoding introduces a hierarchy in the data when there might not have been one before. For instance if "mpeg4" is encoded as 0 and "flv" is encoded as 1, this suggests that flv $>$ mpeg4, which is not necessarily true.

4.6 Question 8

The features for the transcoding dataset were standardized to have a zero mean and unit variance.

4.7 Question 9

The video transcoding dataset shows high correlation between features, suggesting not all features are necessary for proper regression classification. We expect that dropping features that are highly correlated will result in better regression model performance.

5 Linear Regression

Lasso and Ridge are the regularized forms of least squares within this context, and the penalties are, respectively, L1 and L2 normalization. Lasso results in a more sparse representation of the data than L2.

The regularization scheme affects the results since L1 and L2 affect our feature spaces through the model's pipleline and attenuates features by penalizing them based on their values with respect to the other data points and its interaction within the model.

P-values suggests how changes in a predictor are associated with changes in the response. Low p-values for a feature indicate a strong relationship with the target variables, and, conversely, a high p-value indicates a weak relationship. In the feature selection of the project, variables with low p-values are kept.

Feature scaling does play a large role in linear regression when regularization is implemented since it allows us to normalize the feature space and reduce the variability by these outliers. Feature scaling is not required in regular least squares since no penalties are applied.

Bike Dataset Average RMSE vs. Linear Regression Methods

	Lasso RSME Average	Ridge RSME Average	RLS RSME Average
Avg. Train RMSE	858.43	858.43	858.39
Avg. Test RMSE	965.63	965.60	968.53
Feature Selected Train	887.27	887.28	887.26
Feature Selected Test	979.45	978.57	980.96

Suicide Dataset Average RMSE vs. Linear Regression Methods

	Lasso RSME Average	Ridge RSME Average	RLS RSME Average
Avg. Train RMSE	15.51	14.53	14.54
Avg. Test RMSE	15.41	14.71	14.70
Feature Selected Train	15.63	14.77	14.77
Feature Selected Test	15.53	14.75	14.75

Transcoding Dataset Average RMSE vs. Linear Regression Methods

	Lasso RSME Average	Ridge RSME Average	RLS RSME Average
Avg. Train RMSE	12.60	12.36	12.36
Avg. Test RMSE	12.59	12.40	1.40e+09
Feature Selected Train	12.61	12.38	12.38
Feature Selected Test	12.59	12.39	12.39

Although the results varied depending on the features selected,the results are comparable to their non-feature selected counterparts. The results in general, through iterations, indicate that Ridge Linear Regression yields marginal better results.

6 Polynomial Regression

Polynomial Results for Bike Dataset:

	degree_1	degree_2	degree_3	degree_4
LR train	858.399295	599.096683	8.160701e+02	2.258986e-10
LR test	968.531165	1154.735314	1.297224e+14	2.894874e+04
Ridge train	858.427857	607.435816	3.962648e+02	1.955076e+02
Ridge test	965.600245	957.521685	1.096889e+03	3.007559e+03
Lasso train	858.399306	599.884963	3.906269e+02	1.930105e+02
Lasso test	968.484048	986.733945	1.163403e+03	2.702554e+03

Figure 28: Before Feature Selection

	degree_1	degree_2	degree_3	degree_4
LR train	887.259271	856.847224	591.406950	5.080521e+02
LR test	980.962990	1560.723829	2605.493038	5.231955e+13
Ridge train	887.278654	709.026079	581.127033	4.994182e+02
Ridge test	978.571129	982.641457	987.625549	1.725515e+03
Lasso train	887.259279	705.577900	577.340842	4.953924e+02
Lasso test	980.929545	1016.780005	1126.255455	2.090671e+03

Figure 29: After Feature Selection

Polynomial Results for Suicide Dataset:

	degree_1	degree_2	degree_3	degree_4
LR train	17.429160	16.239013	14.979989	17.267470
LR test	17.472831	18.701238	37.068653	126.440270
Ridge train	17.429160	16.235515	14.919723	15.009597
Ridge test	17.472831	18.701747	40.076331	182.017151
Lasso train	17.429160	16.269587	15.048106	14.515364
Lasso test	17.472839	18.488901	30.651420	42.655445

Figure 30: Before Feature Selection

	degree_1	degree_2	degree_3	degree_4
LR train	12.377997	9.988083	7.100196e+00	3.805100e+00
LR test	12.398937	11.429237	6.238932e+10	5.470097e+10
Ridge train	12.377997	9.994616	7.127266e+00	3.919100e+00
Ridge test	12.398913	11.337629	1.088818e+02	2.648555e+03
Lasso train	12.378660	10.017657	7.208552e+00	4.205603e+00
Lasso test	12.398197	10.915085	9.379914e+00	1.010604e+01

Figure 31: After Feature Selection

Polynomial Results for Transcoding Dataset:

	degree_1	degree_2	degree_3
LR train	1.236795e+01	9.931816e+00	6.916923e+00
LR test	5.529480e+08	9.583534e+12	1.205311e+13
Ridge train	1.236816e+01	9.925467e+00	6.944915e+00
Ridge test	1.240558e+01	1.561733e+01	1.387846e+03
Lasso train	1.236891e+01	9.954183e+00	7.064330e+00
Lasso test	1.240315e+01	1.133921e+01	9.341490e+01

Figure 32: Before Feature Selection

	degree_1	degree_2	degree_3	degree_4
LR train	12.377997	9.988083	7.100196e+00	3.805100e+00
LR test	12.398937	11.429237	6.238932e+10	5.470097e+10
Ridge train	12.377997	9.994616	7.127266e+00	3.919100e+00
Ridge test	12.398913	11.337629	1.088818e+02	2.648555e+03
Lasso train	12.378660	10.017657	7.208552e+00	4.205603e+00
Lasso test	12.398197	10.915085	9.379914e+00	1.010604e+01

Figure 33: After Feature Selection

For the inverse of certain features the Train RMSE was 9.99 and the Test RMSE was 18.80 when Ridge was used in tandem with a polynomial order of two. The results seem to deprectiate. Accuracy was not boosted within the parameters tested. Intuitively, more iterative means of feature selection would create a better metric to remove features that are not beneficial for the models implemented without creating a greater dependency among features.

Increasing the order of polynomial degree - as seen in the tables above - can negatively impart the models. The degree of the polynomial should not be increased too significantly - depending on the data - since outliers become more pronounced and can cause an undesired skewness. The 'curse of dimensionality' also illustrates that the feature space may become too distorted and negatively limit the models ability to regularize. The most salient features align with the feature preprocessing steps shown throughout the datasets inspections.

7 Neural Network

Linear regression is very good when used on linear datasets but because our sets are so complex with multiple features it becomes very hard for that model to perform well. Neural networks on the other hand are very good at modeling sets of data with complex features which is why the perform so well in this situation.

A comprehensive search was ran with layers from size 1 to 10, nodes from size 50-1000, and decay values from 1 to 0.0001. After testing these all on the bike dataset and evaluatuing the tradeoff between accuracy and time taken we settled on 10 layers of 200 nodes with a decay value of 0.01.

With the datasets we were given we should be using the relu activation function. There are two main reasons we choose this activation function. The first is relu is a nonlinear function which is essential in allowing us to effectively model our nonlinear data. Secondly, the calculation load of the relu activation function is less than that of functions that perform similarly which leads us to prefer to use it over other options

There were other options for number of nodes that performed better but exponentially increased how long it would take our model to train while having our RMSE only decreasing minimally. In addition to greatly increasing training time, neural networks

also have the potential to overfit the train data as the options for layers and number of nodes are essentially limitless.

7.1 Results

Average RMSE vs. Different Datasets

	Bike Dataset	Suicide Dataset	Video Dataset
Avg. Train RMSE	297.26	5.42	1.64
Avg. Test RMSE	760.80	7.90	4.76
Feature Selected Train	516.41	11.68	4.45
Feature Selected Test	892.57	14.65	5.61

8 Random Forest

Generally, the maximum number of features improved the robustness of the algorithm - as did the number of estimators in the Random Forest algorithm. Empirically, it was found throughout the data sets, that what most noticeably affected the results was the depth of trees. By limiting the depth of trees, it was found that a significant reduction of performance was seen. The models did appear to reach a threshold of performance, regardless of parameters chosen. Optimal results are within this section and are performed using 500 estimators.

Random forest performs well due to its ability to build a decision tree based on the number of random records, and it takes the average of all the values predicted from all the forests in the trees. The large features space of random forest works well in accounting for missing values and with data that, in general, is not processed extensively. As a result, there is a significant regularization effect built in the random forest scheme when the training and testing RMSEs are compared.
The out of bag error is a measure of validating the results from the random forest training sets, since the algorithm partitions the data randomly.OOB Error is the number of incorrect classified OOB Samples.

The F - scores calculated on the raw data indicated the most important features used through the implementation of the "rf.feature_importance" method. The features selected for the root node is the feature for the data split that will split the data as much as possible. At each stage, the split is made to create a maximum homogeneity in the resulting groups

The R^2 score is a statistical description of how data samples fit along a linear model.

Average RMSE vs. Different Datasets

	Bike Dataset	Suicide Dataset	Video Dataset
Avg. Train RMSE	676.98	18.05	7.43
Avg. Test RMSE	956.14	10.63	7.48
Feature Selected Train	702.69	16.69	7.43
Feature Selected Test	971.09	13.72	7.48

As seen in many cases, the random forest results are uniform due to its bragging methods. The results when parameters were varied were constant, and the only significant difference were seen in an improvement in the OOB scores once feature selection was implemented.
It is reasonable, therefore, that improvements could marginally be seen with more rigorous feature selection - predominately in the OOB scores as more iterations were tried and tested.

	Average Out of Bag Error of Different Datasets after Feature Selection
Bike Dataset	0.847
Suicide Dataset	0.464
Transcoding Dataset	0.781

The results so far are a testament of the continuous process of evaluating one's data with a few metrics in mind, and how they would shape and evolve as they are tested through the model's implemented. So far, the main features indicated by the node do in fact correspond with the features expected to contribute significantly. A notable result seen through the Random Forest implementation is the ability to improve over fitting and a general improvement in results with low computational resources compared to other models, such as NNs and PR.

Below are selected Decision Trees for all datasets, with a depth of 4:

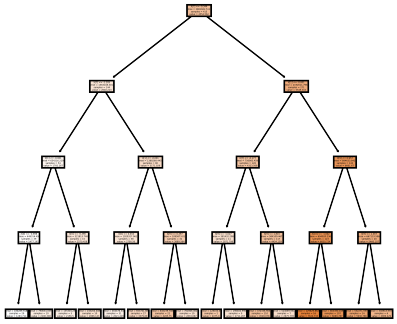


Figure 34: Bike Dataset Tree

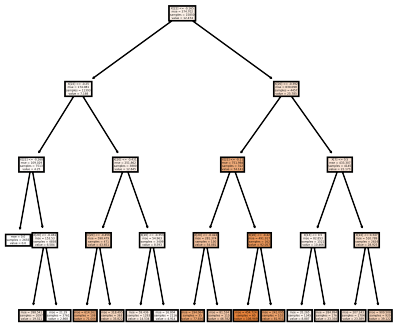


Figure 35: Master Dataset Tree

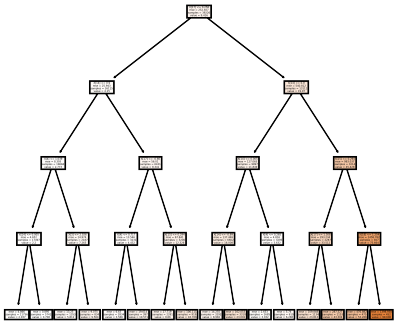


Figure 36: Transcoding Dataset Tree

9 LightGBM, CatBoost and Bayesian Optimization

9.1 Question 24

There are five main hyperparameters to tune for LightGBM: number of estimators, number of leaves, max depth, minimum child samples, and the learning rate. The number of estimators parameters controls the maximum number of trees to fit in the boosting algorithm. For a dataset with thousands of data samples, optimal numbers usually range from 100 to 1000. The number of leaves parameter controls the complexity of each tree model; it limits the number of leaves in each tree. This is an important parameter to control overfitting. The rule of thumb is that it should be less than $2^(max_depth)$. The maximum depth parameter limits the depth of each tree. Optimal values usually range from 1 to 10, as very deep trees tend to overfit the training data. The minimum number child samples parameter controls the number of training samples used for each decision leaf. Increasing this value can help overfitting, but making it too large can cause underfitting. For larger datasets, values in the hundreds to thousands is usually sufficient. The final parameter, learning rate, affects the gradient step size. A very small learning rate will cause an increase in training time and result in more iterations of training.

The main hyperparameters for CatBoost are similar to the hyperparameters for LightGBM: number of iterations, depth, l2 leaf regularization, border count, and learning rate. The number of iterations is the same as the number of trees to fit. The depth parameter controls the maximum depth of each tree. Border count is the same as minimum child samples in LightGBM. The learning rate parameter is also the same. The main different hyperparameter is the l2 leaf reg parameter. This parameter controls the amount of l2 regularization applied to each tree.

9.2 Question 25

The hyperparameter search spaces for LightGBM and CatBoost were similar. The number of trees was ranged on a logarithmic uniform (log-uniform) scale from 100 to 1000. The depth was varied by two on a range of 2 to 10. The minimum child samples for LightGBM was varied from 50 to 500, whereas the border count for CatBoost was varied from 100 to 5000 because better performance was achieved with high values for CatBoost. The learning rate for both was chosen from 0.001, 0.02, 0.05, and 0.1. For LightGBM the number of leaves was ranged on a log-uniform scaled from 16 to 64. The l2 leaf regularization for CatBoost was chosen from 0, 0.05, 1, 3, 5, and 10. Both the LightGMB and the CatBoosting models were optimized to the bike dataset.

For LightGBM, the optimal hyperparameters were a learning rate of 0.05, a max depth of 4, 50 minimum child samples, 977 trees, and 45 leaves per tree. These hyperparameters were able to achieve a test RMSE of 775.18 on the non-feature selected bike dataset. The same parameters were trained on the feature selected bike dataset and were able to achieve a train RMSE of 845.06 and a test RMSE of 855.39.

For CatBoost, the optimal hyperparameters were a border count of 200, a max depth of 2, l2 leaf reg of 3, 257 trees, and a learning rate of 0.05. These hyperparameters were able to achieve a test RMSE of 702.06 on the non-feature selected bike dataset. The

same parameters were trained on the feature selected dataset. The test RMSE was 845.74 and the train RMSE was 793.54.

It is interesting to note that the feature selected dataset performed worse for these algorithms. This is probably because most of the features for the bike dataset had low F1 scores, indicating low correlation between features. Since the bike dataset was the smallest dataset to begin with, there was no need to drop any features.

9.3 Question 26

For both datasets, the minimum child samples (aka border count) was a very import hyperparameter for performance. Figure 37 shows the train RMSE plotted as a function of minimum child samples for the LightGBM model. As the number of samples increases, the train RMSE also increases even if all the other optimal hyperparameters are still used. Clearly, this hyperparameter is very important for model performance.

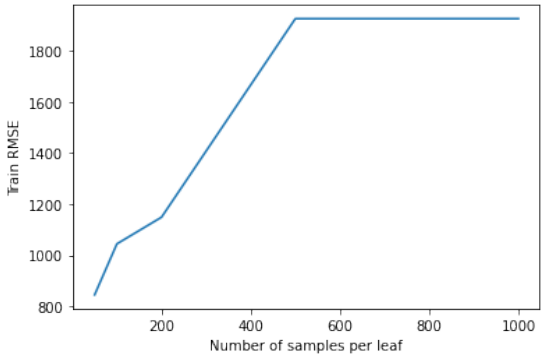


Figure 37: The training set RMSE as a function of minimum child samples for the LightGBM model. Similar results were seen for the CatBoost model. Model performance decreases as the number of samples per leaf is increased.

For both models the depth of the tree acts as a regularizer to reduce overfitting. In Figure 38, the training RMSE and testing RMSE are plotted as a function of the maximum depth of the tree for the LightGBM algorithm. As the depth of the tree is increased, the training RMSE decreases but the test RMSE increases signaling the model is overfitting. Thus, keeping the depth of the trees smaller acts as a regularizer on the model. While the results here are plotted for the LightGBM model, similar results were seen for the CatBoost model as well.

