Regression

Project One

By: Rohith Lingala
Harshith Kasthuri
Charan Reddy Mannuru
James Notoma
Josh Kolesar

Dr. John Miller
CSCI 4360- Data Science II

Sept 20th, 2024

Introduction

In data science, there are many different approaches to constructing a model to solve a given problem. The choice of model can impact the accuracy and reliability of predictions, especially when dealing with diverse datasets. Our report explores different regression algorithms, from two programming languages, across four different datasets, with the goal of comparing how well each model handles different types of problems.

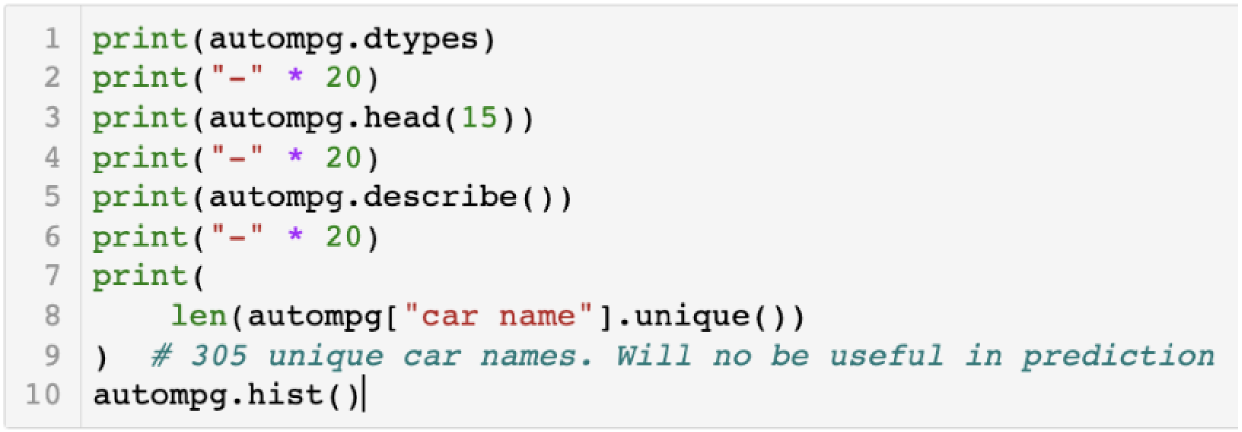
The four datasets that we will be exploring are Auto-MPG, Forest Fires, Seoul Bike Sharing Demand, and Airfoil Self Noise. Auto-MPG created by Ross Quinlan contains data that represents fuel consumption in miles per gallon with attributes that correspond to a car's specifications. The Forest Fires dataset contains meteorological data that shows the height and weather conditions for an area in Poland. The Seoul Bike Sharing Demand dataset contains data on the number of public bicycles rented per hour in Seoul. The Airfoil Self Noise dataset, from NASA, contains data taken from aerodynamic and acoustic tests of airfoil blade sections in a wind tunnel.

The primary tools used in our project are ScalaTion, a scalable simulation library written in Scala, and Python's scikit-learn, statsmodels, and gplearn a Genetic Programming package (for Symbolic Regression). The regression algorithms we will be comparing are Linear Regression, Ridge Regression, Lasso Regression, Transformed Regression, and Symbolic Regression. These tools will serve as our testing platform for analyzing the performance of the models.

Auto-MPG

Exploratory Data Analysis

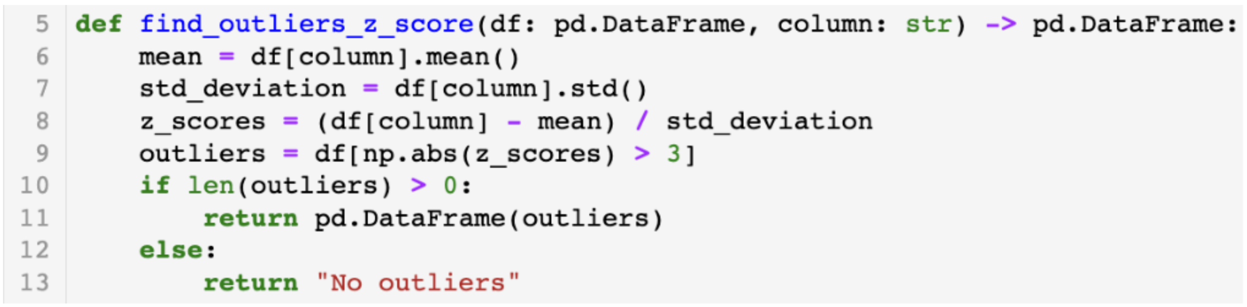
Autompg is a dataset that contains data relevant to the miles per gallon performance of a car. Once the data had been imported I obtained column datatypes, summary statistics, and examined the overall data distribution by column using histogram plots. Below is an example of the code I use to acquire an introductory and simple analysis of the data. In addition to exploring the data manually, I downloaded the autompg.names file. By reading this file, I was able to note that “?” marked unknown data.



```python
print(autompg.dtypes)
print("-" * 20)
print(autompg.head(15))
print("-" * 20)
print(autompg.describe())
print("-" * 20)
print(
    len(autompg["car name"].unique())
)  # 305 unique car names. Will no be useful in prediction
autompg.hist()
```

Data Preprocessing

Based on my manual exploratory analysis and the autompg.names file, I learned that car names had more than 300 unique entries and “?” marked unknown data. Based on these observations, I entirely removed the “car_names” column. Additionally, I mapped all “?” to NaN, indicating to python that the value was truly null. Then, I imputed all values that were missing using K Nearest Neighbors (KNN). KNN is a method of value imputation, that in the context of linear regression finds the euclidian distance between many other instances of data. A subset of these neighbors is then used to impute the missing value. How many neighbors are used is up to the developer. I used 5 as it is a good balance between efficiency and accuracy. Finally, I discovered outliers by u calculating z scores. Based on z-score measurements, there were very few outliers. None of which I removed, as all the data seemed reasonable within context. Below is my process for finding outliers using a z-score.