The learning rate affects the training efficiency of both LightGBM and CatBoost. Decreasing the learning rate results in a longer training time as the model takes longer to converge.

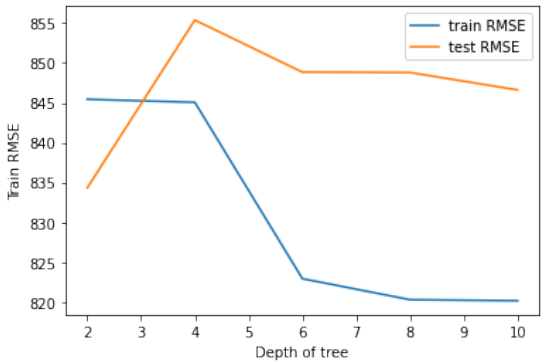


Figure 38: The training RMSE and the testing RMSE plotted as a function of the maximum depth of the trees for the LightGBM model. This hyperparameter acts a regularizer for the model.

9.4 Question 27

The below table shows the training versus testing RMSE for a 10-fold cross-validation on the best hyperparameters on both the LightGBM and CatBoosting models. For both models the training RMSE is lower than the validation/testing RMSE. This is because the model is trained and thus optimized to the training data. The validation data represents ”unseen” data and thus we expect the performance to drop on the validation set. The best boosting models are those that minimize the difference in the training and testing RMSE.

	LightGBM	CatBoost
Train RMSE	845.06	793.54
Test RMSE	855.39	845.74

Mount Drive

```
from google.colab import drive
import sys
import os
drive.mount('/content/drive/')
sys.path.append('/content/drive/MyDrive/4. Project 4 - Regression Analysis/')
```

```
Mounted at /content/drive/
```

```
%matplotlib inline
import matplotlib
matplotlib.rcParams['backend'] = "Qt4Agg"
import matplotlib.pyplot as plt
```

Import libraries

```
# !pip uninstall pandas-profiling
# !pip install pandas-profiling --upgrade
# !pip install pandas --upgrade
!pip install pycountry_convert --quiet
# !pip install matplotlib --upgrade
```

```
     |████████████████████████████████| 10.1MB 17.2MB/s
     |████████████████████████████████| 245kB 64.2MB/s
  Building wheel for pycountry (setup.py) ... done
ERROR: datascience 0.10.6 has requirement coverage==3.7.1, but you'll have coverage 5.5
ERROR: datascience 0.10.6 has requirement folium==0.2.1, but you'll have folium 0.8.3 w
ERROR: coveralls 0.5 has requirement coverage<3.999,>=3.6, but you'll have coverage 5.5
ERROR: pytest-cov 2.11.1 has requirement pytest>=4.6, but you'll have pytest 3.6.4 whic
ERROR: pytest-mock 3.5.1 has requirement pytest>=5.0, but you'll have pytest 3.6.4 whic
```

```
'''run if the pandas profiler is causing issues'''
%matplotlib inline
import matplotlib
from matplotlib import pyplot as plt
```

```
'''run normally'''
import pandas as pd
import numpy as np

import pycountry_convert as pc
```

```
from pandas_profiling import ProfileReport
import seaborn as sns

###sklearn imports
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import f_regression
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso, Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_predict
from sklearn.tree import export_graphviz
from sklearn import tree
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestRegressor
```

Import Datasets

```
'''
Import dataset
'''

day_df = pd.read_csv('/content/drive/MyDrive/4. Project 4 - Regression Analysis /day.csv')


master_df = pd.read_csv('/content/drive/MyDrive/4. Project 4 - Regression Analysis /master.cs


transcoding_df = pd.read_csv('/content/drive/MyDrive/4. Project 4 - Regression Analysis /trar



day_df
```

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	te
0	1	2011-01-01	1	0	1	0	6	0	2	0.3441
1	2	2011-01-02	1	0	1	0	0	0	2	0.3634
2	3	2011-01-03	1	0	1	0	1	1	1	0.1963
3	4	2011-01-04	1	0	1	0	2	1	1	0.2000
4	5	2011-01-05	1	0	1	0	3	1	1	0.2269
...	...	...	...	...	...	...	...	...	...	
726	727	2012-12-27	1	1	12	0	4	1	2	0.2541
727	728	2012-12-28	1	1	12	0	5	1	2	0.2533

```
master_df
```

	country	year	sex	age	suicides_no	population	suicides/100k pop	country-
0	Albania	1987	male	15-24 years	21	312900	6.71	Albania
1	Albania	1987	male	35-54 years	16	308000	5.19	Albania
2	Albania	1987	female	15-24 years	14	289700	4.83	Albania
3	Albania	1987	male	75+ years	1	21800	4.59	Albania
4	Albania	1987	male	25-34 years	9	274300	3.28	Albania
...	...	...	...	...	...	...	...	

```
transcoding_df
```

	id	duration	codec	width	height	bitrate	framerate	i	p	b	fra
0	04t6-jw9czg	130.35667	mpeg4	176	144	54590	12.000000	27	1537	0	1
1	04t6-jw9czg	130.35667	mpeg4	176	144	54590	12.000000	27	1537	0	1
2	04t6-jw9czg	130.35667	mpeg4	176	144	54590	12.000000	27	1537	0	1
3	04t6-jw9czg	130.35667	mpeg4	176	144	54590	12.000000	27	1537	0	1
4	04t6-jw9czg	130.35667	mpeg4	176	144	54590	12.000000	27	1537	0	1
...	...	...	...	...	...	...	...	...	...	...	
68779	ZWEN-71BqPs	972.27100	h264	480	360	278822	29.000000	560	28580	0	29
68780	zWQN-bqqg0o	129.88100	vp8	640	480	639331	30.162790	36	3855	0	3
68781	zX17-vi0sqQ	249.68000	vp8	320	240	359345	25.068274	129	6113	0	6
68782	zyiT-TzxIpk	183.62334	h264	1280	720	2847539	29.000000	98	5405	0	5
68783	zZKo-QsY86U	294.61334	mpeg4	176	144	55242	12.000000	61	3474	0	3

68784 rows × 22 columns

Day Report

```
    day_report = ProfileReport(day_df)
    print(repr(day_report.report))
    print(day_report.report.content)
    day_report.to_notebook_iframe()
```

Summarize dataset: 29/29 [00:18<00:00, 1.37s/it,

100% Completed]

Master Report

```
{"body": container(name="root"), "footer": html, "name": "root"}
```

```
master_report = ProfileReport(master_df)
print(repr(master_report.report))
print(master_report.report.content)
master_report.to_notebook_iframe()
```

Summarize dataset: 100% 25/25 [00:09<00:00, 1.21it/s, Completed]

Generate report structure: 100% 1/1 [00:06<00:00, 6.64s/it]

```
Root
{'body': Container(name=Root), 'footer': HTML, 'name': 'Root'}
```

Render HTML: 100% 1/1 [00:01<00:00, 1.38s/it]

Transcoding Report

```
transcoding_report = ProfileReport(transcoding_df)
print(repr(transcoding_report.report))
print(transcoding_report.report.content)
transcoding_report.to_notebook_iframe()
```

Summarize dataset: 35/35 [00:55<00:00, 6.24s/it, 100% Completed]

Generate report structure: 100% 1/1 [00:11<00:00, 11.41s/it]

```
Root
{'body': Container(name=Root), 'footer': HTML, 'name': 'Root'}
```

Render HTML: 100% 1/1 [00:08<00:00, 8.21s/it]

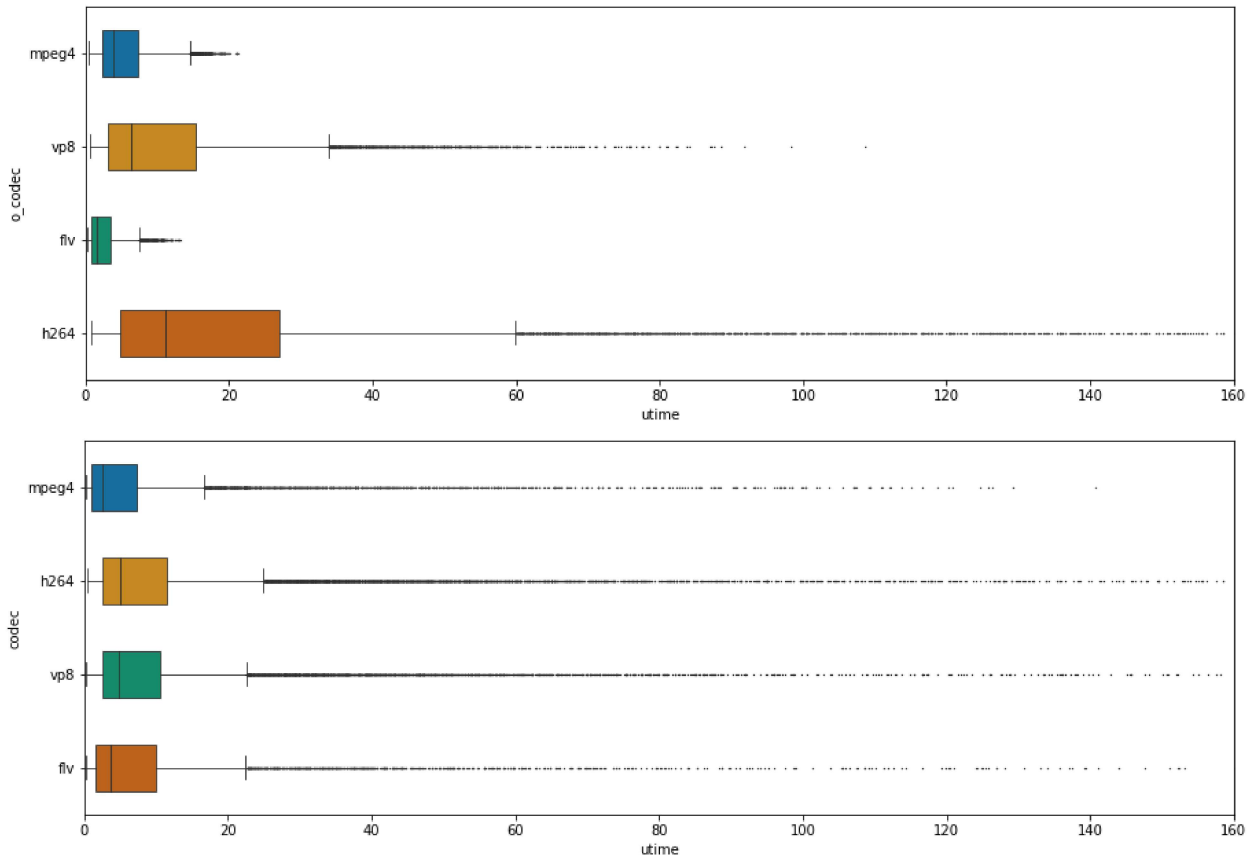
Overview

Transcoding Boxplots

Number of variables 22

```
trans_target = ['utime']
cols = ['o_codec', 'codec']

a4_dims = (15, 5)
fig, ax = plt.subplots(figsize=a4_dims)

df  = transcoding_df
sns.boxplot(y=df['o_codec'], x=df['utime'],
                data= df,
                palette="colorblind", width=.5, linewidth =.8, ax = ax, fliersize = .5
                )
plt.xlim([0, 160])

a4_dims = (15, 5)
fig, ax = plt.subplots(figsize=a4_dims)

df  = transcoding_df
sns.boxplot(y=df['codec'], x=df['utime'],
                data= df,
                palette="colorblind", width=.5, linewidth =.8, ax = ax, fliersize = .5
                )
plt.xlim([0, 160])
```

```
(0.0, 160.0)
```



Feature selection

```
#https://github.com/marcotcr/lime/issues/293
#https://hub.packtpub.com/4-ways-implement-feature-selection-python-machine-learning/
print(df.shape)
print(df.iloc[:,15].shape)

def mi_implement(df, target):

  for x in target:
    mi = mutual_info_regression(df.iloc[:,2:15],df.loc[:,x] ) ##error with dtime
    d[x]= mi

def f_implement(df, target):

  for x in target:
```

```
    f_ = f_regression(df.iloc[:,2:15],df.loc[:,x] ) ##error with dtime
    f[x]= f_

f={}
d = {}

mi_implement(day_df, bike_target_vars)

f_implement(day_df, bike_target_vars)

import numpy as np ## will move to import lib sub
def print_mi_scores():
  for x, y in d.items():
    print('\nThe mutual_info_regression for {} are: {} '.format(x,y))
    print(np.argsort(d[x]))


def print_f_scores():
  for x, (a, b) in f.items():
    print('\nThe F scores for {} are:{} '.format(x,a))
    print('\nThe pval is {}'.format(b))
    print(np.argsort(f[x]))



print_mi_scores()
print_f_scores()
```

```
(62, 16)
(62,)

The mutual_info_regression for casual are: [1.43967653e+02 3.44890586e+02 6.20046245e+0
 3.33109137e+00 2.73674228e+00 7.07292978e+01 4.73471711e+02
 4.82454311e+02 7.46194000e+00 4.24378416e+01 6.02910207e+02
 6.14850370e+03]
[ 3 10  9  6  1  5  4  0  2  7  8 12 11]

The mutual_info_regression for registered are: [1.43967653e+02 3.44890586e+02 6.2004624
 3.33109137e+00 2.73674228e+00 7.07292978e+01 4.73471711e+02
 4.82454311e+02 7.46194000e+00 4.24378416e+01 6.02910207e+02
 6.14850370e+03]
[ 3 10  9  6  5  4  0  2  1  7  8 11 12]

The mutual_info_regression for cnt are: [1.43967653e+02 3.44890586e+02 6.20046245e+01 3
 3.33109137e+00 2.73674228e+00 7.07292978e+01 4.73471711e+02
 4.82454311e+02 7.46194000e+00 4.24378416e+01 6.02910207e+02
 6.14850370e+03]
[ 3  5  4  9 10  6  0  1  2  7  8 11 12]

The F scores for casual are:[ 33.76597821  47.99907687  11.19955141   2.15375168   2.62
 267.40499289  47.50957705 305.27467615 306.19920635   4.34891601
```

```
  21.07271914          inf 134.99817315]

The pval is [9.28831269e-09 9.38283807e-12 8.60125591e-04 1.42653279e-01
 1.05487862e-01 1.94426737e-51 1.18603467e-11 2.30884219e-57
 1.66531192e-57 3.73797801e-02 5.20675306e-06 0.00000000e+00
 9.45427573e-29]
[[ 3  4  9  2 10  0  6  1 12  5  7  8 11]
 [11  8  7  5 12  1  6  0 10  2  9  4  3]]

The F scores for registered are:[1.48714225e+02 3.97966775e+02 6.87109144e+01 8.7239138
 2.40707798e+00 7.41814675e+01 5.30225014e+01 3.00098390e+02
 3.06724201e+02 6.09921653e+00 3.61808572e+01 1.34998173e+02
 1.09437471e+18]

The pval is [2.91761569e-31 5.46967558e-71 5.47310821e-16 3.24168747e-03
 1.21221638e-01 4.38157109e-17 8.56559967e-13 1.44622865e-56
 1.38348826e-57 1.37521741e-02 2.84445325e-09 9.45427573e-29
 0.00000000e+00]
[[ 4  9  3 10  6  2  5 11  0  7  8  1 12]
 [12  1  8  7  0 11  5  2  6 10  3  9  4]]

The F scores for cnt are:[1.43967653e+02 3.44890586e+02 6.20046245e+01 3.42144104e+00
 3.33109137e+00 2.73674228e+00 7.07292978e+01 4.73471711e+02
 4.82454311e+02 7.46194000e+00 4.24378416e+01 6.02910207e+02
 6.14850370e+03]

The pval is [2.13399668e-30 2.48353990e-63 1.24311178e-14 6.47593579e-02
 6.83908070e-02 9.84949616e-02 2.15097582e-16 2.81062240e-81
 1.85450413e-82 6.45414333e-03 1.35995868e-10 1.71746937e-97
 0.00000000e+00]
[[ 5  4  3  9 10  2  6  0  1  7  8 11 12]
 [12 11  8  7  1  0  6  2 10  9  3  4  5]]
/usr/local/lib/python3.7/dist-packages/sklearn/feature_selection/_univariate_selection.
  F = corr ** 2 / (1 - corr ** 2) * degrees_of_freedom
```