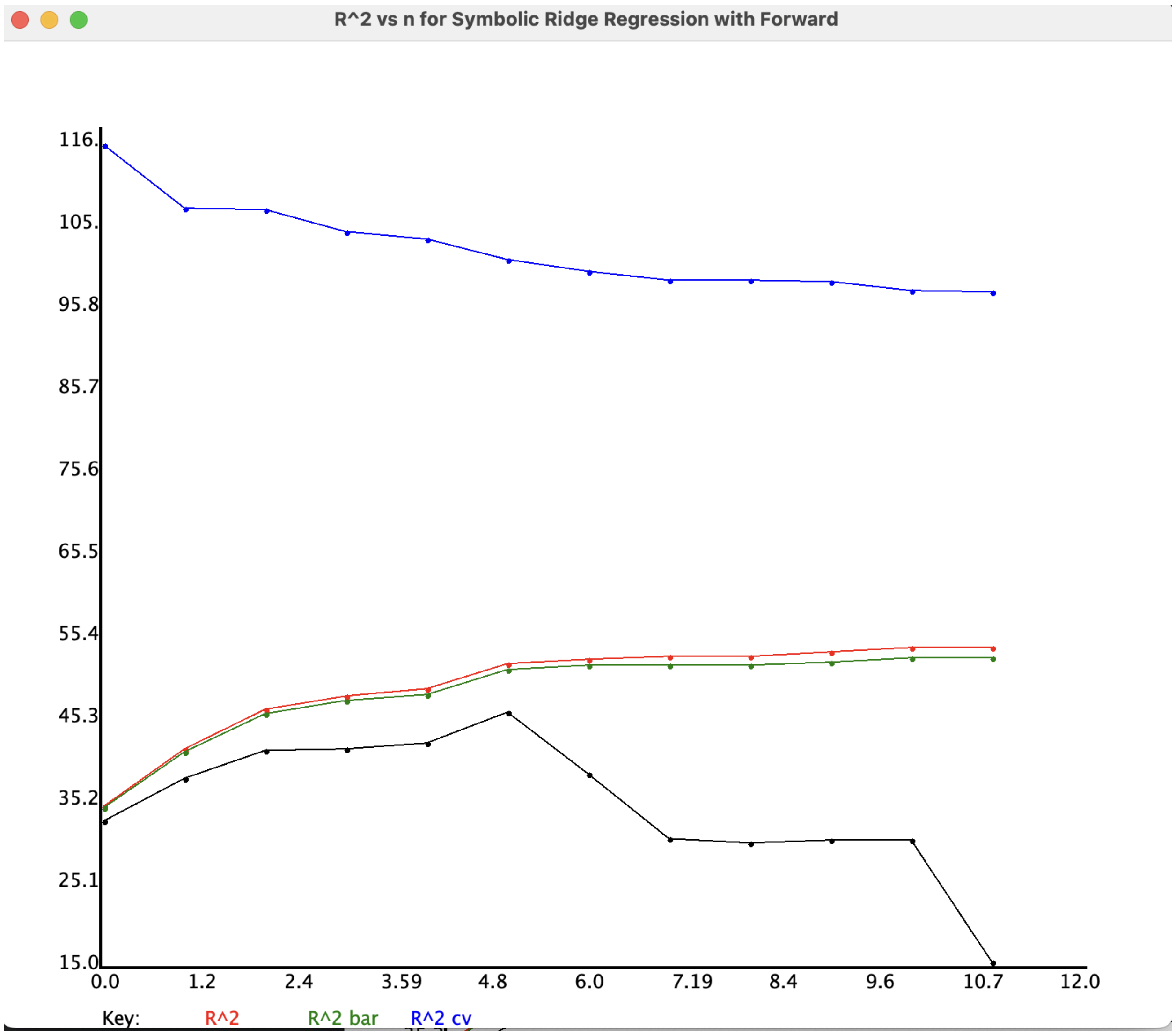


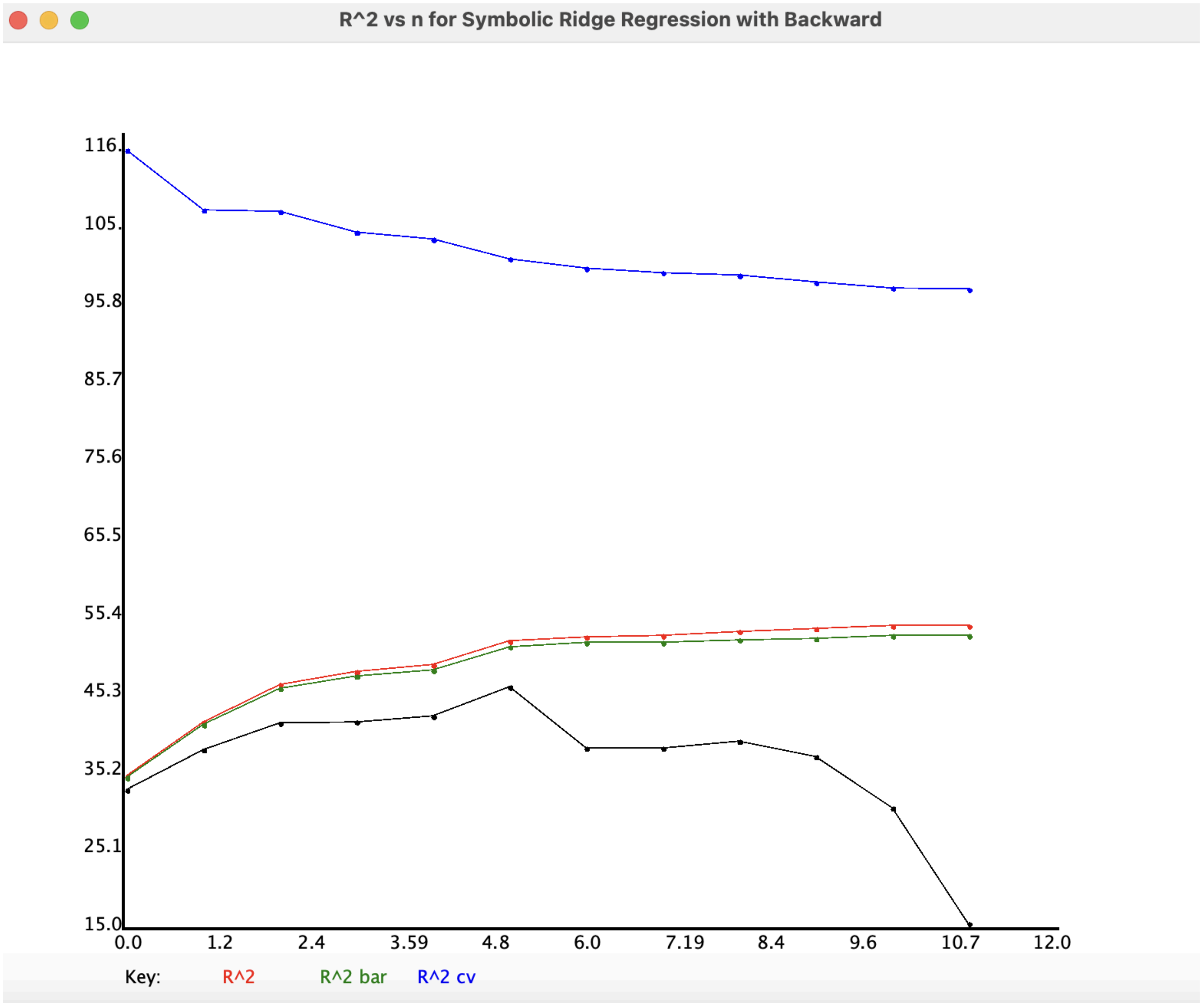
```python
def find_outliers_z_score(df: pd.DataFrame, column: str) -> pd.DataFrame:
    mean = df[column].mean()
    std_deviation = df[column].std()
    z_scores = (df[column] - mean) / std_deviation
    outliers = df[np.abs(z_scores) > 3]
    if len(outliers) > 0:
        return pd.DataFrame(outliers)
    else:
        return "No outliers"
```

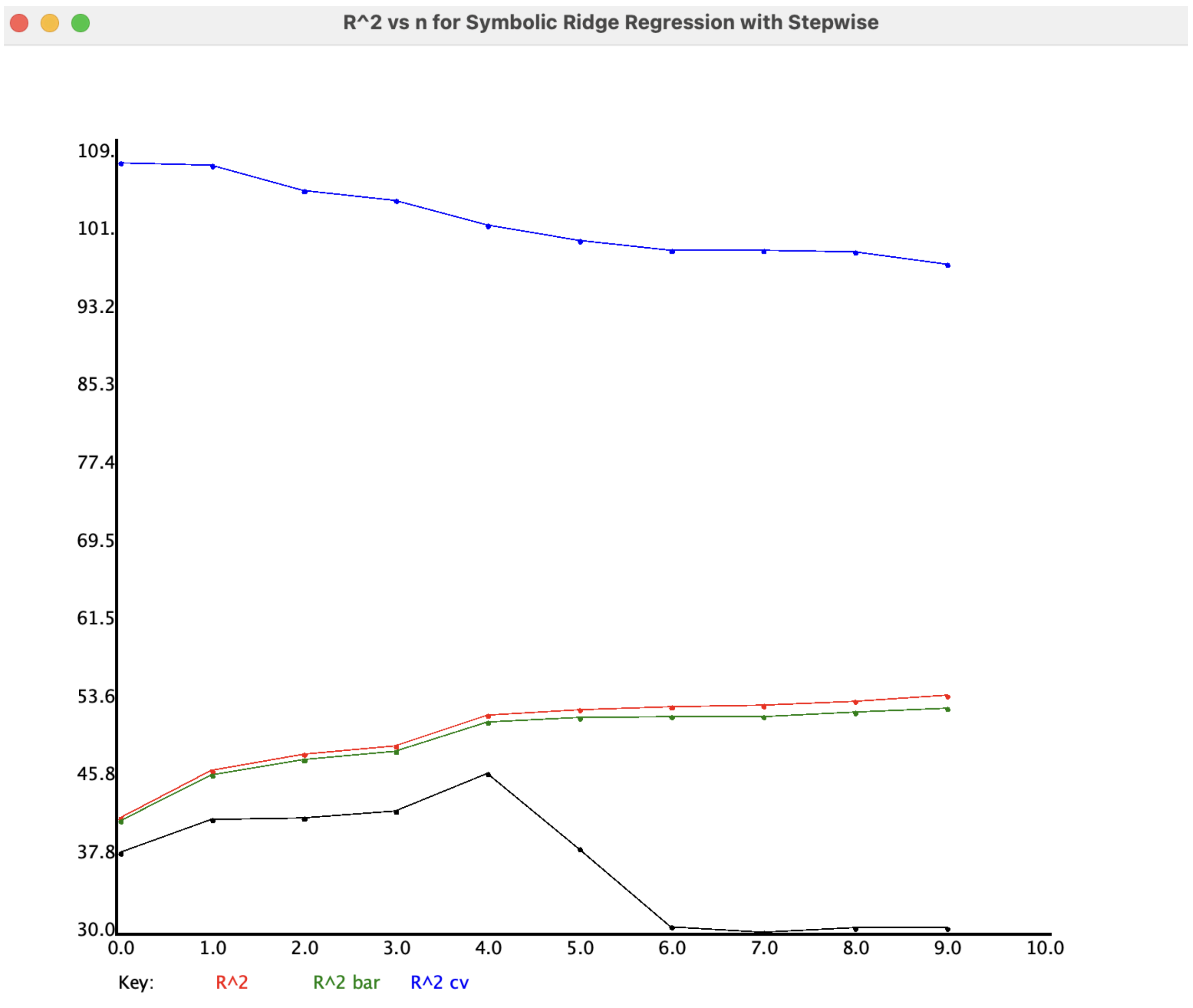
Feature Selection

To begin feature selection I checked the covariance and correlation between all columns.

Although there were positive correlations exceeding .9, no columns were removed, as there were only seven input features in the model. My decision to eliminate car names from auto mpg was made there by seven input features for the prediction of miles per gallon. Generating all unique combinations would only be one-hundred and twenty-eight unique combinations. Because one-hundred and twenty-eight combinations are manageable, I decided to create a list holding all such combinations. This list would be used for all models when appropriate. Below is an example of how I created a list of tuples, each tuple holding a unique column combination.




R^2 vs n for Symbolic Ridge Regression with Backward
116.
105.
95.8
85.7
75.6
65.5
55.4
45.3
35.2
25.1
15.0
0.0
1.2
2.4
3.59
4.8
6.0
7.19
8.4
9.6
10.7
12.0
Key:
R^2
R^2 bar
R^2 cv



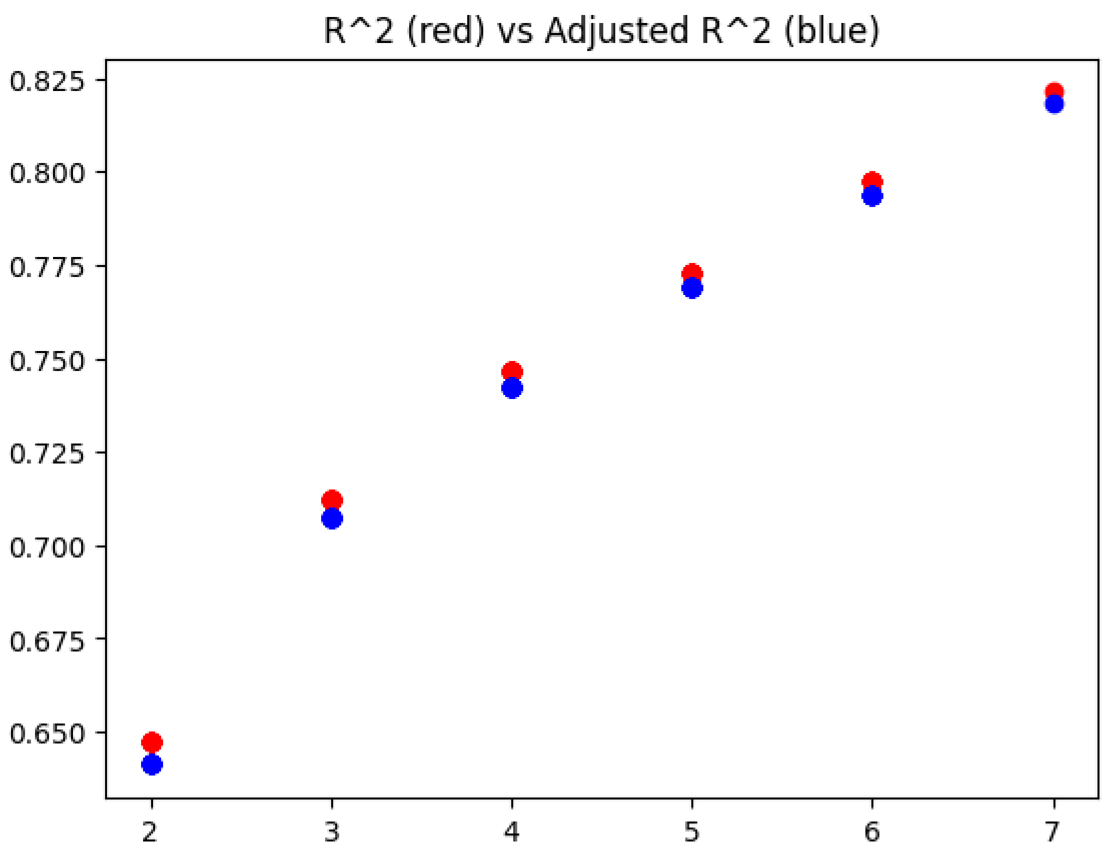
Linear Regression

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. This model type assumes a linear relationship between the input features and the output variable. It aims to find the best-fitting line that represents the relationship between these variables.

The best performance method for the linear regression model for the autompg dataset in python was the in-sample method that utilized all seven input features. This method consistently scored the highest mean R-Squared and Adjusted R-Squared values, floating or exceeding .80. Additionally, the mean absolute errors and mean square errors were the least. Therefore, further indicating that the in-sample method yielded the best results on available data. Scalation had similar results, as the Five Fold Cross validation returned R-Squared measurements around .808

These results are expected as the in-sample method has a low bias and high variance toward new data. Meaning, the in-sample model is likely to overfit the current data, as it uses one-hundred percent of the input features and output variable. Because of the in-sample method's tendency to overfit, we will see that it likely performs the best on current data.

In Sample Linear Regresssion

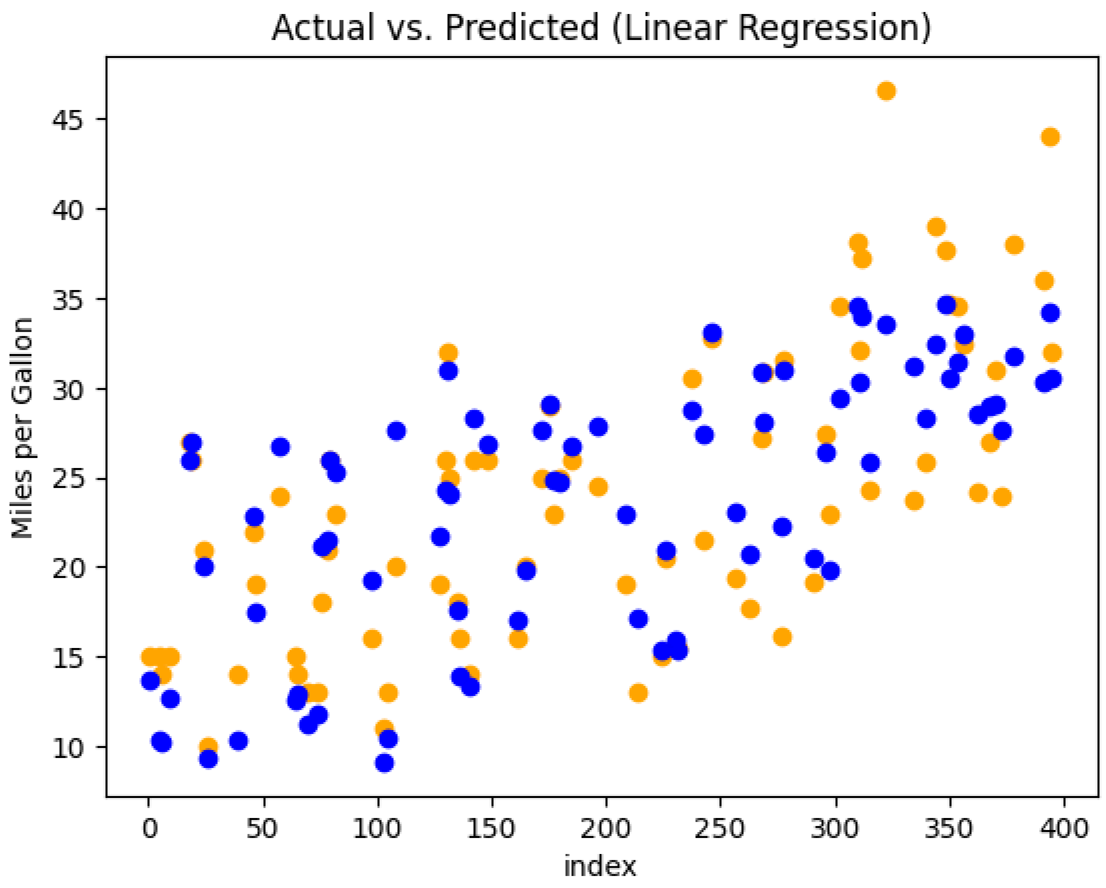


On average, the number of parameters that seem optimal for a simple linear regression model is 2.0

Based on R Squared/Adjusted R Squared calculations the optimal combinations are:

['weight', 'model year']: R Squared: 0.8261019976179578

['weight', 'model year']: Adjusted R Squared 0.8091952473863704



Lasso Regression

Lasso regression is a type of linear regression that adds a penalty term, lambda, to the cost function, which helps to control overfitting and improve the generalization of the model. The penalty term encourages the model to set some of the coefficients of the input features close to or at zero.

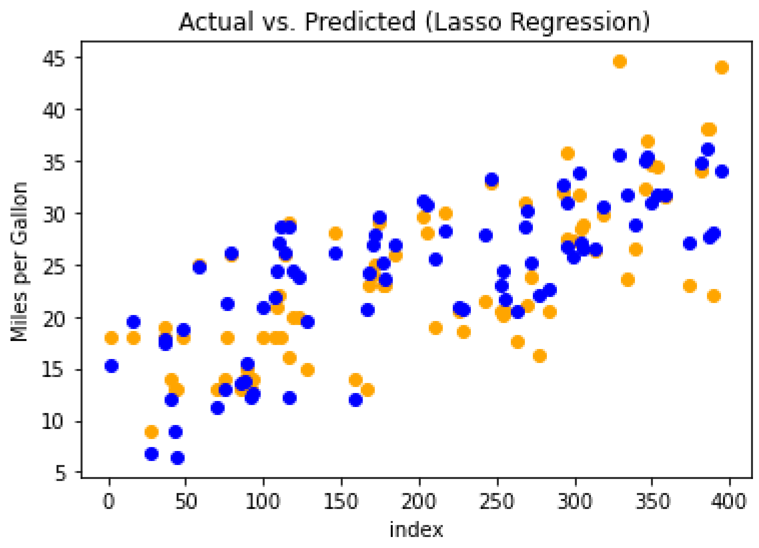
The best performance method for the lasso regression model for the autompg dataset in python was the in-sample method that utilized all seven input features. This method consistently scored the highest mean R-Squared and Adjusted R-Squared values, floating or exceeding .80. Additionally, the mean absolute errors and mean square errors were the least. Therefore, further indicating that the in-sample method yielded the best results on available data. The most optimal lambda value was 0.01, indicating that for in-sample modeling, the closer our model emulated regular linear regression, the better it performed. Scalation yielded similar results as our optimal lambda was .01. This lambda results in R-Squared metrics around .79.

On average, the number of parameters that seem optimal for a Lasso Linear regression model is 7.0

Based on R Squared/Adjusted R Squared calculations the optimal combinations are:

['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model year', 'origin']: R Squared: 0.7546618911829622

['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model year', 'origin']: Adjusted R Squared 0.7502583866657333



MAE: 2.8444

MSE: 14.6996

Five Fold Cross Validation Lasso Regression

LassoCV Mean R-squared score: 0.808 (+/- 0.070)

LassoCV Mean Adjusted R-squared score: 0.804 (+/- 0.072)

LassoCV Optimal lambda: 0.010

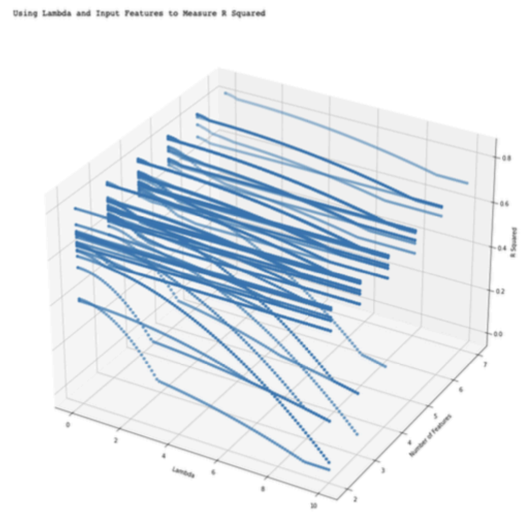
When using the Train and test method, we consistently observed a lambda penalty of around .313. When implementing five-fold cross-validation the optimal lamba becomes lesser around .01. Because one purpose of applying a lambda penalty to a loss function is to prevent overfitting, It is reasonable to expect a penalty associated with the train and test method as well as the five-fold cross-validation method.

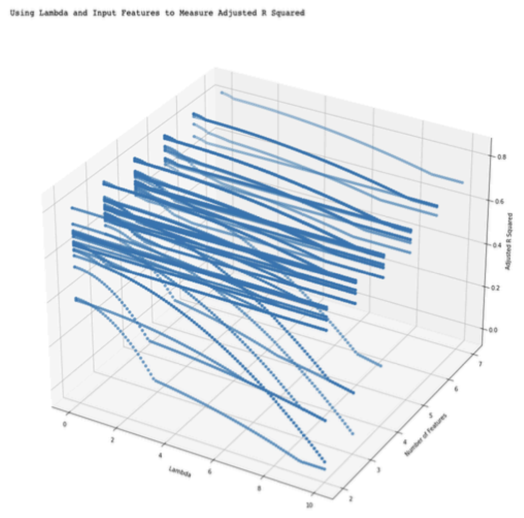
Ridge Regression

Ridge regression is a type of linear regression that adds a penalty term, lambda, to the cost function, which helps to control overfitting and improve the generalization of the model. The penalty term encourages the model to set some of the coefficients of the input features close to but never at zero.

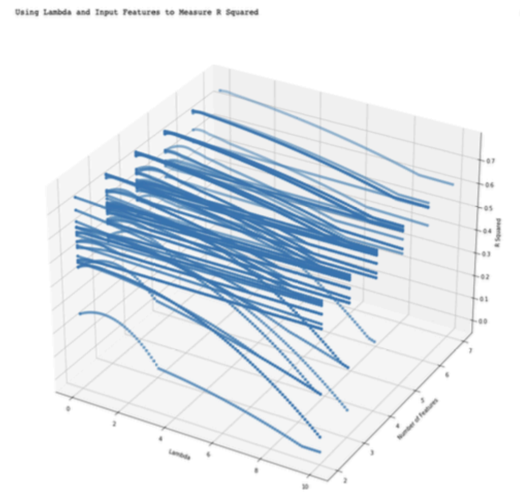
The best performance method for the ridge regression model for the autompg dataset in python was the in-sample method that utilized all seven input features. This method consistently scored the highest mean R-Squared and Adjusted R-Squared values, floating or exceeding .80. Additionally, the mean absolute errors and mean square errors were the least. Therefore, further indicating that the in-sample method yielded the best results on available data. The most optimal lambda value was 0.0, indicating that for in-sample modeling, there was no penalty that could be applied to the loss function to make ridge regression better than regular linear regression. Using Scalation yielded similar results, as R-Squared metrics were around .80 and the optimal lambda was .01.