Datasets

Bike Dataset

Data Inspection

Report

```
day_report = ProfileReport(day_df)
print(repr(day_report.report))
```

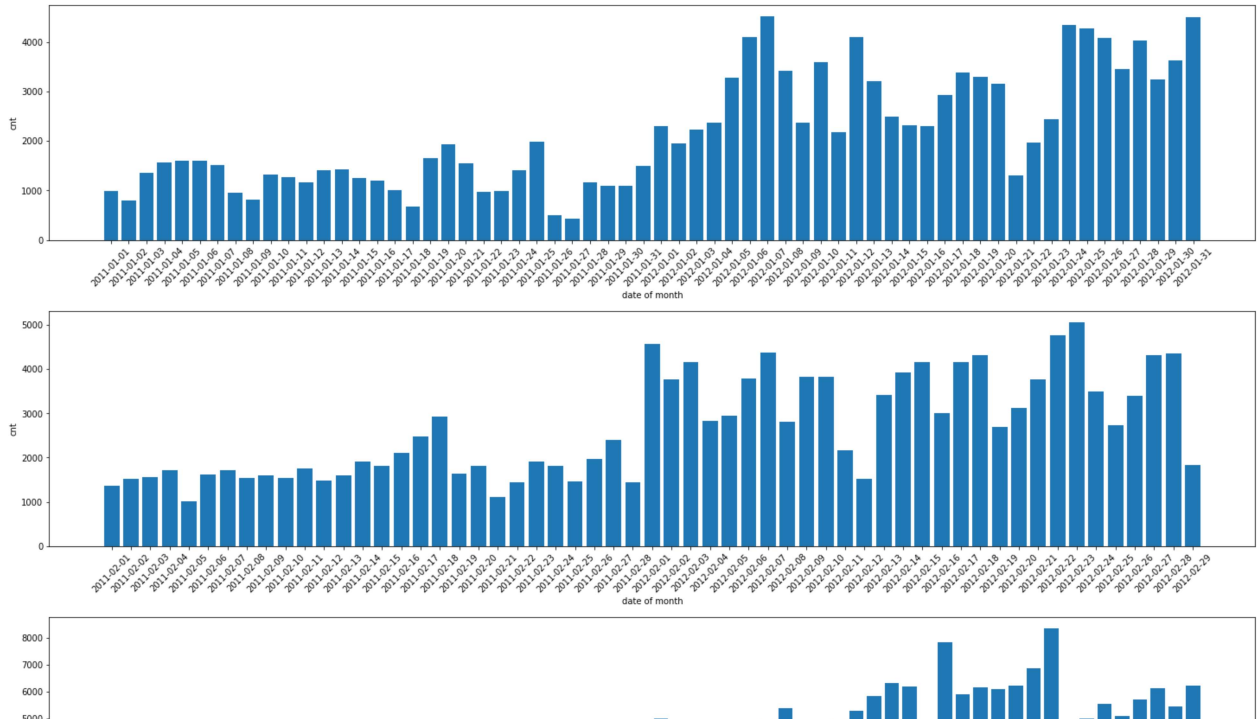
```
print(day_report.report.content)
day_report.to_notebook_iframe()
```

Summarize dataset: 29/29 [00:18<00:00, 1.37s/it

Histogram (cnt vs day of month)

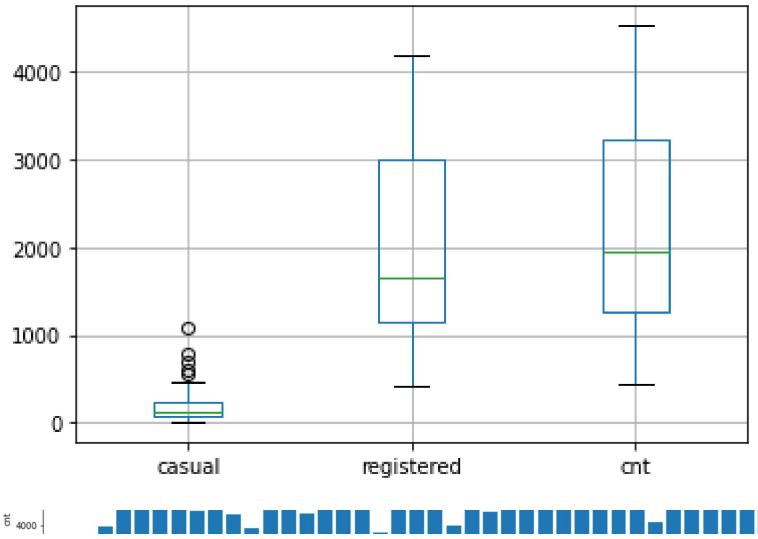
Generate report structure: 100% 1/1 [00:08<00:00, 8.40s/it]

```
'''Plotting a histogram for each month
  - using the dteday instead of work or weekday
  - y axis is cnt'''

for idx in range(1,13):
  temp = day_df[day_df['mnth']==idx]
  plt.figure(figsize=(25, 5))
  plt.bar(temp['dteday'], temp['cnt'])
  plt.xticks(rotation = 45)
  plt.xlabel('date of month')
  plt.ylabel('cnt')
  plt.show()
```

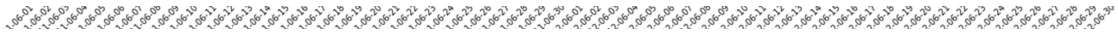


Boxplots

```
''' categorical features of target variables for biking data'''

bike_target_vars = ['casual', 'registered', 'cnt']
df = day_df[day_df['mnth']==1]

boxplot = df.boxplot(column=bike_target_vars)
```



Categorical Features + Standardization



```
day_df = pd.read_csv('/content/drive/MyDrive/4. Project 4 - Regression Analysis /day.csv')
from sklearn.feature_selection import f_regression
```

```
preprocess = make_column_transformer(
    (StandardScaler(), ['season', 'yr', 'mnth', 'holiday', 'weekday',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed']),

    remainder ='passthrough'
)


standardized_data = []
bike_cols = day_df.columns
x_bike = preprocess.fit_transform(day_df[bike_cols])
bike_clean = x_bike[:,0:11]
bike_target = x_bike[:,15]
```

Feature Selection

```
bike_col = ['season', 'yr', 'mnth', 'holiday', 'weekday',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed'
       ]
mutual_info_bike_target = mutual_info_regression(bike_clean,bike_target)

print('Mutual Information between count and features:\n')
for i in range(0, len(bike_col)):
    print(bike_col[i],': ', mutual_info_bike_target[i])

f = {}
df = day_df
f, pval = f_regression(df.iloc[:,2:13],df.loc[:,'cnt'] ) ##error with dtime

print('\n\n F Score and P-Value:\n')
for i in range(0,len(bike_col)):
    print(i,'. ', bike_col[i],', F score:',f[i], ',     P Value: ',pval[i])
```

```
 Mutual Information between count and features:

 season :  0.2145429300245203
 yr :  0.2773231994903549
 mnth :  0.37862837263313187
 holiday :  0.011512657465930554
 weekday :  0.0448600426812793
 workingday :  0.02181675183786691
 weathersit :  0.0649471208926089
 temp :  0.3887679767949783
 atemp :  0.46441733617084324
 hum :  0.04543535262581688
 windspeed :  0.05534508900120061


  F Score and P-Value:
```

```
0 .  season , F score: 143.9676525909154 ,     P Value:  2.133996684343797e-30
1 .  yr , F score: 344.8905855356845 ,     P Value:  2.4835399044501753e-63
2 .  mnth , F score: 62.00462454833198 ,     P Value:  1.2431117778656476e-14
3 .  holiday , F score: 3.421441039972211 ,     P Value:  0.064759357926115
4 .  weekday , F score: 3.331091365174518 ,     P Value:  0.06839080695470057
5 .  workingday , F score: 2.7367422831913517 ,     P Value:  0.0984949616002635
6 .  weathersit , F score: 70.72929782920825 ,     P Value:  2.150975821424902e-16
7 .  temp , F score: 473.4717105349773 ,     P Value:  2.8106223975901415e-81
8 .  atemp , F score: 482.4543105289931 ,     P Value:  1.8545041252824314e-82
9 .  hum , F score: 7.461939996345346 ,     P Value:  0.006454143325437774
10 .  windspeed , F score: 42.437841593463865 ,     P Value:  1.3599586778863725e-10
```

```
bike_feat = bike_clean[:, [0,1,2,6,7,8,10]]
```

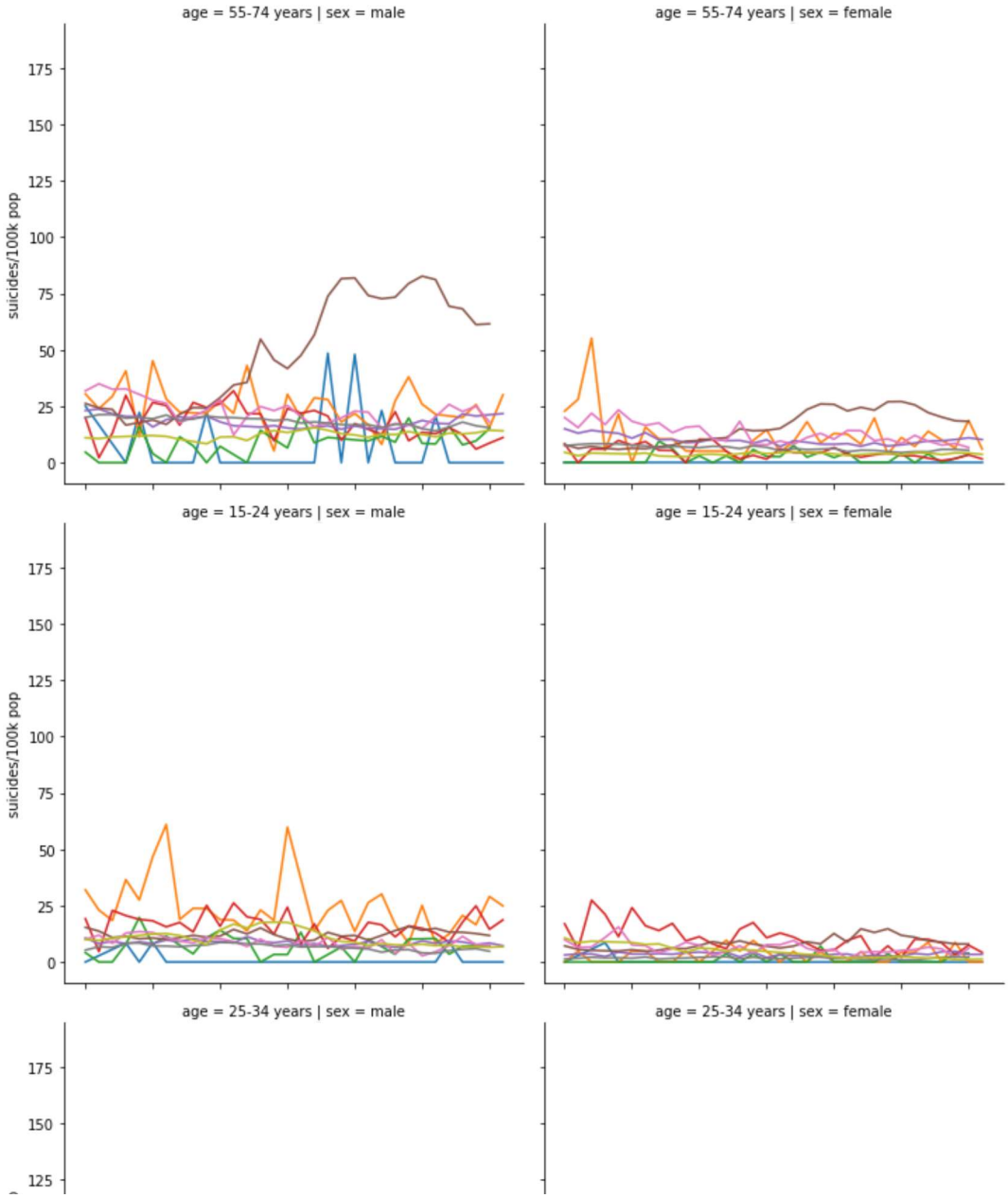
Master

Data Inspection

```
top_ten_countries =[ 'Antigua','Grenada','Iceland','Malta','Mauritius','Netherlands','Singapc
top_ten_df = master_df[master_df['country'].isin(top_ten_countries)]
```

```
sns.relplot(data=top_ten_df,  x="year",y="suicides/100k pop", kind ='line', hue ='country', r
```

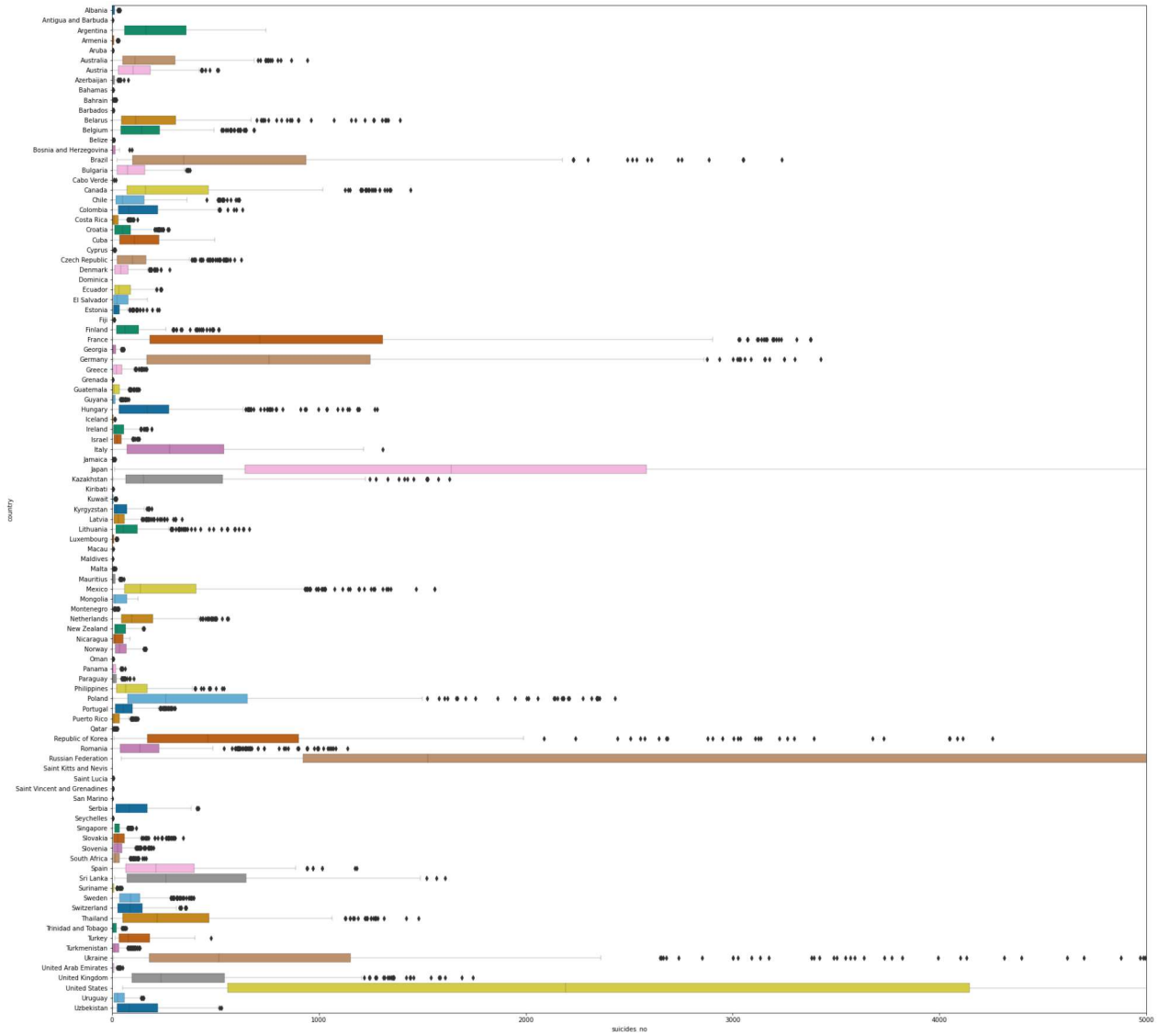
```
<seaborn.axisgrid.FacetGrid at 0x7f8ee3e7ff10>
```




```
a4_dims = (30, 30)
fig1, ax1 = plt.subplots(figsize=a4_dims)

#plt.xticks(rotation=90)


sns.boxplot(y=master_df['country'], x=master_df['suicides_no'],
                data=master_df,
                palette="colorblind", width=.9, linewidth =.3, ax = ax1
                )#, hue='year').subplots(figsize=(20, 10))
plt.xlim([0, 5000])
fig1.show()


a4_dims = (15, 5)
fig2, ax2 = plt.subplots(figsize=a4_dims)
```

```
fig2, ax2 = plt.subplots(figsize=a4_dims)

sns.boxplot(y=master_df['sex'], x=master_df['suicides_no'],
                data=master_df,
                palette="colorblind", width=.5, linewidth =.1, ax = ax2, fliersize = .5
                )
plt.xlim([0, 1000])
fig2.show()


a4_dims = (15, 5)
fig3, ax3 = plt.subplots(figsize=a4_dims)
sns.boxplot(y=master_df['generation'], x=master_df['suicides_no'],
                data=master_df,
                palette="colorblind", width=.5, linewidth =.1, ax = ax3
                )
plt.xlim([0, 1000])
fig3.show()

a4_dims = (15, 5)
fig4, ax4 = plt.subplots(figsize=a4_dims)
sns.boxplot(y=master_df['age'], x=master_df['suicides_no'],
                data=master_df,
                palette="colorblind", width=.5, linewidth =.06, ax = ax4
                )

plt.xlim([0, 1000])
```