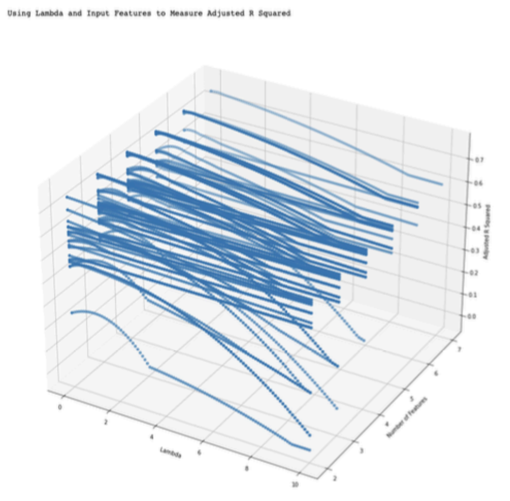
In sample R-Squared and Adjusted R-Squared vs (Lambda Value and Number of Features used in the Regression Model):





Test and Train R-Squared and Adjusted R-Squared vs (Lambda Value and Number of Features used in the Regression Model):





When using the Train and test method, we consistently observed a lambda penalty of 10. When implementing five-fold cross-validation the optimal lambda becomes lesser around 9.40. Because one purpose of applying a lambda penalty to a loss function is to prevent overfitting, It is reasonable to expect a penalty associated with the train and test method as well as the five-fold cross-validation method. The visualizations below help to uncover the consistency of the seven-feature model and that performance was relatively unwavering when different penalties were used.

Symbolic Regression

Symbolic regression seeks to discover an equation in terms of the input features that accurately describe the relationship between input variables and an output variable. The best performance method for the Symbolic Regression model was the Five Fold Cross validation method. This method consistently delivered R Squared values that exceeded/surrounded .6. Using ScalaTion yielded better results, as our R-Squared metrics were around .87. Using Python, both the in-sample method as well as the Train and Test method were too inconsistent to be useful. R Squared metrics for either model could range from -.02 to .6.

Symbolic Ridge Regression

For Python, Symbolic Ridge Regression was not implemented as there is no functionality to apply a range of lambda penalties to a loss function for symbolic regression. Using Symbolic Regression in ScalaTion we found that cylinders, model year, weight, horsepower, and displacement were the best features. Our R-Squared metrics were around .8.

Transformed Regression

Transformed regression is a type of linear regression that involves transforming the dependent variable or the independent variables to improve the fit of the model. Transformed regression is used when the relationship between a model's input features and the output variable is not linear. Common transformations of the dependent variable include logarithmic or power transformations

The first transformation applied was a natural log transformation. Using the in sample method, this transformation elicited the best performance metrics as the distribution of the output variable, mpg, became more linear and more normally distributed. Unexpectedly, using the Train and Test method, we received worse results. The performance of the transformed regression in ScalaTion was also our best model for autompg.

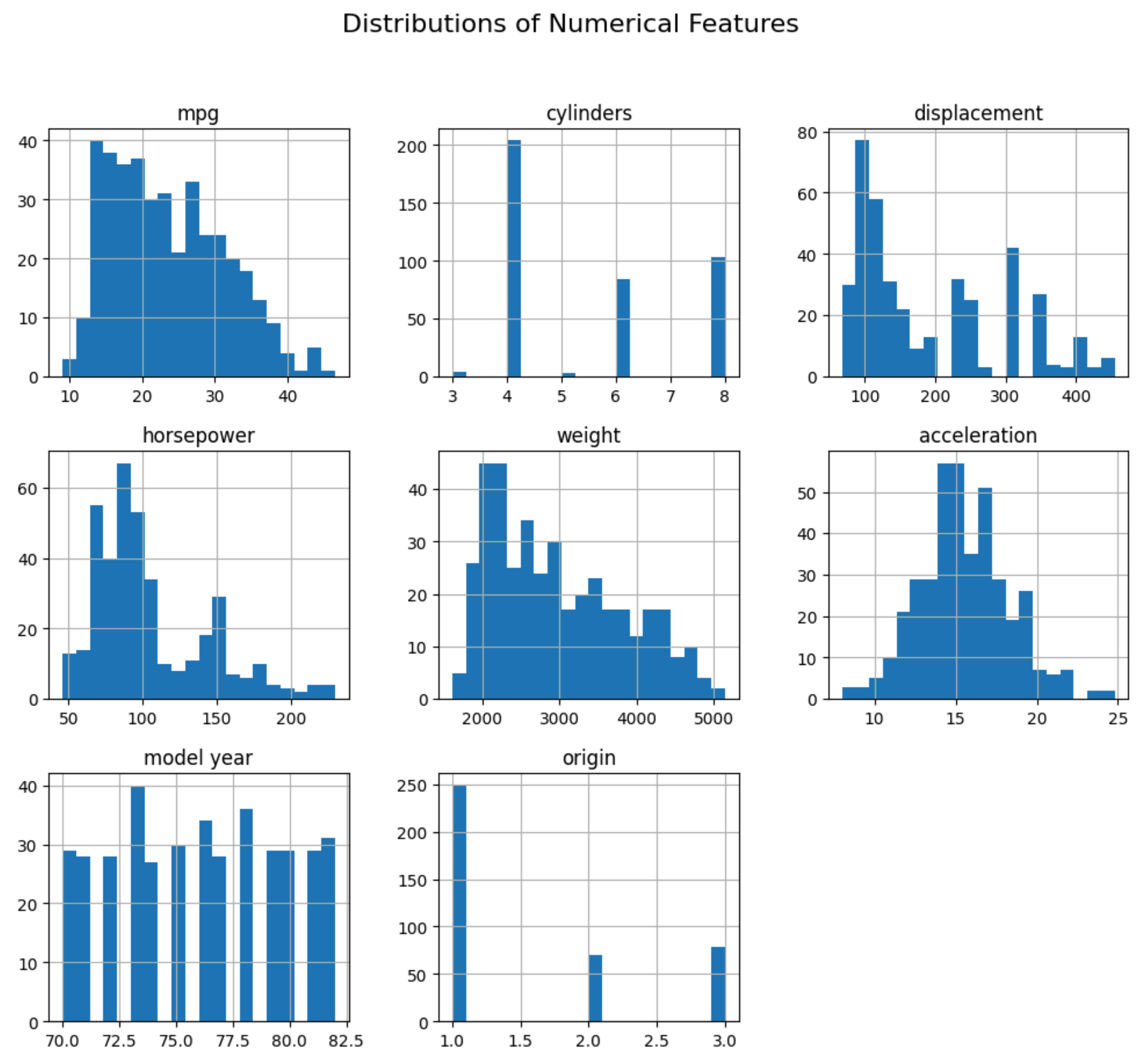
Data preprocessing & Exploratory Data Analysis

Within the Auto-MPG data set there are only 6 NaN’s found in the horse power column. In order to clean this up we chose to use a K-Nearest Neighbors Imputer with the number of neighbors set to 5. As well as dropping the column “Car Name” as there are many unique entries and it is a categorical variable. After this, we consider the data set cleaned and can start the EDA.

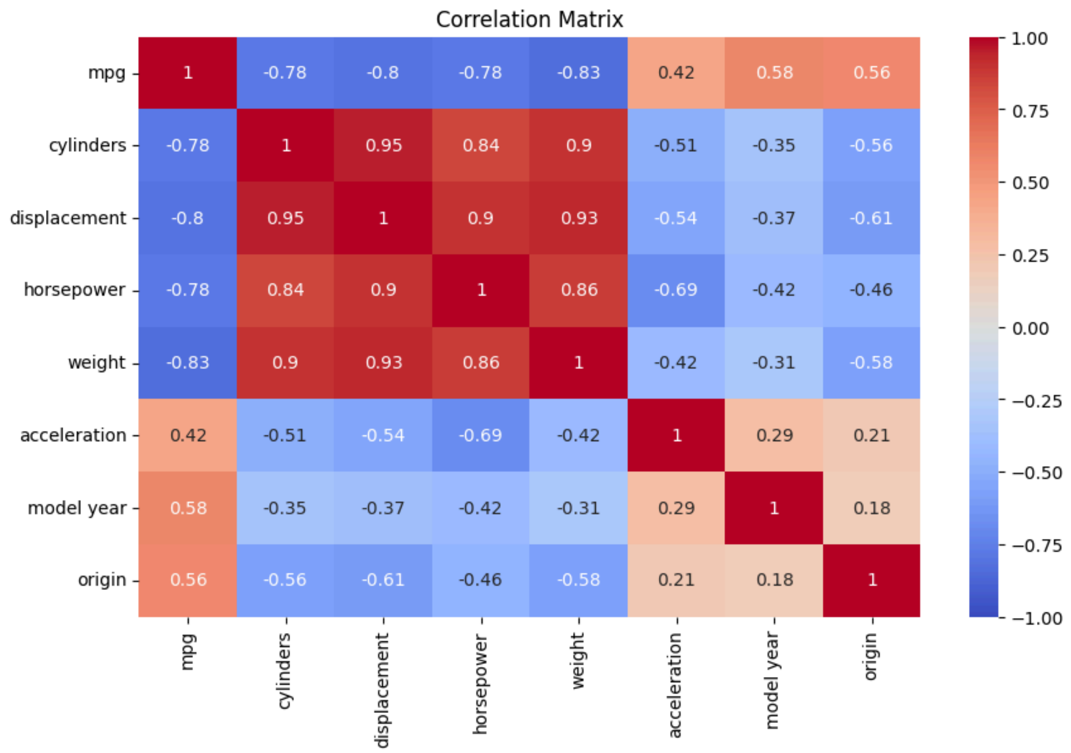
Getting some basic information from our cleaned dataset:

	mpg	cylinders	displacement	horsepower	weight	acceleration	Model year	origin
count	398	398	398	392.0	398	398	398	398
mean	23.515	5.455	193.426	104.469	2970.425	15.568	76.010	1.572
std	7.816	1.701	104.270	38.491	846.842	2.758	3.698	0.802
min	9.0	3.0	68.0	46.0	1613.0	8.0	70.0	1.0
25%	17.5	4.0	104.25	75.0	2223.75	13.825	73.0	1.0
50%	23.0	4.0	148.5	93.5	2803.5	15.5	76.0	1.0
75%	29.0	8.0	262.0	126.0	3608.0	17.175	79.0	2.0
max	46.6	8.0	455.0	230.0	5140.0	24.8	82.0	3.0

From the summary statistics we can see that the average mpg is 23.515 with a standard deviation of 7.816, the data covers a spread of fuel efficiencies. The average weight is 2970.425 with a standard deviation of 846.842, and comparing the min and max values for weight we can see that the data includes engines of all sizes. Next we generate some histograms so we can see the distribution of values for each attribute.