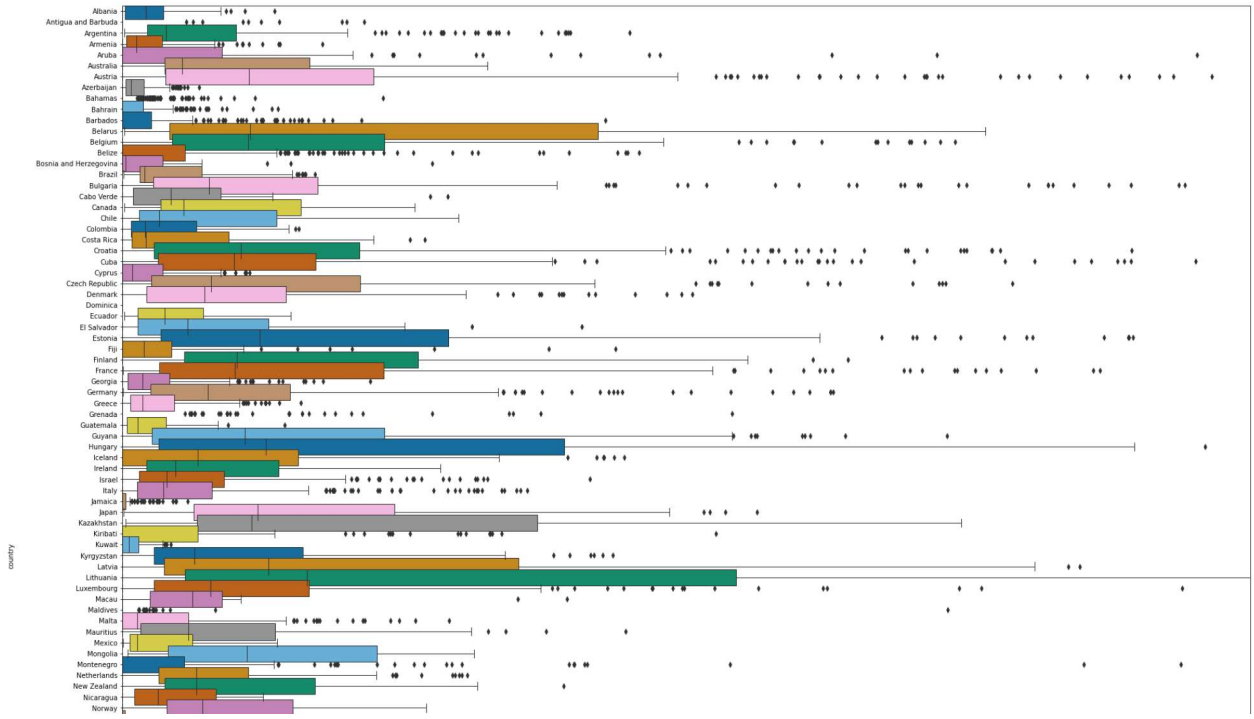
```
(0.0, 1000.0)
```




```
a4_dims = (30, 30)
fig, ax = plt.subplots(figsize=a4_dims)


sns.boxplot(y=master_df['country'], x=master_df['suicides/100k pop'],
                 data=master_df,
                 palette="colorblind", width=1.5, linewidth =1, ax = ax
                 )#, hue='year').subplots(figsize=(20, 10))
plt.xlim([0, 140])


a4_dims = (15, 5)
fig, ax = plt.subplots(figsize=a4_dims)

sns.boxplot(y=master_df['sex'], x=master_df['suicides/100k pop'],
                data=master_df,
                palette="colorblind", width=.5, linewidth =.8, ax = ax, fliersize = .5
                )
plt.xlim([0, 160])
```

```
a4_dims = (15, 5)
fig, ax = plt.subplots(figsize=a4_dims)
sns.boxplot(y=master_df['generation'], x=master_df['suicides/100k pop'],
                data=master_df,
                palette="colorblind", width=.5, linewidth =.8, ax = ax
                )
plt.xlim([0, 160])

a4_dims = (15, 5)
fig, ax = plt.subplots(figsize=a4_dims)
sns.boxplot(y=master_df['age'], x=master_df['suicides/100k pop'],
                data=master_df,
                palette="colorblind", width=.5, linewidth =.8, ax = ax
                )

plt.xlim([0, 160])
```

```
(0.0, 160.0)
```



Add continent to country


```
countries=master_df['country'].unique()

countries
sorted_countries = {}
countries = countries.tolist()
countries
countries.remove('Republic of Korea')

for x in countries:
  try:
    country_code = pc.country_name_to_country_alpha2(x, cn_name_format="default")

    continent_name = pc.country_alpha2_to_continent_code(country_code)

    sorted_countries[x] = continent_name
  except:
    pass

sorted_countries['Republic of Korea'] = 'AS'
sorted_countries['Saint Vincent and Grenadines'] = 'NA'
continent_list =[]
countries_grouped =[]
for x in sorted_countries.values():
  if x not in continent_list:
    continent_list.append(x)

EU_list=[]
NA_list =[]
```

```
NA_list =[]
SA_list =[]
AS_list =[]
OC_list =[]
AF_list = []
for x,y in sorted_countries.items():
  if sorted_countries[x] == 'EU':
    EU_list.append(x)
  if sorted_countries[x] == 'NA':
    NA_list.append(x)
  if sorted_countries[x] == 'SA':
    SA_list.append(x)
  if sorted_countries[x] == 'AS':
    AS_list.append(x)
  if sorted_countries[x] == 'OC':
    OC_list.append(x)
  if sorted_countries[x] == 'AF':
    AF_list.append(x)
all_df =pd.DataFrame({'EU':pd.Series(EU_list),
                  'NA': pd.Series(NA_list),
                  'SA': pd.Series(SA_list),
                  'AS': pd.Series(AS_list),
                  'OC': pd.Series(OC_list),
                  "AF": pd.Series(AF_list)}
                  )
sorted_count_df = master_df.copy()
sorted_count_df.insert(1, 'Continent', pd.Series() )
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:54: DeprecationWarning: Th
```

```
df= sorted_count_df


for x in AS_list:  # if this is true then place this, otherwise return the original
  sorted_count_df['Continent'] = pd.np.where(df.country.str.contains(x), "AS", sorted_count_c

for x in AF_list:
  sorted_count_df['Continent'] = pd.np.where(df.country.str.contains(x), "AF", sorted_count_c


for x in OC_list:
  sorted_count_df['Continent'] = pd.np.where(df.country.str.contains(x), "OC", sorted_count_c

for x in EU_list:
  sorted_count_df['Continent'] = pd.np.where(df.country.str.contains(x), "EU", sorted_count_c


for x in NA_list:
  sorted_count_df['Continent'] = pd.np.where(df.country.str.contains(x), "NA", sorted_count_c
```

```
for x in SA_list:
  sorted_count_df['Continent'] = pd.np.where(df.country.str.contains(x), "SA", sorted_count_d
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: FutureWarning: The pand

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: FutureWarning: The pand
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:13: FutureWarning: The pan
  del sys.path[0]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16: FutureWarning: The pan
  app.launch_new_instance()
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:20: FutureWarning: The pan
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:23: FutureWarning: The pan
```

```
print(sorted_count_df.shape)
sorted_count_df
```

```
(27820, 13)
```

	country	Continent	year	sex	age	suicides_no	population	suicides/100k pop
0	Albania	EU	1987	male	15-24 years	21	312900	6.71
1	Albania	EU	1987	male	35-54 years	16	308000	5.19
2	Albania	EU	1987	female	15-24 years	14	289700	4.83
3	Albania	EU	1987	male	75+ years	1	21800	4.59
4	Albania	EU	1987	male	25-34 years	9	274300	3.28
...	...	...	...	...	...	...	...	..
27815	Uzbekistan	AS	2014	female	35-54 years	107	3620833	2.96
27816	Uzbekistan	AS	2014	female	75+ years	9	348465	2.58
27817	Uzbekistan	AS	2014	male	5-14	60	2762158	2.17

One hot encoding for Master

```
from sklearn.feature_selection import f_regression
'''
preprocess is a 'pipeline' for preprocessing a DataFrame

StandardScaler = Standardize features by removing the mean and scaling to unit variance
OneHotEncoder = Encode categorical features as a one-hot numeric array


'''
'''the preprocess method has to be changed accordding to the dataframe. it currently has the

since it shows the one- hot encoding better'''

preprocess = make_column_transformer(
    (OneHotEncoder(sparse=False), ['Continent','sex', 'age','generation']),
    (StandardScaler(), ['population', 'gdp_per_capita ($)', 'suicides_no', 'year']),
    remainder ='passthrough'
)


master_cols = sorted_count_df.columns
x_master = preprocess.fit_transform(sorted_count_df[master_cols])
master_clean = x_master[:,0:24]
master_target_df = sorted_count_df['suicides/100k pop']
master_target = pd.DataFrame(master_target_df).to_numpy()




'''
Cont
0 - AF
1 - AS
2 - EU
3 - NA
4 - OC
5 - SA
Gender
```

```
6 - Female
7 - Male
8-13 age
8 - 15-24
10 - 35-54
11 - 5-14
12 - 55-74
13 - 75+
Generation
14 - Boomers
15 - G.I.Generation
16 - Generation X
17 - Generation Z
18 - Millenials
19 - Silent
'''



master_encoded_cols = ['Africa', 'Asia', 'Europe', 'North America', 'Oceania', 'South America
                       '15-24', '35-54', '5-14', '55-74', '75+', 'Boomers', 'G.I. Generat
                       'Population', 'GDP Per Capita', 'Number of Suicides', 'Year']
mutual_info_master_target = mutual_info_regression(master_clean,master_target)

print('Mutual Information between suicides/100k and features:\n')
for i in range(0, len(master_encoded_cols)):
    print(master_encoded_cols[i],': ', mutual_info_master_target[i])

f = {}
df = master_df
f, pval = f_regression(master_clean,df.loc[:,'suicides/100k pop'] ) ##error with dtime

print('\n\n F Score and P-Value:\n')
for i in range(0,len(master_encoded_cols)):
    print(i,'. ', master_encoded_cols[i],', F score:',f[i], ',     P Value: ',pval[i])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionW
  y = column_or_1d(y, warn=True)
Mutual Information between suicides/100k and features:

Africa :  0.0023815704666181325
Asia :  0.012675724501414631
Europe :  0.038959598962064135
North America :  0.042854447601373025
Oceania :  0.006334712836827627
South America :  0.015009826212970534
Female :  0.1387707243514682
Male :  0.13163179463197627
15-24 :  0.029083172499539955
35-54 :  0.021102726884357992
5-14 :  0.009944633787011714
55-74 :  0.21990054834108763
75+ :  0.025020730528599433
Boomers :  0.03421032526984957
```

```
G.I. Generation :  0.019819216891598934
Generation X :  0.024277017676901824
Generation Z :  0.006626400155068435
Millenials :  0.058536975862405694
Silent :  0.047273948231584484
Population :  0.031995299489298024
GDP Per Capita :  0.5858516970421563
Number of Suicides :  0.18915559863974973
Year :  0.7139550374254116


 F Score and P-Value:

0 .  Africa , F score: 66.93963431264717 ,     P Value:  2.9178246134584683e-16
1 .  Asia , F score: 66.18741307664381 ,     P Value:  4.2696414590000695e-16
2 .  Europe , F score: 1222.6471815325954 ,     P Value:  3.481432312444257e-262
3 .  North America , F score: 656.456248139557 ,     P Value:  4.036575288863593e-143
4 .  Oceania , F score: 4.407559194330083 ,     P Value:  0.0357889872516574
5 .  South America , F score: 19.069537289838756 ,     P Value:  1.2649570815338951e-05
6 .  Female , F score: 5035.427899106122 ,     P Value:  0.0
7 .  Male , F score: 5035.427899106153 ,     P Value:  0.0
8 .  15-24 , F score: 233.90850746820163 ,     P Value:  1.3713970108595773e-52
9 .  35-54 , F score: 6.136421882676922 ,     P Value:  0.013248396840543672
10 .  5-14 , F score: 70.57193326529135 ,     P Value:  4.646500819275748e-17
11 .  55-74 , F score: 2490.576486714397 ,     P Value:  0.0
12 .  75+ , F score: 173.89594422337152 ,     P Value:  1.3714946624158368e-39
13 .  Boomers , F score: 2065.611860765473 ,     P Value:  0.0
14 .  G.I. Generation , F score: 62.875561990211516 ,     P Value:  2.2839457691318597e
15 .  Generation X , F score: 1089.9982012836413 ,     P Value:  1.6616024074534953e-23
16 .  Generation Z , F score: 118.69386675648262 ,     P Value:  1.3894620205580866e-27
17 .  Millenials , F score: 654.7693474710669 ,     P Value:  9.213587345745627e-143
18 .  Silent , F score: 1185.0732848023233 ,     P Value:  2.3381110030483988e-254
19 .  Population , F score: 739.5625042761219 ,     P Value:  9.468085850175482e-161
20 .  GDP Per Capita , F score: 1.909580251300788 ,     P Value:  0.167020989064028
21 .  Number of Suicides , F score: 0.08864797749177898 ,     P Value:  0.7659052836271
22 .  Year , F score: 2886.4074133700137 ,     P Value:  0.0
```

```
master_feat = master_clean[:,(2,3,6,7,8,11,13,15,17,18,19,22)]
```

3.2.1 Linear Regression

```
preprocessed_data = preprocessed_data[:, 0:23]
targets = sorted_count_df['suicides/100k pop']
print(targets.shape)
preprocessed_data.shape
```

```
(27820,)
(27820, 23)
```