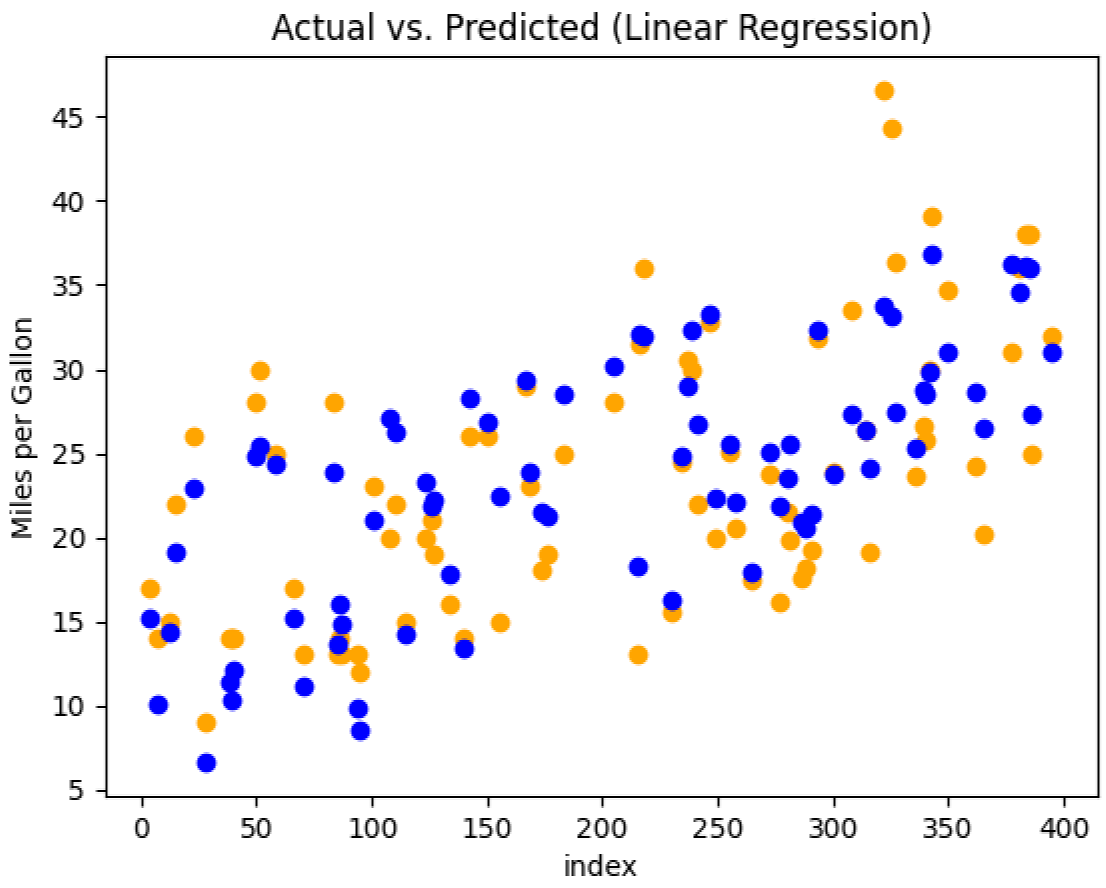
From the generated charts we can see the mpg, weight, and horsepower features all have a right skew. The feature origin does not have much distribution at all. While acceleration has a nice bell shaped distribution. Next we take a look at the Correlation Matrix to determine relationships between features.



According to the Correlation Matrix, We can see two groups of features that have positive correlation with each other. The larger group consists of features; cylinders, displacement, horsepower, and weight. While the second smaller group contains mpg, acceleration, model year, and origin. The feature mpg has a strong negative correlation to weight and displacement which means as engines get bigger and heavier the fuel efficiency goes down. As well as when the build of the car can accelerate faster, the fuel efficiency is also better.

In-Sample, Validation (80-20 train-test-split), 5x Cross-Validation

Across the three sampling methods the most performant model and testing method for the Auto-MPG dataset was Log Transformation Linear Regression with 5x Cross-Validation, with R^2 and Adjusted R^2 being 0.871 (+/- 0.051) and 0.869 (+/- 0.052) respectively. All sampling methods and models performed fairly well however.



MAE: 2.8322

MSE: 13.6196

Five Fold Cross Validation - Linear Regression - In Sample

Mean R-squared score: 0.805 (+/- 0.057)

Linear Regression Mean Adjusted R-squared score: 0.802 (+/- 0.058)

Forest Fires

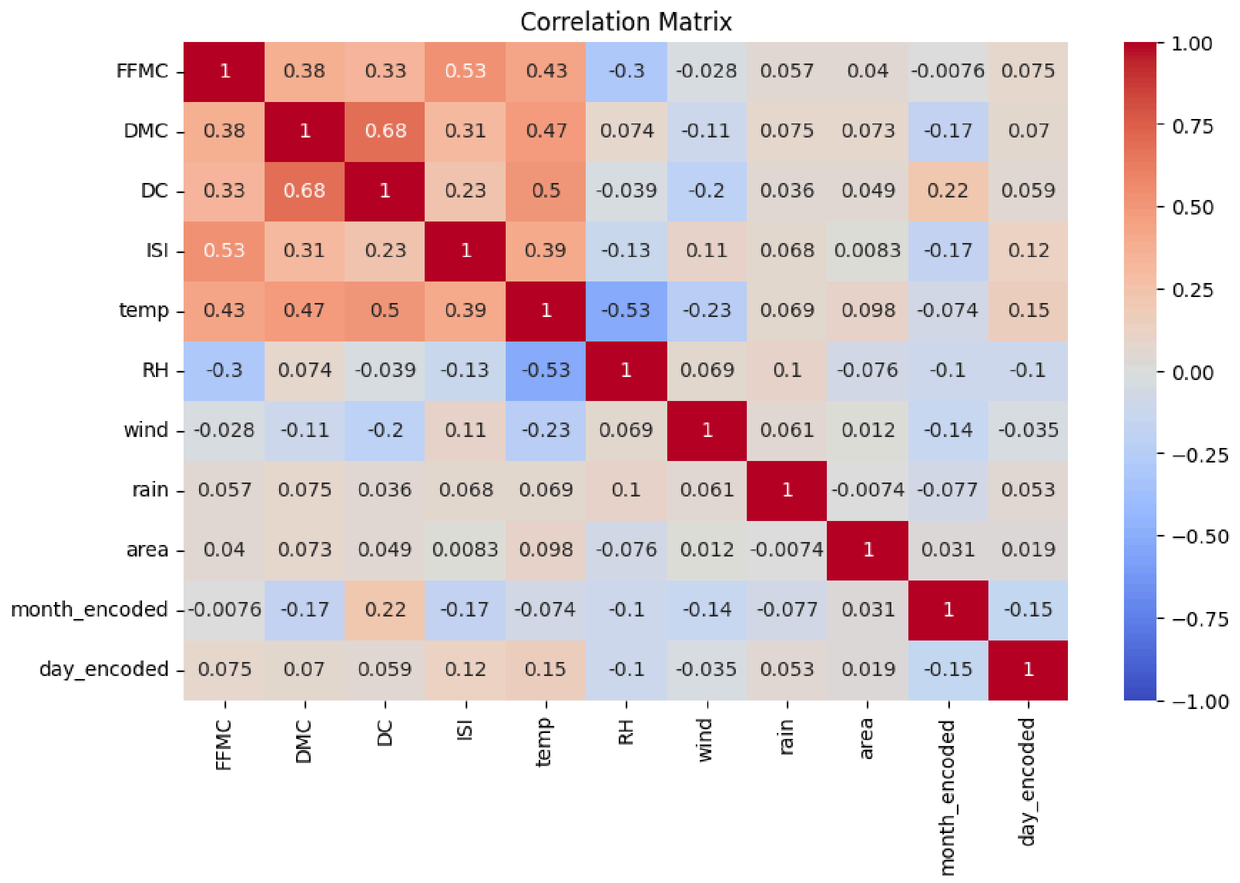
Data preprocessing & Exploratory Data Analysis

```
First few rows of the dataset:
   X  Y month  day FFMC DMC DC  ISI  temp  RH  wind  rain  area
0  7  5   mar  fri  86.2  26.2   94.3  5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6  35.4  669.1  6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6  43.7  686.9  6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7  33.3   77.5  9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3  51.3  102.2  9.6  11.4  99   1.8   0.0   0.0

Summary statistics:
                X          Y          FFMC           DMC            DC       ISI                    \
count  517.000000  517.000000  517.000000  517.000000  517.000000  517.000000
mean    4.669246     4.299807   90.644681  110.872340  547.940039    9.021663
std     2.313778     1.229900       5.520111   64.046482  248.066192    4.559477
min     1.000000     2.000000   18.7000001.100000       7.900000     0.000000
25%     3.000000     4.000000   90.200000   68.600000  437.700000     6.500000
50%     4.000000     4.000000   91.600000  108.300000  664.200000     8.400000
75%     7.000000     5.000000   92.900000  142.400000  713.900000   10.800000
max     9.000000     9.000000   96.200000  291.300000  860.600000   56.100000

        temp          RH          wind         rain           area
count  517.000000  517.000000  517.0000   517.00000       517.000000
mean    18.889168   44.288201   4.017602  0.021663      12.847292
std     5.806625   16.317469   1.791653  0.295959      63.655818
min     2.200000   15.0000000 .400000   0.000000      0.000000
25%     15.500000   33.000000   2.700000  0.000000      0.000000
50%     19.300000   42.000000   4.000000  0.000000      0.520000
75%     22.800000   53.000000   4.900000  0.000000      6.570000
max     33.300000  100.000000   9.400000  6.400000  1090.840000

Missing values in the dataset:
X       0
Y       0
month  0
day     0
FFMC  0
DMC   0
DC      0
ISI      0
temp   0
```

```
RH      0
wind    0
rain    0
area    0
dtype: int64

Data types of the columns:
X        int64
Y        int64
month   object
day     object
FFMC   float64
DMC    float64
DC     float64
ISI    float64
temp   float64
RH       int64
wind   float64
rain   float64
area   float64
dtype: object
```