3.2.1 Linear Regression

```
def rmse_cal(algo, data, target_var):
    X_train = []
    X_test = []
    y_train = []
    y_test = []
    kf = KFold(n_splits=10)
    test_rmse = 0
    train_rmse = 0
    for trainset, testset in kf.split(data,target_var):

        X_train = data[trainset,:]
        y_train = target_var[trainset]
        X_test = data[testset,:]
        y_test = target_var[testset]
        y_train = y_train.astype('int')
        algo.fit(X_train, y_train)
        test_predictions = algo.predict(X_test)
        train_predictions = algo.predict(X_train)
        test_rmse = test_rmse + mean_squared_error(y_test, test_predictions, squared=False)
        train_rmse = train_rmse + mean_squared_error(y_train, train_predictions, squared=Fals
    test_rmse = (test_rmse/10)
    train_rmse = (train_rmse/10)

    return train_rmse, test_rmse



X = preprocessed_data
y = targets
X_train = []
X_test = []
y_train = []
y_test = []

train_rmse=[]
test_rmse=[]



def rmse_cal(algo, data, target_var):
    X_train = []
    X_test = []
    y_train = []
    y_test = []
    kf = KFold(n_splits=10)
    test_rmse = 0
    train_rmse = 0
    for trainset, testset in kf.split(data,target_var):

        X_train = data[trainset,:]
        y_train = target_var[trainset]
```

```
            y_train = target_var[trainset]
            X_test = data[testset,:]
            y_test = target_var[testset]
            y_train = y_train.astype('int')
            algo.fit(X_train, y_train)
            test_predictions = algo.predict(X_test)
            train_predictions = algo.predict(X_train)
            test_rmse = test_rmse + mean_squared_error(y_test, test_predictions, squared=False)
            train_rmse = train_rmse + mean_squared_error(y_train, train_predictions, squared=Fals
        test_rmse = (test_rmse/10)
        train_rmse = (train_rmse/10)

        return train_rmse, test_rmse
```

```
def regression(processed_data, targets):

    algo = LinearRegression(normalize=True)

    #https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_predic
    predicted = cross_val_predict(algo,master_clean, targets, cv=10)
    y = targets

    lr_fig, ax = plt.subplots()
    ax.scatter(y, predicted, edgecolors=(0, 0, 0))
    ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
    ax.set_xlabel('Measured - using ordinary least squares regression')
    ax.set_ylabel('Predicted  - using ordinary least squares regression')
    plt.show()

    algo = Lasso()
    predicted = cross_val_predict(algo, master_clean, targets, cv=10)
    y = targets
    lasso_fig, ax = plt.subplots()
    ax.scatter(y, predicted, edgecolors=(0, 0, 0))
    ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
    ax.set_xlabel('Measured - using Lasso regression')
    ax.set_ylabel('Predicted - using Lasso regression')
    plt.show()


    algo = Ridge()
    predicted = cross_val_predict(algo, master_clean, targets, cv=10)
    y = targets
    lasso_fig, ax = plt.subplots()
    ax.scatter(y, predicted, edgecolors=(0, 0, 0))
    ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
    ax.set_xlabel('Measured - using Ridge regression')
```

```
  ax.set_ylabel('Predicted - using Ridge regression')
  plt.show()
```

```
regression(master_clean, targets)

lasso_rmse = rmse_cal(Lasso(), master_clean, targets)
linear_rmse = rmse_cal(LinearRegression(normalize=True), master_clean, targets)
ridge_rmse = ridge_rmse = rmse_cal(Ridge(), master_clean, targets)

d={}
d['Lasso Average RMSE'] = lasso_rmse
d['Ridge RMSE'] = ridge_rmse
d['Linear RMSE'] = linear_rmse
rmse_df = pd.DataFrame(d, index = ['Suicide Dataset: Train', 'Suicide Dataset:  Test' ])
rmse_df
```

3.2.2 Polynomial Regression

```
features_test = ['population', 'year', 'suicides_no', 'gdp_per_capita ($)']
#features that will be sent to linear regression function after increasing up to a degree

y = master_df['suicides/100k pop']
 # target variable

dlr = {}  #containers
dlasso = {}
dridge = {}

x= master_df[features_test].values # the values of the features only


upper_bound = 5

## getting values over all

for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform(x)
  train, test = rmse_cal(Lasso(alpha = 0.005), x_new, y)
  dlr[deg] = [ train, test]



for deg in range(1, upper_bound):
```

```
  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform (x)
  train, test = rmse_cal(LinearRegression(), x_new, y)
  dlasso[deg] = [ train, test]


for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform (x)
  train, test = rmse_cal(Ridge(), x_new, y)
  dridge[deg] = [ train, test]




lr_df = pd.DataFrame.from_dict(dlasso)
lr_df.index = ['LR train', 'LR test']
lr_df = lr_df.add_prefix('degree_')

lasso_df = pd.DataFrame.from_dict(dlr)
lasso_df.index = ['Lasso train', 'Lasso test']
lasso_df = lasso_df.add_prefix('degree_')

ridge_df =pd.DataFrame.from_dict(dridge)
ridge_df.index = ['Ridge train', 'Ridge test']
ridge_df = ridge_df.add_prefix('degree_')

pd.concat([lr_df, ridge_df, lasso_df])
```

	degree_1	degree_2	degree_3	degree_4
LR train	17.429160	16.239013	14.979989	17.267470
LR test	17.472831	18.701238	37.068653	126.440270
Ridge train	17.429160	16.235515	14.919723	15.009597
Ridge test	17.472831	18.701747	40.076331	182.017151
Lasso train	17.429160	16.269587	15.048106	14.515364
Lasso test	17.472839	18.488901	30.651420	42.655445

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.neural_network import MLPRegressor
'''
hidden layer sizes lets you choose how many layers and how many nodes per layer
weight decay  is inversely used to decrease the learning rate per iteration
```

```
'''
def neural_network(data, target_var, hidden_layer_sizes, weight_decay):
    X_train = []
    X_test = []
    y_train = []
    y_test = []
    kf = KFold(n_splits=10)
    test_rmse = 0
    train_rmse = 0
    for trainset, testset in kf.split(data,target_var):
        X_train = data[trainset,:]
        y_train = target_var[trainset]
        X_test = data[testset,:]
        y_test = target_var[testset]
        y_train = y_train.astype('int')
        nn = MLPRegressor(hidden_layer_sizes = hidden_layer_sizes, alpha = weight_decay, acti
        test_predictions = nn.predict(X_test)
        train_predictions = nn.predict(X_train)
        test_rmse = test_rmse + mean_squared_error(y_test, test_predictions, squared=False)
        train_rmse = train_rmse + mean_squared_error(y_train, train_predictions, squared=Fals
    test_rmse = test_rmse/10
    train_rmse = train_rmse/10

    return train_rmse, test_rmse

train, test = neural_network(preprocessed_data, targets, (100,50), 0.99)
print('Average Suicide Dataset train RMSE: ', train,'\nAverage Suicide Dataset test RMSE: ',


decay_values = [.1,.2,.3,.4,.5,.6,.7,.8,.9]
train = np.zeros(len(decay_values))
test = np.zeros(len(decay_values))
i = 0
for val in decay_values:
    train[i], test[i] = neural_network(preprocessed_data, targets, (256, 128, 64, 32, 16), va
    i = i + 1


for i in range(0, i+2):

    print('Decay Value: ',decay_values[i], '\nTrain RMSE: ', train[i],'\nTest RMSE: ', test[i


temp = {}

for val in range(len(decay_values)):
    temp[decay_values[val]] = [train[val], test[val]]




nn_df = pd.DataFrame.from_dict(temp )
nn_df.index = index=  [ 'Train RMSE', 'Test RMSE']
```

```
nn_df.add_prefix('Weight_decay_')
```

Video Transcoding

[ ] ↳ *14 cells hidden*

Classifiers

Linear Regression

Bike Dataset

```
lasso_rmse = rmse_cal(Lasso(), bike_clean, bike_target)
linear_rmse = rmse_cal(LinearRegression(normalize=True), bike_clean, bike_target)
ridge_rmse = ridge_rmse = rmse_cal(Ridge(), bike_clean, bike_target)

mc={}
mc['Lasso Average RMSE'] = lasso_rmse
mc['Ridge RMSE'] = ridge_rmse
mc['Linear RMSE'] = linear_rmse
mc_rmse_df = pd.DataFrame(mc, index = ['Bike Dataset: Train', 'Bike Dataset:  Test' ])
mc_rmse_df
```

	Lasso Average RMSE	Ridge RMSE	Linear RMSE
Bike Dataset: Train	858.430699	858.427857	858.399295
Bike Dataset: Test	965.636596	965.600245	968.531165

```
lasso_rmse = rmse_cal(Lasso(), bike_feat, bike_target)
linear_rmse = rmse_cal(LinearRegression(normalize=True), bike_feat, bike_target)
ridge_rmse = ridge_rmse = rmse_cal(Ridge(), bike_feat, bike_target)

mc={}
mc['Lasso Average RMSE'] = lasso_rmse
mc['Ridge RMSE'] = ridge_rmse
mc['Linear RMSE'] = linear_rmse
mc_rmse_df = pd.DataFrame(mc, index = ['After FE Bike Dataset: Train', 'After FE Bike Dataset
```

```
mc_rmse_df
```

	Lasso Average RMSE	Ridge RMSE	Linear RMSE
After FE Bike Dataset: Train	887.270500	887.278654	887.259271
After FE Bike Dataset: Test	979.450565	978.571129	980.962990

Master Dataset

```
master_target_df = sorted_count_df['suicides/100k pop']
lasso_rmse = rmse_cal(Lasso(), master_clean, master_target_df)
linear_rmse = rmse_cal(LinearRegression(normalize=True), master_clean, master_target_df)
ridge_rmse = ridge_rmse = rmse_cal(Ridge(), master_clean, master_target_df)

m={}
m['Lasso Average RMSE'] = lasso_rmse
m['Ridge RMSE'] = ridge_rmse
m['Linear RMSE'] = linear_rmse
m_rmse_df = pd.DataFrame(m, index = ['Suicide Dataset: Train', 'Suicide Dataset:  Test' ])
m_rmse_df
```

	Lasso Average RMSE	Ridge RMSE	Linear RMSE
Suicide Dataset: Train	15.519176	14.538254	14.541159
Suicide Dataset: Test	15.413893	14.716115	14.700015

```
master_target_df = sorted_count_df['suicides/100k pop']
lasso_rmse = rmse_cal(Lasso(), master_feat, master_target_df)
linear_rmse = rmse_cal(LinearRegression(normalize=True), master_feat, master_target_df)
ridge_rmse = ridge_rmse = rmse_cal(Ridge(), master_feat, master_target_df)

mc={}
mc['Lasso Average RMSE'] = lasso_rmse
mc['Ridge RMSE'] = ridge_rmse
mc['Linear RMSE'] = linear_rmse
mc_rmse_df = pd.DataFrame(mc, index = ['After FE Suicide Dataset: Train', 'After FE Suicide D
mc_rmse_df
```

	Lasso Average RMSE	Ridge RMSE	Linear RMSE
After FE Suicide Dataset: Train	15.634447	14.770508	14.770498
After FE Suicide Dataset: Test	15.537173	14.759724	14.759677

Transcoding Dataset

```
targets = transcoding_df['utime']

lasso_rmse = rmse_cal(Lasso(), transcoding_clean, targets)
linear_rmse = rmse_cal(LinearRegression(normalize=True), transcoding_clean, targets)
ridge_rmse = ridge_rmse = rmse_cal(Ridge(), transcoding_clean, targets)

mc={}
mc['Lasso Average RMSE'] = lasso_rmse
mc['Ridge RMSE'] = ridge_rmse
mc['Linear RMSE'] = linear_rmse
mc_rmse_df = pd.DataFrame(mc, index = ['Transcoding Dataset: Train', 'Transcoding Dataset:  T
mc_rmse_df
```

	Lasso Average RMSE	Ridge RMSE	Linear RMSE
Transcoding Dataset: Train	12.607308	12.368160	1.236795e+01
Transcoding Dataset: Test	12.596144	12.405582	1.406356e+09

```
targets = transcoding_df['utime']

lasso_rmse = rmse_cal(Lasso(), transcoding_feat, targets)
linear_rmse = rmse_cal(LinearRegression(normalize=True), transcoding_feat, targets)
ridge_rmse = ridge_rmse = rmse_cal(Ridge(), transcoding_feat, targets)

mc={}
mc['Lasso Average RMSE'] = lasso_rmse
mc['Ridge RMSE'] = ridge_rmse
mc['Linear RMSE'] = linear_rmse
mc_rmse_df = pd.DataFrame(mc, index = ['After FE Transcoding Dataset: Train', 'After FE Trans
mc_rmse_df
```

	Lasso Average RMSE	Ridge RMSE	Linear RMSE
After FE Transcoding Dataset: Train	12.607308	12.377997	12.377997
After FE Transcoding Dataset: Test	12.596144	12.398913	12.398937

Polynomial Regression

Bike Dataset

```
#features that will be sent to linear regression function after increasing up to a degree

y = bike_target
 # target variable

dlr = {}  #containers
dlasso = {}
dridge = {}

x= bike_clean # the values of the features only


upper_bound = 5

## getting values over all

for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform(x)
  train, test = rmse_cal(Lasso(alpha = 0.005), x_new, y)
  dlr[deg] = [ train, test]



for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform (x)
  train, test = rmse_cal(LinearRegression(), x_new, y)
  dlasso[deg] = [ train, test]


for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform (x)
  train, test = rmse_cal(Ridge(), x_new, y)
  dridge[deg] = [ train, test]


lr_df = pd.DataFrame.from_dict(dlasso)
lr_df.index = ['LR train', 'LR test']
lr_df = lr_df.add_prefix('degree_')

lasso_df = pd.DataFrame.from_dict(dlr)
lasso_df.index = ['Lasso train', 'Lasso test']
lasso_df = lasso_df.add_prefix('degree_')
```

```
ridge_df =pd.DataFrame.from_dict(dridge)
ridge_df.index = ['Ridge train', 'Ridge test']
ridge_df = ridge_df.add_prefix('degree_')
```

```
pd.concat([lr_df, ridge_df, lasso_df])
```

	degree_1	degree_2	degree_3	degree_4
LR train	858.399295	599.096683	8.160701e+02	2.258986e-10
LR test	968.531165	1154.735314	1.297224e+14	2.894874e+04
Ridge train	858.427857	607.435816	3.962648e+02	1.955076e+02
Ridge test	965.600245	957.521685	1.096889e+03	3.007559e+03
Lasso train	858.399306	599.884963	3.906269e+02	1.930105e+02
Lasso test	968.484048	986.733945	1.163403e+03	2.702554e+03

```
#features that will be sent to linear regression function after increasing up to a degree

y = bike_target
 # target variable

dlr = {}  #containers
dlasso = {}
dridge = {}

x= bike_feat # the values of the features only


upper_bound = 5

## getting values over all

for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform(x)
  train, test = rmse_cal(Lasso(alpha = 0.005), x_new, y)
  dlr[deg] = [ train, test]



for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
```

```
    x_new = poly.fit_transform (x)
    train, test = rmse_cal(LinearRegression(), x_new, y)
    dlasso[deg] = [ train, test]


  for deg in range(1, upper_bound):

    poly = PolynomialFeatures(degree = deg)
    x_new = poly.fit_transform (x)
    train, test = rmse_cal(Ridge(), x_new, y)
    dridge[deg] = [ train, test]


  lr_df = pd.DataFrame.from_dict(dlasso)
  lr_df.index = ['LR train', 'LR test']
  lr_df = lr_df.add_prefix('degree_')

  lasso_df = pd.DataFrame.from_dict(dlr)
  lasso_df.index = ['Lasso train', 'Lasso test']
  lasso_df = lasso_df.add_prefix('degree_')

  ridge_df =pd.DataFrame.from_dict(dridge)
  ridge_df.index = ['Ridge train', 'Ridge test']
  ridge_df = ridge_df.add_prefix('degree_')



  print('After FE:')
  pd.concat([lr_df, ridge_df, lasso_df])
```

```
After FE:
```

	degree_1	degree_2	degree_3	degree_4
LR train	887.259271	856.847224	591.406950	5.080521e+02
LR test	980.962990	1560.723829	2605.493038	5.231955e+13
Ridge train	887.278654	709.026079	581.127033	4.994182e+02
Ridge test	978.571129	982.641457	987.625549	1.725515e+03
Lasso train	887.259279	705.577900	577.340842	4.953924e+02
Lasso test	980.929545	1016.780005	1126.255455	2.090671e+03

Master Dataset