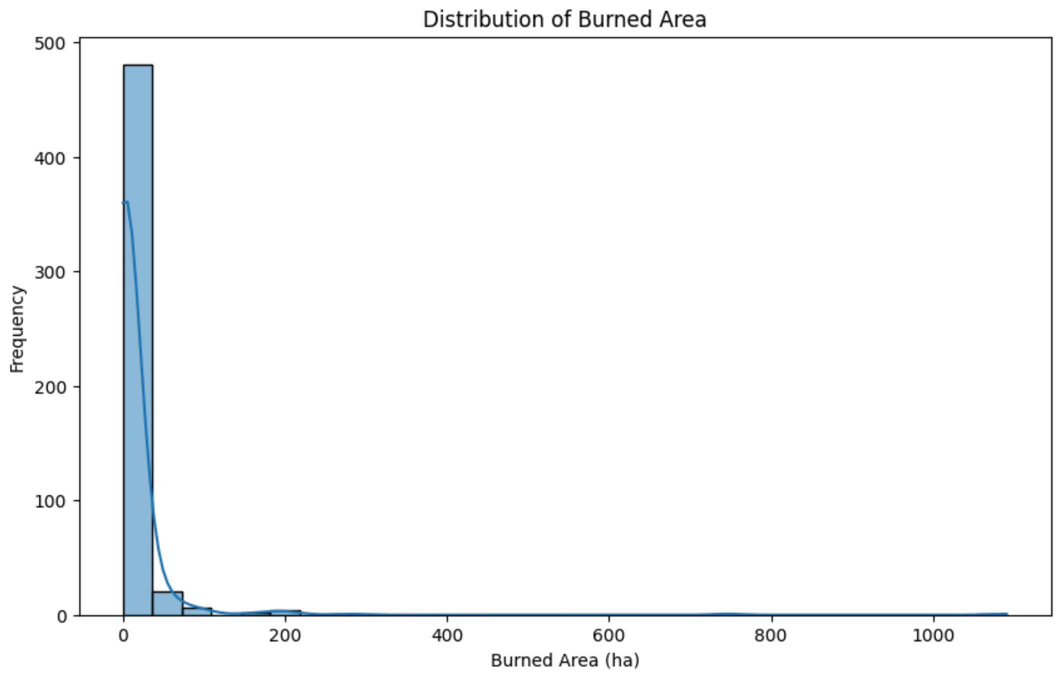
The Forest Fires Dataset contains 10 Predictor Features, to predict the area a forest fire covers. There are 517 rows of data none of which had missing values, however the columns “month” and “day” are categorical so we will use a label encoder to encode them into numbers.
This will allow the model to actually use the data. We then tested the data for outliers using z-score greater than 3 standard deviations away from the mean. This resulted in 30 rows being deemed outliers. The original dataset had 517 rows while after preprocessing and cleaning the new dataset has 487 rows.

Then we generated a correlation matrix.

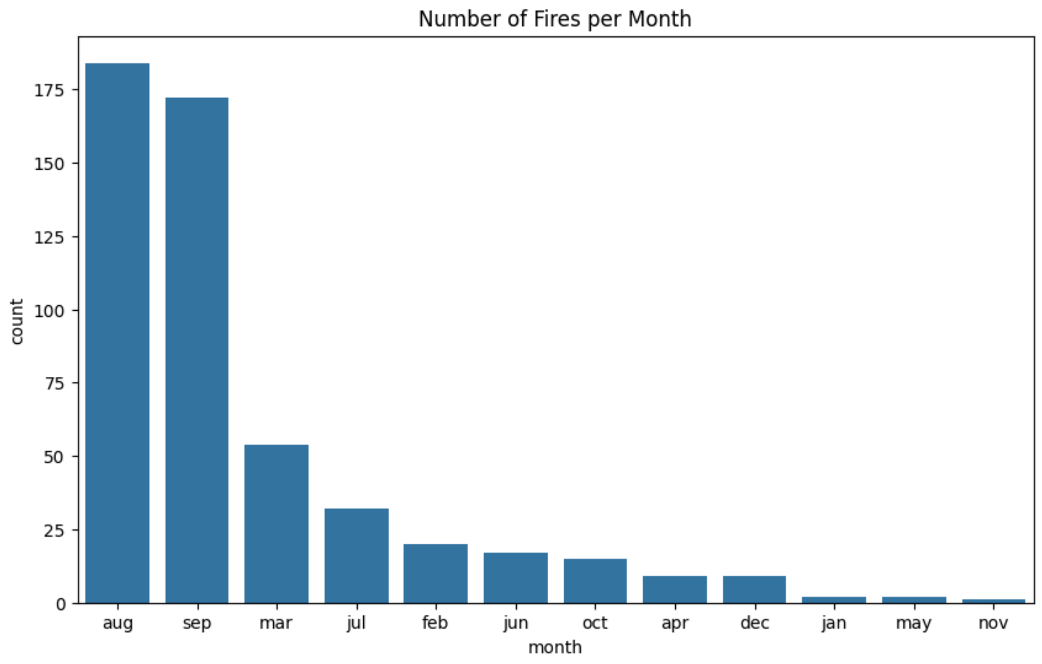


	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	month_encoded	day_encoded
FFMC	1	0.38	0.33	0.53	0.43	-0.3	-0.028	0.057	0.04	-0.0076	0.075
DMC	0.38	1	0.68	0.31	0.47	0.074	-0.11	0.075	0.073	-0.17	0.07
DC	0.33	0.68	1	0.23	0.5	-0.039	-0.2	0.036	0.049	0.22	0.059
ISI	0.53	0.31	0.23	1	0.39	-0.13	0.11	0.068	0.0083	-0.17	0.12
temp	0.43	0.47	0.5	0.39	1	-0.53	-0.23	0.069	0.098	-0.074	0.15
RH	-0.3	0.074	-0.039	-0.13	-0.53	1	0.069	0.1	-0.076	-0.1	-0.1
wind	-0.028	-0.11	-0.2	0.11	-0.23	0.069	1	0.061	0.012	-0.14	-0.035
rain	0.057	0.075	0.036	0.068	0.069	0.1	0.061	1	-0.0074	-0.077	0.053
area	0.04	0.073	0.049	0.0083	0.098	-0.076	0.012	-0.0074	1	0.031	0.019
month_encoded	-0.0076	-0.17	0.22	-0.17	-0.074	-0.1	-0.14	-0.077	0.031	1	-0.15
day_encoded	0.075	0.07	0.059	0.12	0.15	-0.1	-0.035	0.053	0.019	-0.15	1

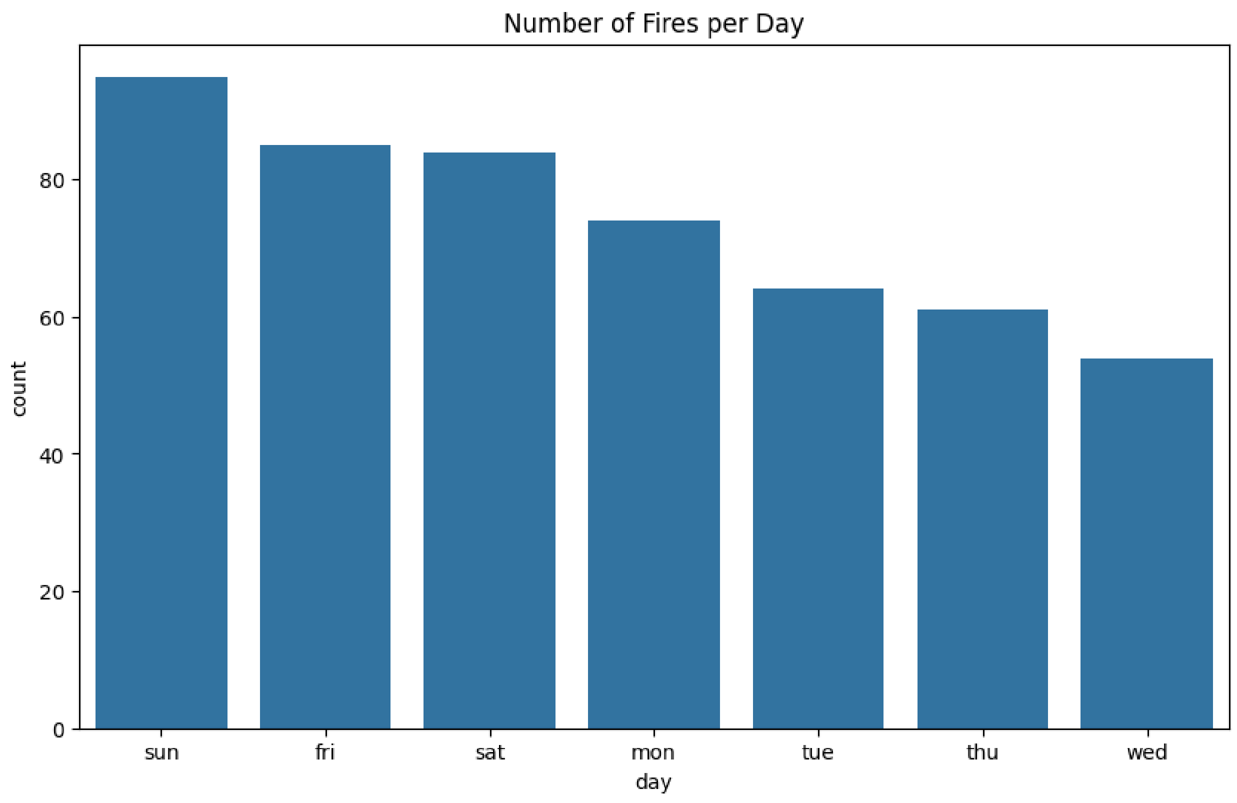
According to this matrix the strongest negative correlation is between Rain Humidity and Temperature, while the strongest correlation is between Drought Code and Duff Moisture Code. DMC is the numerical rating of the average moisture content of loosely compacted organic layers and DC is the numerical rating of the average moisture content of deep and compact layers of organic material.



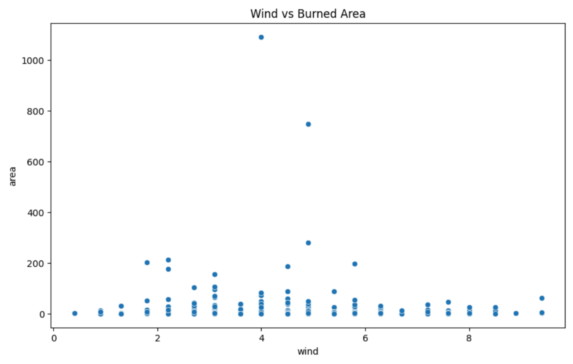
This graph shows that there is a great skew towards 0 for the predicted variable "Area" measured in hectares. Since it is heavily skewed some models might perform better than others.