```
  #features_test = ['population', 'year', 'suicides_no', 'gdp_per_capita ($)']
  #features that will be sent to linear regression function after increasing up to a degree

  y = master_df['suicides/100k pop']
```

```
 # target variable

dlr = {}  #containers
dlasso = {}
dridge = {}

x= master_clean # the values of the features only


upper_bound = 5

## getting values over all

for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform(x)
  train, test = rmse_cal(Lasso(alpha = 0.005), x_new, y)
  dlr[deg] = [ train, test]



for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform (x)
  train, test = rmse_cal(LinearRegression(), x_new, y)
  dlasso[deg] = [ train, test]


for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform (x)
  train, test = rmse_cal(Ridge(), x_new, y)
  dridge[deg] = [ train, test]



lr_df = pd.DataFrame.from_dict(dlasso)
lr_df.index = ['LR train', 'LR test']
lr_df = lr_df.add_prefix('degree_')

lasso_df = pd.DataFrame.from_dict(dlr)
lasso_df.index = ['Lasso train', 'Lasso test']
lasso_df = lasso_df.add_prefix('degree_')

ridge_df =pd.DataFrame.from_dict(dridge)
ridge_df.index = ['Ridge train', 'Ridge test']
ridge_df = ridge_df.add_prefix('degree_')

pd.concat([lr_df, ridge_df, lasso_df])
```

```
y = master_df['suicides/100k pop']
 # target variable

dlr = {}  #containers
dlasso = {}
dridge = {}

x= master_feat # the values of the features only


upper_bound = 5

## getting values over all

for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform(x)
  train, test = rmse_cal(Lasso(alpha = 0.005), x_new, y)
  dlr[deg] = [ train, test]



for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform (x)
  train, test = rmse_cal(LinearRegression(), x_new, y)
  dlasso[deg] = [ train, test]


for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform (x)
  train, test = rmse_cal(Ridge(), x_new, y)
  dridge[deg] = [ train, test]




print('After FE:')
md_f = pd.concat([lr_df, ridge_df, lasso_df])
md_f
```

After FE:

	degree_1	degree_2	degree_3	degree_4
LR train	12.377997	9.988083	7.100196e+00	3.805100e+00
LR test	12.398937	11.429237	6.238932e+10	5.470097e+10
Ridge train	12.377997	9.994616	7.127266e+00	3.919100e+00
Ridge test	12.398913	11.337629	1.088818e+02	2.648555e+03
Lasso train	12.378660	10.017657	7.208552e+00	4.205603e+00

Transcoding Dataset

```
#features that will be sent to linear regression function after increasing up to a degree

y = transcoding_df['utime']
 # target variable

dlr = {}  #containers
dlasso = {}
dridge = {}

x= transcoding_clean # the values of the features only


upper_bound = 4

## getting values over all

for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform(x)
  train, test = rmse_cal(Lasso(alpha = 0.005), x_new, y)
  dlr[deg] = [ train, test]



for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform (x)
  train, test = rmse_cal(LinearRegression(), x_new, y)
  dlasso[deg] = [ train, test]


for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
```

```
    x_new = poly.fit_transform (x)
    train, test = rmse_cal(Ridge(), x_new, y)
    dridge[deg] = [ train, test]


lr_df = pd.DataFrame.from_dict(dlasso)
lr_df.index = ['LR train', 'LR test']
lr_df = lr_df.add_prefix('degree_')

lasso_df = pd.DataFrame.from_dict(dlr)
lasso_df.index = ['Lasso train', 'Lasso test']
lasso_df = lasso_df.add_prefix('degree_')

ridge_df =pd.DataFrame.from_dict(dridge)
ridge_df.index = ['Ridge train', 'Ridge test']
ridge_df = ridge_df.add_prefix('degree_')



print('Before FE:')
td = pd.concat([lr_df, ridge_df, lasso_df])
td
```

Before FE:

	degree_1	degree_2	degree_3
LR train	1.236795e+01	9.931816e+00	6.916923e+00
LR test	5.529480e+08	9.583534e+12	1.205311e+13
Ridge train	1.236816e+01	9.925467e+00	6.944915e+00
Ridge test	1.240558e+01	1.561733e+01	1.387846e+03
Lasso train	1.236891e+01	9.954183e+00	7.064330e+00
Lasso test	1.240315e+01	1.133921e+01	9.341490e+01

```
#features that will be sent to linear regression function after increasing up to a degree

y = transcoding_df['utime']
 # target variable

dlr = {}  #containers
dlasso = {}
dridge = {}

x= transcoding_feat # the values of the features only


upper_bound = 5
```

```
## getting values over all

for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform(x)
  train, test = rmse_cal(Lasso(alpha = 0.005), x_new, y)
  dlr[deg] = [ train, test]



for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform (x)
  train, test = rmse_cal(LinearRegression(), x_new, y)
  dlasso[deg] = [ train, test]


for deg in range(1, upper_bound):

  poly = PolynomialFeatures(degree = deg)
  x_new = poly.fit_transform (x)
  train, test = rmse_cal(Ridge(), x_new, y)
  dridge[deg] = [ train, test]


lr_df = pd.DataFrame.from_dict(dlasso)
lr_df.index = ['LR train', 'LR test']
lr_df = lr_df.add_prefix('degree_')

lasso_df = pd.DataFrame.from_dict(dlr)
lasso_df.index = ['Lasso train', 'Lasso test']
lasso_df = lasso_df.add_prefix('degree_')

ridge_df =pd.DataFrame.from_dict(dridge)
ridge_df.index = ['Ridge train', 'Ridge test']
ridge_df = ridge_df.add_prefix('degree_')


print('After FE:')
td_f = pd.concat([lr_df, ridge_df, lasso_df])
td_f
```

After FE:

	degree_1	degree_2	degree_3	degree_4
LR train	12.377997	9.988083	7.100196e+00	3.805100e+00
LR test	12.398937	11.429237	6.238932e+10	5.470097e+10
Ridge test	12.398913	11.337629	1.088818e+02	2.648555e+03

Transcoding Adjusted Dataset

```
x = transcoding_adjusted
y = transcoding_target

tic = time()
poly = PolynomialFeatures(degree = 2)
x_new = poly.fit_transform (x)
train, test = rmse_cal(Ridge(), x_new, y)
t = time() - tic


print('\nTime Taken: ',t, '\nTrain RMSE: ', train,'\nTest RMSE: ', test, '\n\n')
```

```
Time Taken:  3.104208469390869
Train RMSE:  9.997433195547876
Test RMSE:  18.80638714326085
```

Neural Network

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.neural_network import MLPRegressor
from time import time
'''
hidden layer sizes lets you choose how many layers and how many nodes per layer
weight decay  is inversely used to decrease the learning rate per iteration
'''
def neural_network(data, target_var, layers):
    X_train = []
    X_test = []
    y_train = []
    y_test = []
    kf = KFold(n_splits=10)
    test_rmse = 0
    train_rmse = 0
    for trainset, testset in kf.split(data,target_var):
        X_train = data[trainset,:]
```

```
            y_train = target_var[trainset]
            X_test = data[testset,:]
            y_test = target_var[testset]
            y_train = y_train.astype('int')
            nn = MLPRegressor(hidden_layer_sizes = layers , alpha = 0.01, activation = 'relu').fi
            test_predictions = nn.predict(X_test)
            train_predictions = nn.predict(X_train)
            test_rmse = test_rmse + mean_squared_error(y_test, test_predictions, squared=False)
            train_rmse = train_rmse + mean_squared_error(y_train, train_predictions, squared=Fals
        test_rmse = test_rmse/10
        train_rmse = train_rmse/10

        return train_rmse, test_rmse


    #gridsearch to help find best params

    from sklearn.neural_network import MLPClassifier
    from sklearn.model_selection import KFold
    from sklearn.metrics import mean_squared_error
    from sklearn.neural_network import MLPRegressor
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.model_selection import GridSearchCV
    from sklearn.pipeline import Pipeline
    from sklearn.metrics import make_scorer



    layers = [(100,100,100,100,100),(200,200,200,200), (300,300,300), (400,400), (500)]
    decay_values = [.1,.01,.001,.0001]

    pipeline = Pipeline([
        ('nn', MLPRegressor()),
    ],
    )

    param_grid = [
        {
        'nn__hidden_layer_sizes': layers,
        'nn__alpha': decay_values, # 2 choices
        }
        ]

    grid = GridSearchCV(pipeline, cv=10, n_jobs=1, param_grid=param_grid, scoring='neg_root_mean_
    grid.fit(bike_clean, bike_target)

    results_df = pd.DataFrame.from_dict(grid.cv_results_)
    print(grid.best_params_)
```

Bike Dataset

```
tic = time()
train, test = neural_network(bike_clean, bike_target, (200, 200, 200, 200, 200, 200, 200, 20
t_time = time() - tic

print('\n\nAverage bike train RMSE: ', train,'\nAverage bike test RMSE: ', test,'\nTime: ',t_
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)


Average bike train RMSE:  297.2577666492729
Average bike test RMSE:  760.7967317781016
Time:  187.600337266922
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
```

```
tic = time()
train, test = neural_network(bike_feat, bike_target, (200, 200, 200, 200, 200, 200, 200, 200,
t_time = time() - tic

print('\n\nAverage bike feature selected train RMSE: ', train,'\nAverage bike feature selecte
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)


Average bike feature selected train RMSE:  516.4072166393479
Average bike feature selected test RMSE:  892.5728379852919
Time:  213.3823537826538
```

Master Datset

```
tic = time()
train, test = neural_network(master_clean, master_target, (200, 200, 200, 200, 200, 200, 200,
t_time = time() - tic

print('\n\nAverage suicide train RMSE: ', train,'\nAverage suicide test RMSE: ', test,'\nTime
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
```

```
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)


Average suicide train RMSE:  5.417268081741748
Average suicide test RMSE:  7.894900457549378
Time:  4147.508615970612
```

```
tic = time()
train, test = neural_network(master_feat, master_target, (200, 200, 200, 200, 200, 200, 200,
t_time = time() - tic
# For feature selected dataset
print('\n\nAverage suicide train RMSE: ', train,'\nAverage suicide test RMSE: ', test,'\nTime
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  y = column_or_1d(y, warn=True)


Average suicide train RMSE:  11.67555418093607
Average suicide test RMSE:  14.654518833047101
Time:  4597.0768394470215
```

Transcoding Dataset

```
tic = time()
train, test = neural_network(transcoding_clean, transcoding_target, (200, 200, 200, 200, 200,
t_time = time() - tic

print('Average video transcoding train RMSE: ', train,'\nAverage video transcoding test RMSE:
```

```
Average transcoding train RMSE:  1.6378864949975973
Average transcoding test RMSE:  4.7588058375402955
Time:  4860.205420017242
```

```
tic = time()
train, test = neural_network(transcoding_feat, transcoding_target, (200, 200, 200, 200, 200,
t_time = time() - tic
# with smaller feat data set
print('\n\nAverage transcoding train RMSE: ', train,'\nAverage transcoding test RMSE: ', test
```

```
Average transcoding train RMSE:  4.446821610063808
Average transcoding test RMSE:  5.612084659733318
Time:  4478.126942873001
```

Random Forest

Bike Dataset

```
rf = RandomForestRegressor(n_estimators = 2000, random_state = 42, max_depth=4)

train_val, test_val = rmse_cal(rf, bike_clean, bike_target)
```

```
temp ={}
temp['Train RMSE'] = train_val
temp['Test RMSE'] = test_val

results = pd.DataFrame.from_dict(temp,  orient='index')
results.columns =['RandomForest Results']

results
```

	RandomForest Results
Train RMSE	676.234994
Test RMSE	951.956127

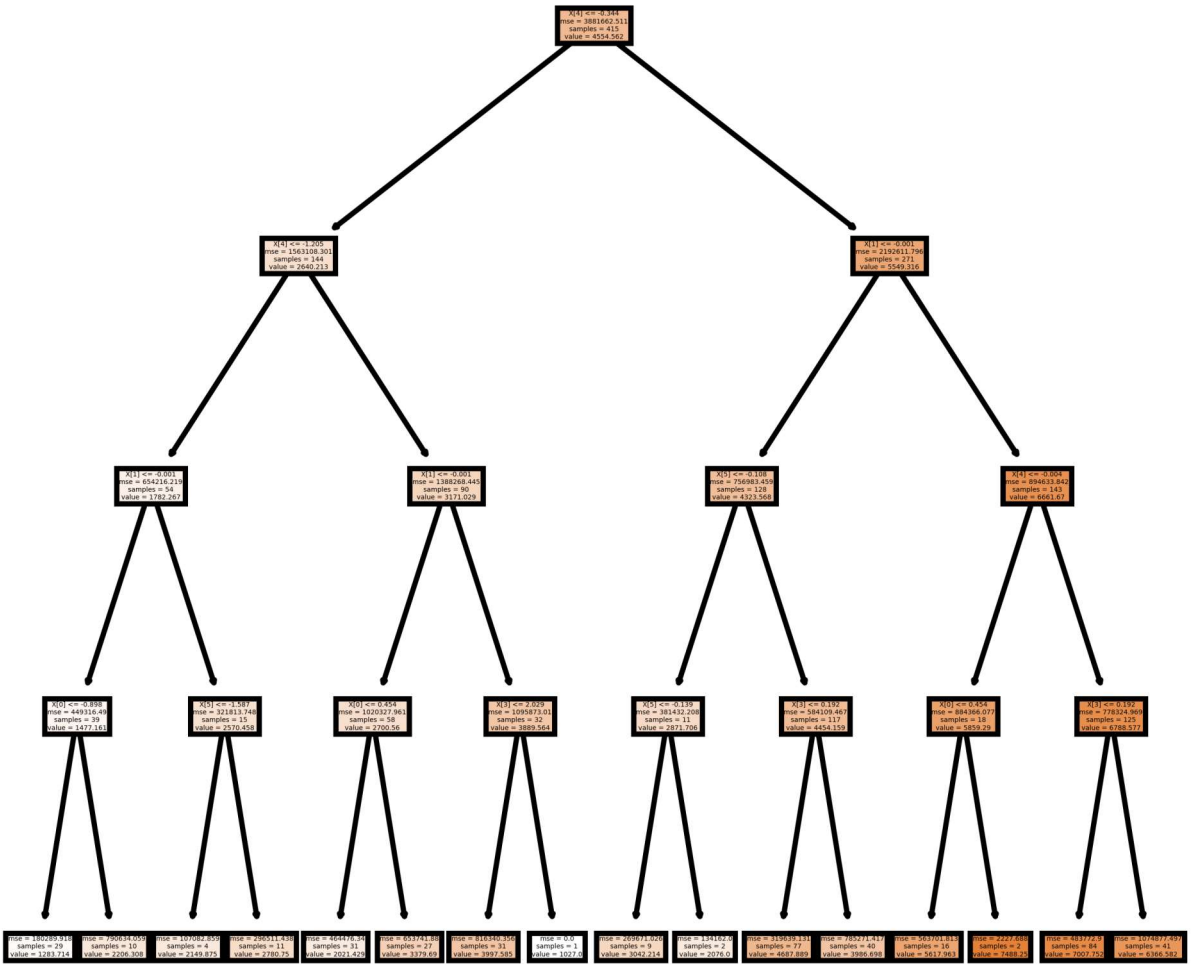
```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(rf.estimators_[4], filled = True);

plt.show()
```

```
rf = RandomForestRegressor(n_estimators = 2000, random_state = 42, max_depth=4)

train_val, test_val = rmse_cal(rf, bike_feat, bike_target)
```

```
temp ={}
temp['Train RMSE'] = train_val
temp['Test RMSE'] = test_val

results = pd.DataFrame.from_dict(temp,  orient='index')
results.columns =['RandomForest Results']

results
```

	RandomForest Results
Train RMSE	702.660216
Test RMSE	971.880932

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(rf.estimators_[4], filled = True);

plt.show()
```



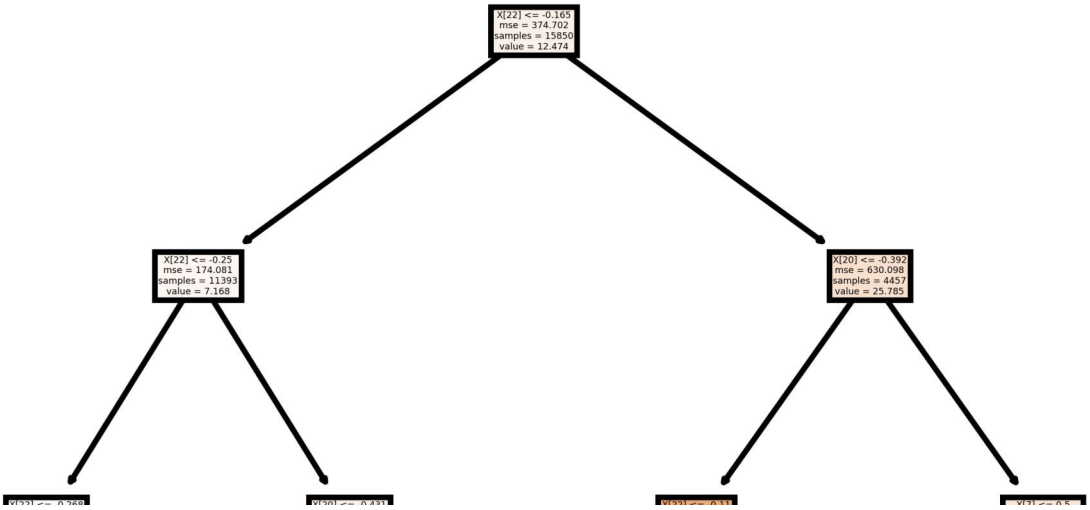
Master Dataset

```
rf = RandomForestRegressor(n_estimators = 500, random_state = 42, max_depth=4)

train_val, test_val = rmse_cal(rf, master_clean, master_target_df)


temp ={}
temp['Train RMSE'] = train_val
temp['Test RMSE'] = test_val

results = pd.DataFrame.from_dict(temp, orient='index')
```

```
results = pd.DataFrame.from_dict(temp, orient = 'index')
results.columns =['RandomForest Results']

results
```

	RandomForest Results
Train RMSE	11.421277
Test RMSE	12.069506

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(rf.estimators_[4], filled = True);

plt.show()
```

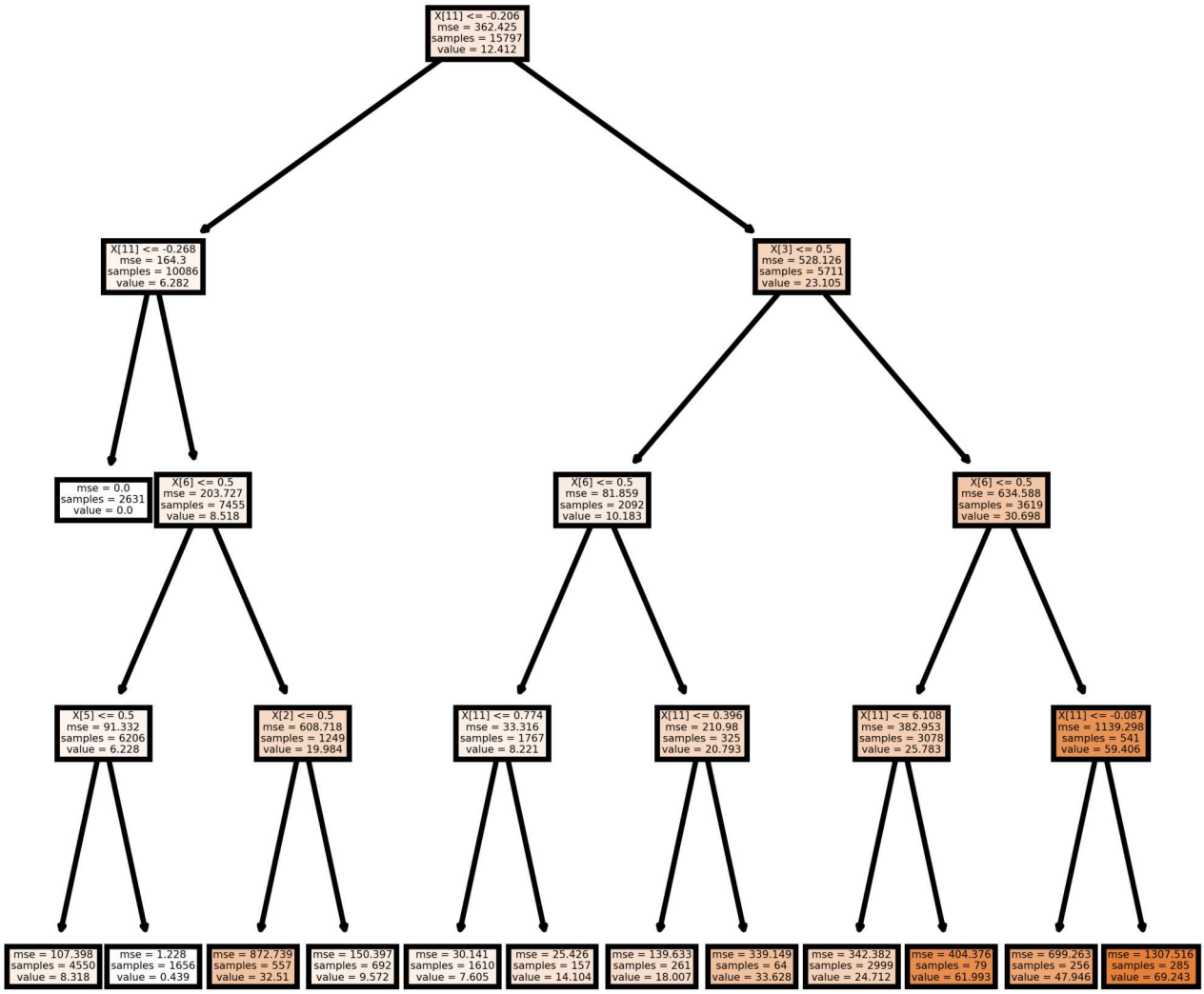



```
print('The following is after feature selection:')

rf = RandomForestRegressor(n_estimators = 2000, random_state = 42, max_depth=4)

train_val, test_val = rmse_cal(rf, master_feat, master_target_df)


temp ={}
temp['Train RMSE'] = train_val
temp['Test RMSE'] = test_val

results = pd.DataFrame.from_dict(temp,  orient='index')
results.columns =['RandomForest Results']

results
```

```
The following is after feature selection:
```

	RandomForest Results
Train RMSE	13.189695
Test RMSE	14.045662

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(rf.estimators_[2], filled = True);

plt.show()
```



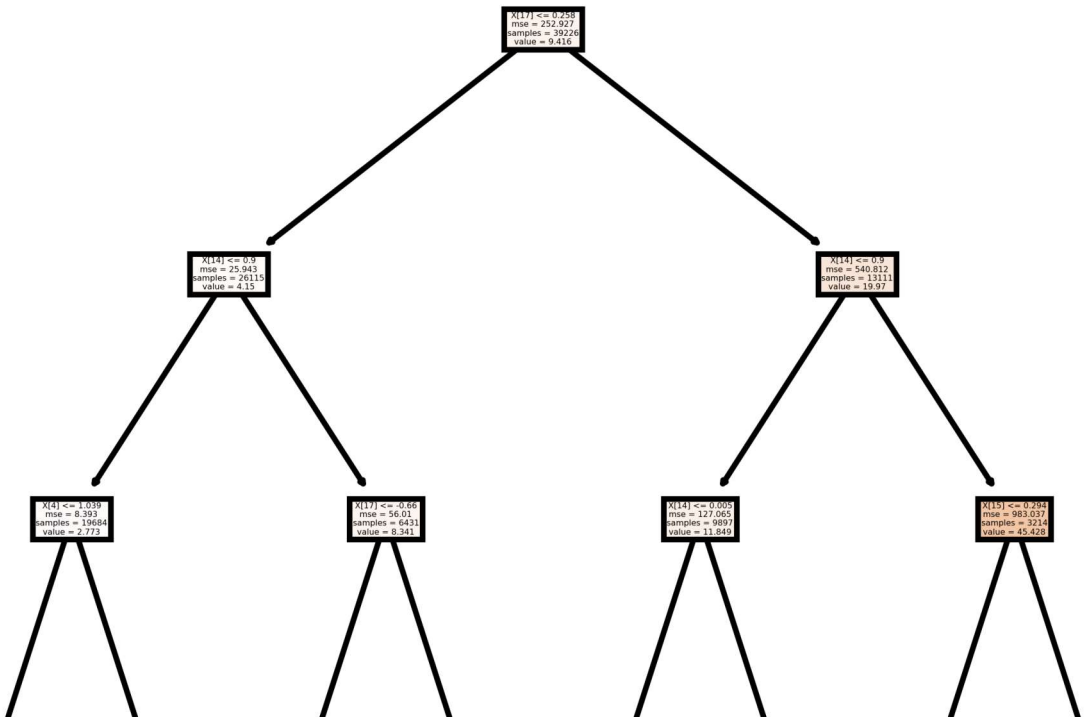
Transcoding Dataset

```
rf = RandomForestRegressor(n_estimators = 500, random_state = 42, max_depth=4)
targets = transcoding_df['utime']
train_val, test_val = rmse_cal(rf, transcoding_clean, targets)
```

```
temp ={}
temp['Train RMSE'] = train_val
temp['Test RMSE'] = test_val

results = pd.DataFrame.from_dict(temp,  orient='index')
results.columns =['RandomForest Results']

results
```

	RandomForest Results
Train RMSE	7.393055
Test RMSE	7.451648

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(rf.estimators_[4], filled = True);

plt.show()
```

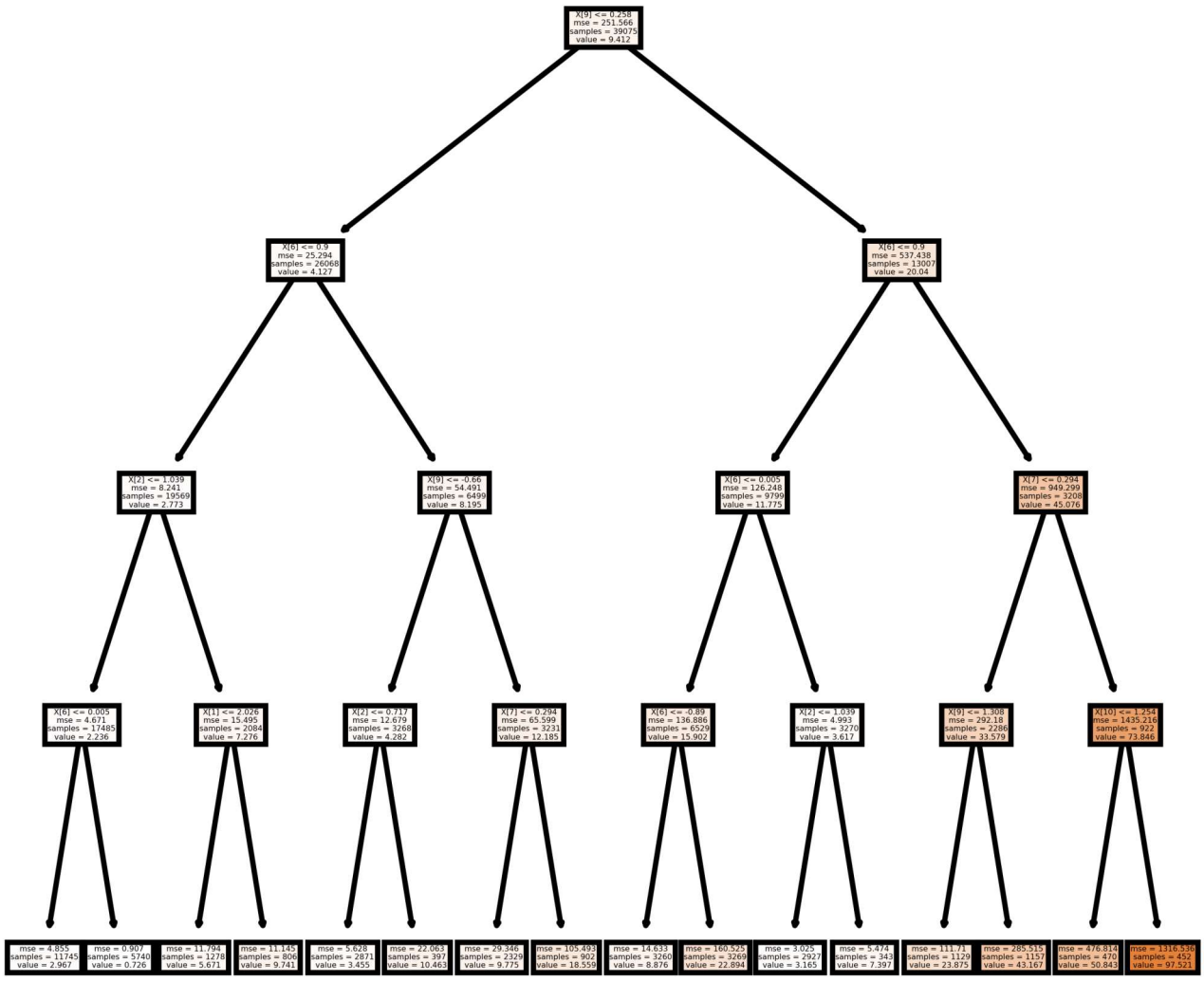



```
rf = RandomForestRegressor(n_estimators = 500, random_state = 42, max_depth=4)
targets = transcoding_df['utime']
train_val, test_val = rmse_cal(rf, transcoding_feat, targets)
```

```
temp ={}
temp['Train RMSE'] = train_val
temp['Test RMSE'] = test_val

results = pd.DataFrame.from_dict(temp,  orient='index')
results.columns =['RandomForest Results']

results
```

	RandomForest Results
Train RMSE	7.393055
Test RMSE	7.451648

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(rf.estimators_[2], filled = True);

plt.show()
```



LightGBM, CatBoost, Bayesian Optimization

```
!pip install catboost
```

```
!pip install lightgbm
!pip install scikit-optimize
```

```
Collecting catboost
  Downloading https://files.pythonhosted.org/packages/96/3b/bb419654adcf7efff42ed8a3f84
     |████████████████████████████████| 65.7MB 111kB/s
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (from c
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from ca
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from catb
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/l
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (
Installing collected packages: catboost
Successfully installed catboost-0.24.4
Requirement already satisfied: lightgbm in /usr/local/lib/python3.7/dist-packages (2.2.
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from li
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from li
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (
Collecting scikit-optimize
  Downloading https://files.pythonhosted.org/packages/8b/03/be33e89f55866065a02e515c5b3
     |████████████████████████████████| 102kB 4.3MB/s
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages
Collecting pyaml>=16.9
  Downloading https://files.pythonhosted.org/packages/15/c4/1310a054d33abc318426a956e7d
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-packages (from p
Installing collected packages: pyaml, scikit-optimize
Successfully installed pyaml-20.4.0 scikit-optimize-0.8.1
```

```
from catboost import CatBoostClassifier, Pool, cv, CatBoostRegressor
import lightgbm as lgb
from lightgbm.sklearn import LGBMModel
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from skopt import BayesSearchCV
```

```
class Boosting():

    def __init__(self, data, target, algo):
        '''
        df : DataFrame containing data
        target : target variable
```

```
        target : target variable
        '''
        self.data = data
        self.target = target
        self.algo = algo
        self.train_rmse = 0
        self.test_rmse = 0


    def train(self, params):
        '''
        Train the CatBoost Model on 10-fold cross-validation with the parameters
        given in params dictionary
        '''
        kf = KFold(n_splits=10)
        for trainset, testset in kf.split(self.data, self.target):

            # make training/testing datasets for fold
            X_train = self.data[trainset,:]
            y_train = self.target[trainset].astype('int')
            X_test = self.data[testset,:]
            y_test = self.target[testset]


            if self.algo == 'cat':
                # convert to correct data format
                train = Pool(data=X_train, label=y_train)
                eval = Pool(data=X_test, label=y_test)

                # define model
                model = CatBoostRegressor(loss_function='RMSE',
                                          iterations=params['iterations'],
                                          learning_rate=params['learning_rate'],
                                          depth=params['depth'],
                                          border_count=params['border_count'],
                                          l2_leaf_reg=params['l2_leaf_reg'],
                                          random_seed=42)