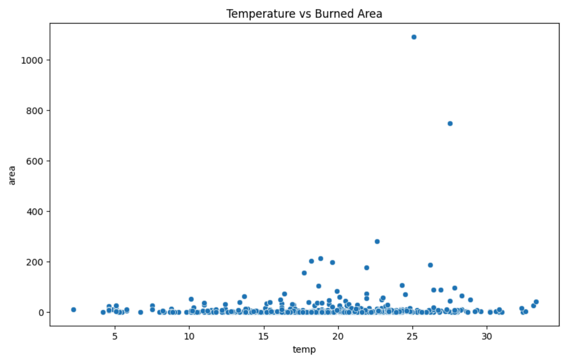


This graph shows the distribution of Forest Fires by Month. August and September have the highest counts. My assumption is that those months are the hottest months with the least amount of rainfall. While January and November are cold and dry. With May being a hot month with plenty of humidity, you know the saying April Shower brings May Flowers.

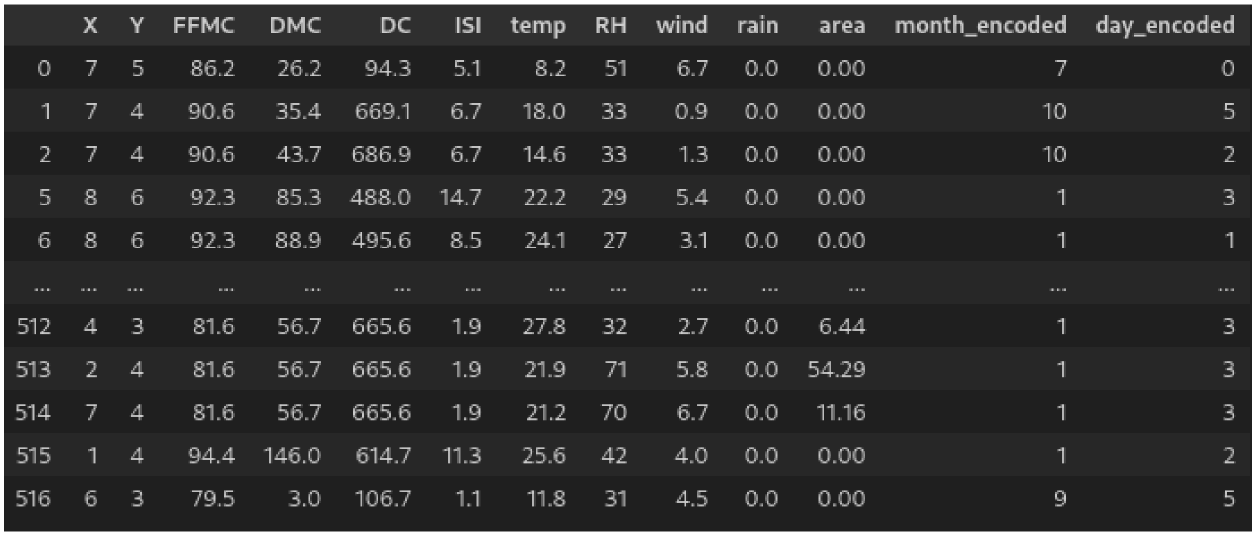


This graph shows distribution of Forest Fires by Days of the week.





These graphs clearly show why the correlation between Wind and Area is lower than the correlation between Temperature and Area. As temperature rises the burned area seems to rise. Same with wind up until it hits around the 5km/h mark then the burned area falls off.

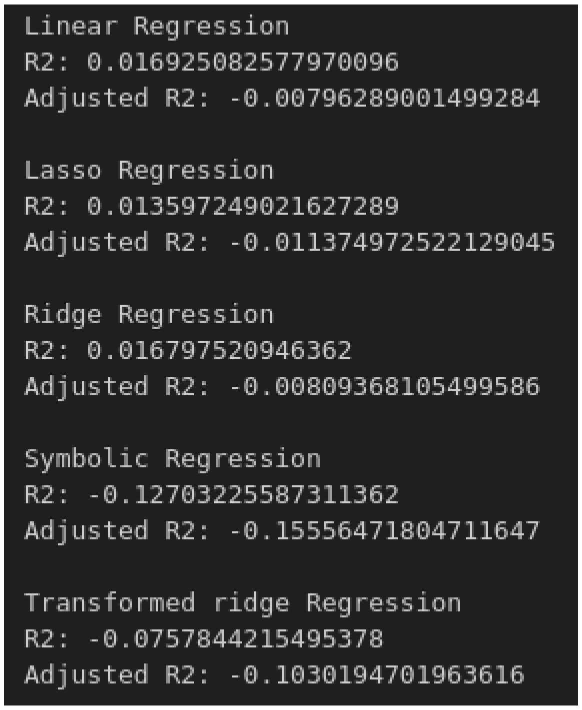


	X	Y	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	month_encoded	day_encoded
0	7	5	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.00	7	0
1	7	4	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.00	10	5
2	7	4	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.00	10	2
5	8	6	92.3	85.3	488.0	14.7	22.2	29	5.4	0.0	0.00	1	3
6	8	6	92.3	88.9	495.6	8.5	24.1	27	3.1	0.0	0.00	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
512	4	3	81.6	56.7	665.6	1.9	27.8	32	2.7	0.0	6.44	1	3
513	2	4	81.6	56.7	665.6	1.9	21.9	71	5.8	0.0	54.29	1	3
514	7	4	81.6	56.7	665.6	1.9	21.2	70	6.7	0.0	11.16	1	3
515	1	4	94.4	146.0	614.7	11.3	25.6	42	4.0	0.0	0.00	1	2
516	6	3	79.5	3.0	106.7	1.1	11.8	31	4.5	0.0	0.00	9	5

This is the describe() of the cleaned and encoded Forest Fires Dataset.

In-Sample, Validation (80-20 train-test-split), 5x Cross-Validation

Across the three sampling methods In-Sample testing with Linear Regression performed the best with the R^2 and Adjusted R^2s of .0169 and -.008 respectively. This is unsurprising that it performed the best as it is training on the data points it is tested over, however the low R^2 indicates that you cannot predict Forest Fire spread area from these predictor values.



```
Linear Regression
R2: 0.016925082577970096
Adjusted R2: -0.00796289001499284

Lasso Regression
R2: 0.013597249021627289
Adjusted R2: -0.011374972522129045

Ridge Regression
R2: 0.016797520946362
Adjusted R2: -0.00809368105499586

Symbolic Regression
R2: -0.12703225587311362
Adjusted R2: -0.15556471804711647

Transformed ridge Regression
R2: -0.0757844215495378
Adjusted R2: -0.1030194701963616
```

Seoul Bike Sharing

Dataset does not have any identifier columns, Null values, hence no imputations are done. However, the dataset contains 3 categorical columns (Seasons, Holiday and Functioning day), these are converted into categorical codes. The Hour column is converted into categorical by cutting into Morning, afternoon, evening and night.

Data preprocessing & Exploratory Data Analysis

```
Date                        object
Rented Bike Count           int64
Hour                        int64
Temperature(°C)             float64
Humidity(%)                 int64
Wind speed (m/s)            float64
Visibility (10m)            int64
Dew point temperature(°C)   float64
Solar Radiation (MJ/m2)     float64
Rainfall(mm)                float64
Snowfall (cm)               float64
Seasons                     object
Holiday                     object
Functioning Day             object
dtype: object
```

The Seoul Bike Sharing dataset consists of 13 predictor variables to be used to predict Rented Bike Count in Seoul. The dataset is pretty large with 8760 rows of data. The data contained no NaN values, however we transform the "hour" variable from an int 0-23 to a format of Early Morning, Morning, Afternoon, Evening, and Night. We also drop the "date" column as there is no use in predicting for a day that will never happen again. We also encode the "Seasons", "Holiday", and "Functioning Day" variables. Then we calculated z-scores for each data point to highlight any outliers.
None of the outliers seemed to actually be outliers so we do not drop any more rows.

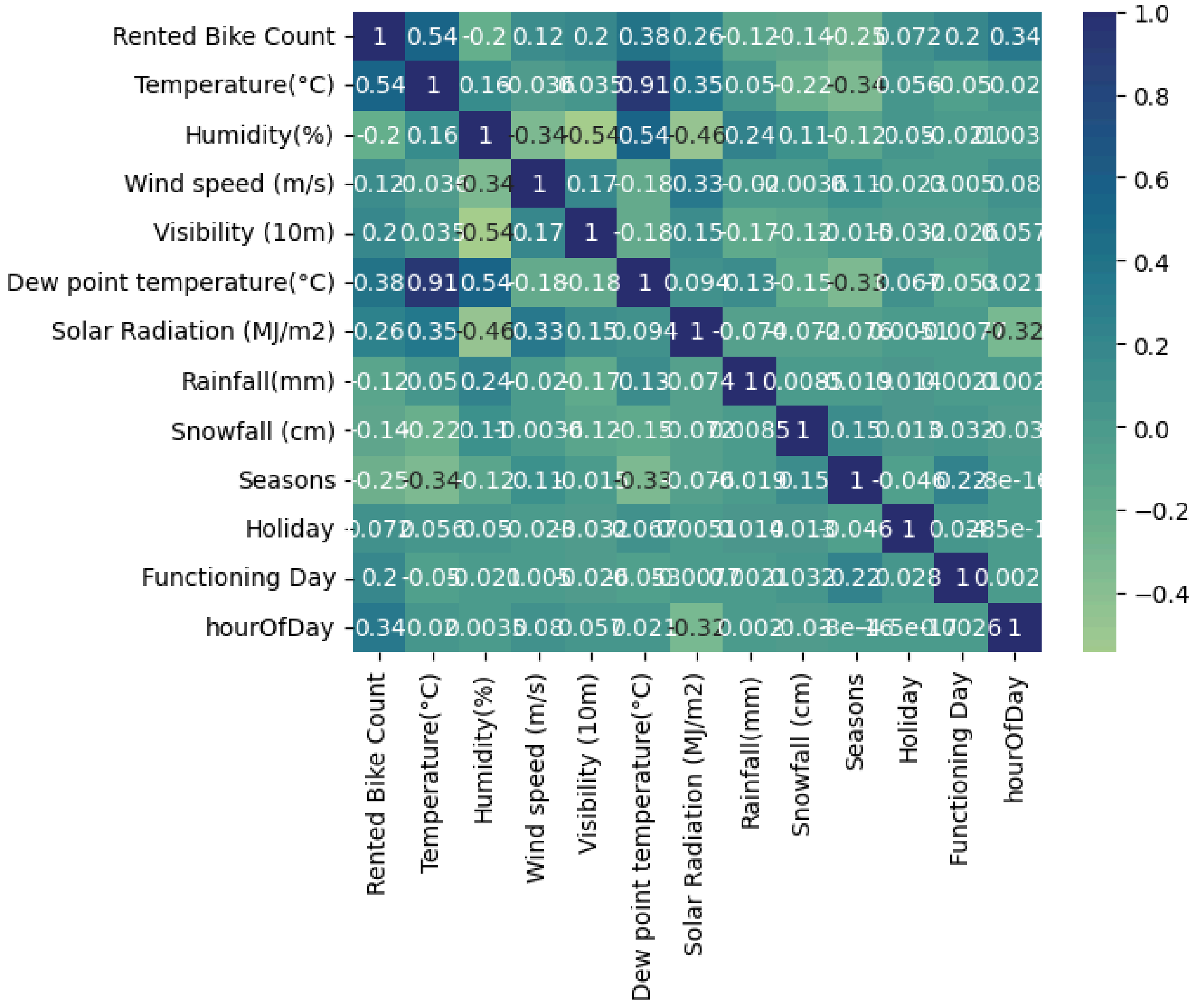
Then we calculated variance, covariance and multicollinearity.

```
                     Variance
Rented Bike Count         416021.733390
```