                # train
                model.fit(train,
                          use_best_model=True,
                          eval_set=eval,
                          early_stopping_rounds=5)


            elif self.algo == 'lgb':
                # define model with hyperparameters
                model = lgb.LGBMRegressor(boosting_type='gbdt',
                                          num_leaves=params['num_leaves'],
                                          max_depth=params['max_depth'],
                                          learning_rate=params['learning_rate'],
                                          n_estimators=params['num_trees'],
```

```
                                    min_child_samples=params['min_data_in_leaf'])

            # train
            model.fit(X_train, y_train,
                      eval_set=[(X_test, y_test)],
                      eval_metric='rmse',
                      early_stopping_rounds=5,
                      verbose=False)

        else:
            print("algo must be either 'cat' or 'lgb' ")
            return 0


        # predict
        test_predictions = model.predict(X_test)
        train_predictions = model.predict(X_train)
        self.test_rmse += mean_squared_error(y_test, test_predictions,
                                             squared=False)
        self.train_rmse += mean_squared_error(y_train, train_predictions,
                                              squared=False)

    self.test_rmse /= 10
    self.train_rmse /= 10
    return [self.train_rmse, self.test_rmse]




def bayes_param_search(self):
    '''
    Performs optimization of hyperparameters using Bayesian Optimization

    https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-hov

    https://machinelearningmastery.com/configure-gradient-boosting-algorithm/

    https://effectiveml.com/using-grid-search-to-optimise-catboost-parameters.html



    Hyperparameters for LightGBM
    num_estimators: max number of trees to fit
    num_leaves: the main parameter to control the complexity of the tree model
                should be smaller than 2^(max_depth) to avoid overfit
    max_depth: limits the tree depth explicitly
    min_child_samples: very important parameter to prevent over-fitting
                       optimal value depends on the number of training samples and num_lea
                       setting to a large value may cause under-fitting
                       hundreds or thousands is enough for a large dataset
    learning_rate : reducing the gradient step; affects the overall time of training
```

```
Hyperparameters for CatBoost
iterations : number of trees (same as "num_trees")
depth : depth of the tree
l2_leaf_reg : regularizer to find the best possible
border_count : number of splits for numerical features
learning_rate

'''

if self.algo == 'cat':
    booster = CatBoostRegressor(loss_function='RMSE',
                                random_seed=42)
    params = {
        'iterations': (100, 1000, 'log-uniform'),
        'depth': (2, 4, 6, 8, 10),
        'l2_leaf_reg': (0, 0.05, 1, 3, 5, 10),
        'border_count': (100, 200, 500, 1000, 2000, 5000),
        'learning_rate': (0.001, 0.01, 0.05, 0.1)
    }
elif self.algo == 'lgb':
    booster = lgb.LGBMRegressor(boosting_type='gbdt')
    params = {
        'num_leaves': (16, 64, 'log-uniform'),
        'max_depth': (2,4, 6, 8, 10),
        'learning_rate': (0.001, 0.01, 0.05, 0.1),
        'n_estimators': (100, 1000, 'log-uniform'),
        'min_child_samples': (50, 100, 200, 500)
    }
else:
    print("algo must be either 'cat' or 'lgb' ")
    return 0


# define search space
model = BayesSearchCV(
    booster,
    params,
    cv=10,
    n_iter=10,
    scoring='neg_root_mean_squared_error',
    n_points=2,
    n_jobs=4,
    verbose=1,
    return_train_score=True
)

# fit model to data
model.fit(self.data, self.target)

# get results
```

```
        results = model.cv_results_
        best_score = model.best_score_
        best_setting = model.cv_results_['params'][model.best_index_]

        return [results, best_score, best_setting]
```

LightGBM

```
'''
Best param search using Bayesian optimization for LightGBM on the bike dataset
'''

lgbm = Boosting(bike_clean, bike_target, algo='lgb')
results, best_score, best_setting = lgbm.bayes_param_search()
```

```
    Fitting 10 folds for each of 2 candidates, totalling 20 fits
    [Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
    [Parallel(n_jobs=4)]: Done  20 out of  20 | elapsed:    1.4s finished
    [Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
    Fitting 10 folds for each of 2 candidates, totalling 20 fits
    [Parallel(n_jobs=4)]: Done  20 out of  20 | elapsed:    2.0s finished
    [Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
    Fitting 10 folds for each of 2 candidates, totalling 20 fits
    [Parallel(n_jobs=4)]: Done  13 out of  20 | elapsed:    1.1s remaining:    0.6s
    [Parallel(n_jobs=4)]: Done  20 out of  20 | elapsed:    1.4s finished
    [Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
    Fitting 10 folds for each of 2 candidates, totalling 20 fits
    [Parallel(n_jobs=4)]: Done  13 out of  20 | elapsed:    1.7s remaining:    0.9s
    [Parallel(n_jobs=4)]: Done  20 out of  20 | elapsed:    2.2s finished
    Fitting 10 folds for each of 2 candidates, totalling 20 fits
    [Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
    [Parallel(n_jobs=4)]: Done  20 out of  20 | elapsed:    2.1s finished
```

```
print(best_score)
```

```
    -792.7789250403005
```

```
'''
LightGBM using the best features found below on the feature selected bike dataset
'''

params = {}
params['num_leaves'] = best_setting['num_leaves']
params['max_depth'] = best_setting['max_depth']
params['learning_rate'] = best_setting['learning_rate']
params['num_trees'] = best_setting['n_estimators']
params['min_data_in_leaf'] =  best_setting['min_child_samples']

lgbm = Boosting(bike_feat, bike_target, algo='lgb')
```

```
train_rmse, test_rmse = lgbm.train(params)

print("train RMSE: ", train_rmse)
print("test RMSE: ", test_rmse)
```

```
train RMSE:  867.9628371452421
test RMSE:  868.1911310105952
```

CatBoosting

```
'''
Best param search using Bayesian optimization for CatBoost on the bike dataset
'''

cat = Boosting(bike_clean, bike_target, algo='cat')
results, best_score, best_setting = cat.bayes_param_search()
```

```
Fitting 10 folds for each of 2 candidates, totalling 20 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  20 out of  20 | elapsed:   30.3s finished
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
Fitting 10 folds for each of 2 candidates, totalling 20 fits
[Parallel(n_jobs=4)]: Done  20 out of  20 | elapsed:  1.1min finished
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
Fitting 10 folds for each of 2 candidates, totalling 20 fits
[Parallel(n_jobs=4)]: Done  20 out of  20 | elapsed:    9.9s finished
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
Fitting 10 folds for each of 2 candidates, totalling 20 fits
[Parallel(n_jobs=4)]: Done  20 out of  20 | elapsed:   17.4s finished
Fitting 10 folds for each of 2 candidates, totalling 20 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  20 out of  20 | elapsed:  1.3min finished
0:      learn: 1872.2439479     total: 498us    remaining: 128ms
1:      learn: 1814.6418380     total: 1.2ms    remaining: 152ms
2:      learn: 1759.9132789     total: 1.68ms   remaining: 142ms
3:      learn: 1705.5859327     total: 2.05ms   remaining: 130ms
4:      learn: 1655.8869797     total: 2.52ms   remaining: 127ms
5:      learn: 1609.9687649     total: 3.01ms   remaining: 126ms
6:      learn: 1565.4952956     total: 3.86ms   remaining: 138ms
7:      learn: 1525.2314837     total: 4.3ms    remaining: 134ms
8:      learn: 1488.8660720     total: 6.66ms   remaining: 183ms
9:      learn: 1451.3065492     total: 7.12ms   remaining: 176ms
10:     learn: 1419.3147562     total: 7.54ms   remaining: 169ms
11:     learn: 1387.1677157     total: 8.11ms   remaining: 166ms
12:     learn: 1364.1339200     total: 14.1ms   remaining: 264ms
13:     learn: 1335.0905048     total: 14.6ms   remaining: 254ms
14:     learn: 1304.8607474     total: 15.1ms   remaining: 243ms
15:     learn: 1284.3607843     total: 15.5ms   remaining: 234ms
16:     learn: 1265.8719596     total: 15.9ms   remaining: 225ms
17:     learn: 1243.9649001     total: 16.3ms   remaining: 216ms
18:     learn: 1220.7212586     total: 16.7ms   remaining: 209ms
19:     learn: 1196.3874925     total: 17.1ms   remaining: 202ms
20:     learn: 1177.0613246     total: 17.5ms   remaining: 196ms
```

```
21:     learn: 1157.6282575     total: 17.8ms   remaining: 191ms
22:     learn: 1139.1946332     total: 18.3ms   remaining: 186ms
23:     learn: 1122.2520138     total: 18.9ms   remaining: 184ms
24:     learn: 1106.9860135     total: 19.6ms   remaining: 182ms
25:     learn: 1089.3703894     total: 20.2ms   remaining: 180ms
26:     learn: 1076.3720565     total: 20.9ms   remaining: 178ms
27:     learn: 1061.5444074     total: 21.4ms   remaining: 175ms
28:     learn: 1047.6227226     total: 21.8ms   remaining: 171ms
29:     learn: 1036.8048188     total: 22.2ms   remaining: 168ms
30:     learn: 1024.6777165     total: 22.7ms   remaining: 165ms
31:     learn: 1013.4129871     total: 23.1ms   remaining: 163ms
32:     learn: 1004.1752489     total: 23.6ms   remaining: 160ms
33:     learn: 992.3428740      total: 24.2ms   remaining: 159ms
34:     learn: 982.1818370      total: 24.7ms   remaining: 157ms
35:     learn: 972.0108461      total: 25.2ms   remaining: 155ms
36:     learn: 962.0037285      total: 25.7ms   remaining: 153ms
37:     learn: 952.2792363      total: 26.1ms   remaining: 150ms
38:     learn: 943.2443419      total: 26.6ms   remaining: 149ms
39:     learn: 936.0134995      total: 27.2ms   remaining: 147ms
40:     learn: 928.5162482      total: 27.7ms   remaining: 146ms
41:     learn: 921.4030283      total: 28.1ms   remaining: 144ms
42:     learn: 913.9965291      total: 28.5ms   remaining: 142ms
43:     learn: 906.1600607      total: 29ms     remaining: 140ms
```

```
print(best_score)
```

```
-702.0636067575873
```

```
print(best_setting)
```

```
OrderedDict([('border_count', 200), ('depth', 2), ('iterations', 257), ('l2_leaf_reg',
```

```
'''
CatBoost using the best features found below on the feature selected bike dataset
'''

params = {}
params['iterations'] = best_setting['iterations']
params['depth'] = best_setting['depth']
params['l2_leaf_reg'] = best_setting['l2_leaf_reg']
params['border_count'] = best_setting['border_count']
params['learning_rate'] =  best_setting['learning_rate']

cat = Boosting(bike_feat, bike_target, algo='cat')
train_rmse, test_rmse = cat.train(params)

print("train RMSE: ", train_rmse)
print("test RMSE: ", test_rmse)
```

```
0:      learn: 1693.9146180     test: 3240.5664301      best: 3240.5664301 (0)  total
1:      learn: 1646.9076253     test: 3152.6079582      best: 3152.6079582 (1)  total
2:      learn: 1603.8774378     test: 3071.9530190      best: 3071.9530190 (2)  total
```

```
3:      learn: 1562.2715986     test: 2990.1328937      best: 2990.1328937 (3)  total
4:      learn: 1522.9254668     test: 2911.2365061      best: 2911.2365061 (4)  total
5:      learn: 1486.5097721     test: 2837.0573019      best: 2837.0573019 (5)  total
6:      learn: 1453.8686388     test: 2768.4848345      best: 2768.4848345 (6)  total
7:      learn: 1425.1261654     test: 2708.2799657      best: 2708.2799657 (7)  total
8:      learn: 1395.6073590     test: 2643.3941858      best: 2643.3941858 (8)  total
9:      learn: 1366.9462886     test: 2562.6836713      best: 2562.6836713 (9)  total
10:     learn: 1341.1282984     test: 2502.5362340      best: 2502.5362340 (10) total
11:     learn: 1318.1198964     test: 2445.1328355      best: 2445.1328355 (11) total
12:     learn: 1293.9701214     test: 2372.0941275      best: 2372.0941275 (12) total
13:     learn: 1273.6724580     test: 2322.3588719      best: 2322.3588719 (13) total
14:     learn: 1254.7637205     test: 2271.4178258      best: 2271.4178258 (14) total
15:     learn: 1236.2701608     test: 2221.4045601      best: 2221.4045601 (15) total
16:     learn: 1216.8659373     test: 2157.1652375      best: 2157.1652375 (16) total
17:     learn: 1200.0877265     test: 2113.1218996      best: 2113.1218996 (17) total
18:     learn: 1185.7138591     test: 2081.1642628      best: 2081.1642628 (18) total
19:     learn: 1168.9469916     test: 2021.9513960      best: 2021.9513960 (19) total
20:     learn: 1154.9849055     test: 1981.0350729      best: 1981.0350729 (20) total
21:     learn: 1141.6776076     test: 1954.8984526      best: 1954.8984526 (21) total
22:     learn: 1131.8960234     test: 1910.9492146      best: 1910.9492146 (22) total
23:     learn: 1118.4639393     test: 1874.3432216      best: 1874.3432216 (23) total
24:     learn: 1104.4090469     test: 1819.9689672      best: 1819.9689672 (24) total
25:     learn: 1096.2251710     test: 1782.9834203      best: 1782.9834203 (25) total
26:     learn: 1085.4426870     test: 1751.4409833      best: 1751.4409833 (26) total
27:     learn: 1074.8665221     test: 1716.3878404      best: 1716.3878404 (27) total
28:     learn: 1063.8244478     test: 1688.6107588      best: 1688.6107588 (28) total
29:     learn: 1054.5298861     test: 1667.3824367      best: 1667.3824367 (29) total
30:     learn: 1045.8944622     test: 1640.3944397      best: 1640.3944397 (30) total
31:     learn: 1037.3556135     test: 1609.1503744      best: 1609.1503744 (31) total
32:     learn: 1028.1971276     test: 1588.5638986      best: 1588.5638986 (32) total
33:     learn: 1017.0328941     test: 1542.2330779      best: 1542.2330779 (33) total
34:     learn: 1009.2143276     test: 1527.3146148      best: 1527.3146148 (34) total
35:     learn: 1002.8561478     test: 1491.7452537      best: 1491.7452537 (35) total
36:     learn: 994.2048059      test: 1460.8927273      best: 1460.8927273 (36) total
37:     learn: 986.7264806      test: 1445.8542419      best: 1445.8542419 (37) total
38:     learn: 979.3739137      test: 1432.4866679      best: 1432.4866679 (38) total
39:     learn: 974.4576296      test: 1408.3151418      best: 1408.3151418 (39) total
40:     learn: 967.5663142      test: 1387.2590827      best: 1387.2590827 (40) total
41:     learn: 958.9517437      test: 1347.5581798      best: 1347.5581798 (41) total
42:     learn: 954.3644985      test: 1316.6607330      best: 1316.6607330 (42) total
43:     learn: 948.3100725      test: 1314.9639820      best: 1314.9639820 (43) total
44:     learn: 940.5504188      test: 1277.2925084      best: 1277.2925084 (44) total
45:     learn: 934.9056743      test: 1255.7997034      best: 1255.7997034 (45) total
46:     learn: 929.5854403      test: 1254.3650379      best: 1254.3650379 (46) total
47:     learn: 923.5470121      test: 1238.2295475      best: 1238.2295475 (47) total
48:     learn: 916.7886448      test: 1218.2670451      best: 1218.2670451 (48) total
49:     learn: 911.6670850      test: 1215.7631416      best: 1215.7631416 (49) total
50:     learn: 906.9422748      test: 1201.3758578      best: 1201.3758578 (50) total
51:     learn: 903.4515880      test: 1191.5027760      best: 1191.5027760 (51) total
52:     learn: 899.1472323      test: 1181.5338591      best: 1181.5338591 (52) total
53:     learn: 894.3281105      test: 1169.7395484      best: 1169.7395484 (53) total
54:     learn: 888.4950106      test: 1135.5642760      best: 1135.5642760 (54) total
55:     learn: 884.7656415      test: 1134.7360442      best: 1134.7360442 (55) total
56:     learn: 880.2895798      test: 1121.8202305      best: 1121.8202305 (56) total
57:     learn: 875.8754839      test: 1111.1425599      best: 1111.1425599 (57) total
```