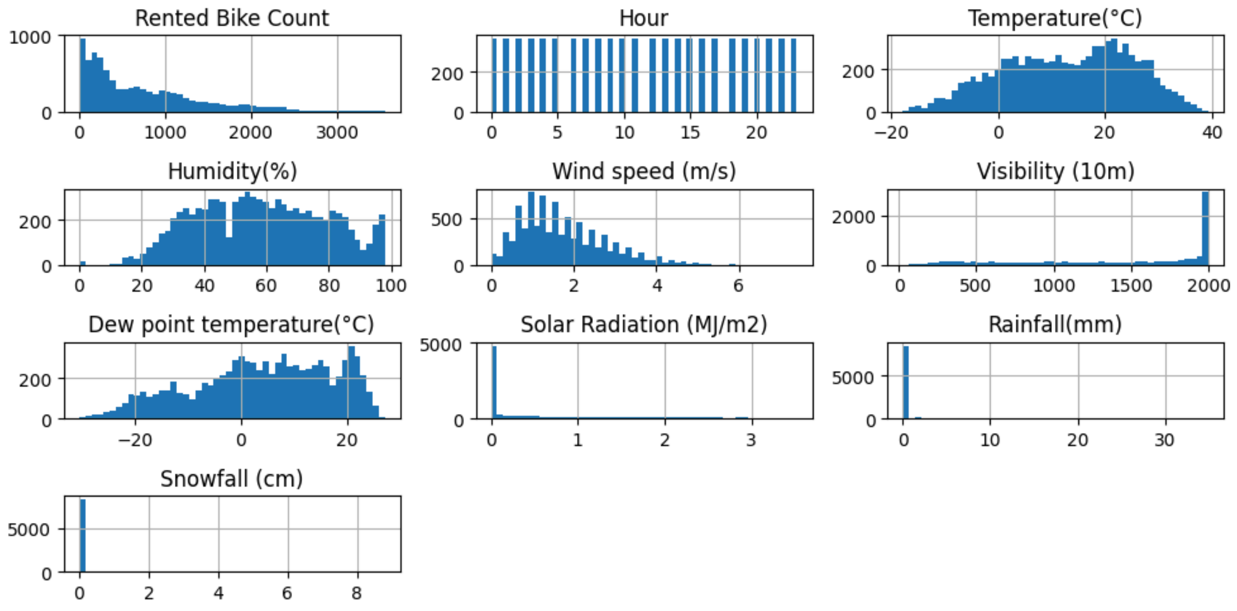
```
Temperature(°C)              142.678850
Humidity(%)                  414.627875
Wind speed (m/s)               1.073918
Visibility (10m)          370027.323001
Dew point temperature(°C)    170.573247
Solar Radiation (MJ/m2)        0.754720
Rainfall(mm)                   1.272819
Snowfall (cm)                  0.190747
Seasons                        1.241906
Holiday                        0.046888
Functioning Day                0.032545
hourOfDay                      2.771150

                          Rented Bike Count Temperature(°C)  Humidity(%)  \
Rented Bike Count              416021.73339     4149.257754 -2623.853782
Temperature(°C)                 4149.257754       142.67885    38.763038
Humidity(%)                    -2623.853782       38.763038   414.627875
Wind speed (m/s)                  80.950203       -0.448739     -7.10454
Visibility (10m)               78187.849382      252.817084 -6726.950421
Dew point temperature(°C)       3199.299111      142.400017   142.782065
Solar Radiation (MJ/m2)          146.717508        3.668334    -8.171237
Rainfall(mm)                     -89.558657        0.677602     5.430677
Snowfall (cm)                    -39.946114       -1.139387     0.962098
Seasons                         -181.895345       -4.462075     -2.73009
Holiday                           10.103104        0.144665     0.221685
Functioning Day                   23.730746        -0.10811    -0.076408
hourOfDay                        361.557769        0.397705     0.117536

                          Wind speed (m/s) Visibility (10m)  \
Rented Bike Count                80.950203     78187.849382
Temperature(°C)                  -0.448739       252.817084
Humidity(%)                       -7.10454     -6726.950421
Wind speed (m/s)                  1.073918        108.11466
Visibility (10m)                 108.11466    370027.323001
Dew point temperature(°C)        -2.388639     -1403.253586
Solar Radiation (MJ/m2)            0.29914        79.130141
Rainfall(mm)                     -0.023002      -115.040313
Snowfall (cm)                    -0.001609       -32.330842
Seasons                           0.125824         -10.0167
Holiday                          -0.005165        -4.185096
```

```
Functioning Day              0.000942       -2.853224
hourOfDay                   0.13719       58.075751

                    Dew point temperature(°C) Solar Radiation (MJ/m2)  \
Rented Bike Count                    3199.299111           146.717508
Temperature(°C)                      142.400017            3.668334
Humidity(%)                        142.782065            -8.171237
Wind speed (m/s)                      -2.388639             0.29914
Visibility (10m)                 -1403.253586           79.130141
Dew point temperature(°C)             170.573247             1.070865
Solar Radiation (MJ/m2)               1.070865              0.75472
Rainfall(mm)                         1.85062            -0.072813
Snowfall (cm)                      -0.860668            -0.027432
Seasons                            -4.788851            -0.073396
Holiday                           0.188799            0.000955
Functioning Day                      -0.124492            -0.001201
hourOfDay                          0.459316            -0.462553

                    Rainfall(mm) Snowfall (cm)     Seasons    Holiday  \
Rented Bike Count          -89.558657    -39.946114 -181.895345  10.103104
Temperature(°C)             0.677602     -1.139387   -4.462075   0.144665
Humidity(%)               5.430677      0.962098    -2.73009   0.221685
Wind speed (m/s)           -0.023002     -0.001609    0.125824  -0.005165
Visibility (10m)         -115.040313    -32.330842    -10.0167  -4.185096
Dew point temperature(°C)      1.85062     -0.860668   -4.788851   0.188799
Solar Radiation (MJ/m2)     -0.072813     -0.027432  -0.073396   0.000955
Rainfall(mm)              1.272819      0.004188    -0.02426   0.003486
Snowfall (cm)             0.004188      0.190747    0.070796   0.001191
Seasons                 -0.02426      0.070796    1.241906  -0.011163
Holiday                0.003486      0.001191   -0.011163   0.046888
Functioning Day            0.000418      0.002528    0.044901   0.001079
hourOfDay                 0.003765       -0.022       -0.0       0.0

                    Functioning Day   hourOfDay
Rented Bike Count              23.730746  361.557769
Temperature(°C)               -0.10811    0.397705
Humidity(%)                -0.076408    0.117536
Wind speed (m/s)                0.000942     0.13719
Visibility (10m)             -2.853224   58.075751
Dew point temperature(°C)       -0.124492    0.459316
```

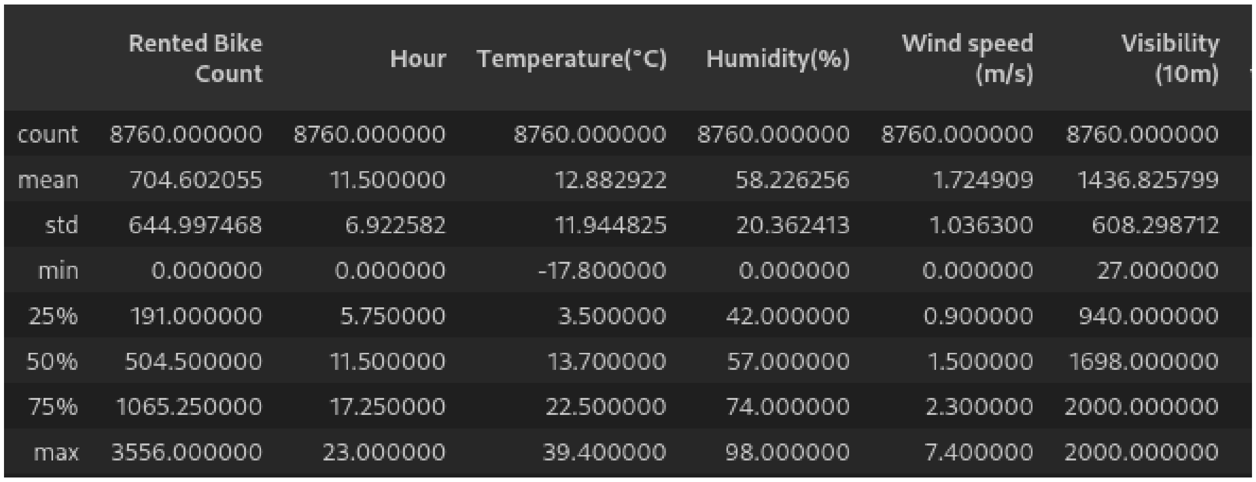
```
Solar Radiation (MJ/m2)        -0.001201   -0.462553
Rainfall(mm)                  0.000418    0.003765
Snowfall (cm)                 0.002528      -0.022
Seasons                      0.044901       -0.0
Holiday                     0.001079        0.0
Functioning Day                0.032545    0.000771
hourOfDay                    0.000771     2.77115

                   feature        VIF
0           Rented Bike Count   5.087292
1             Temperature(°C)  51.250818
2                 Humidity(%)  28.527863
3            Wind speed (m/s)   4.772358
4            Visibility (10m)   9.125990
5   Dew point temperature(°C)  29.877470
6     Solar Radiation (MJ/m2)   3.595171
7                Rainfall(mm)   1.099297
8               Snowfall (cm)   1.129091
9                     Seasons   3.564110
10                    Holiday  19.748355
11            Functioning Day  34.589208
12                  hourOfDay   4.395375
```



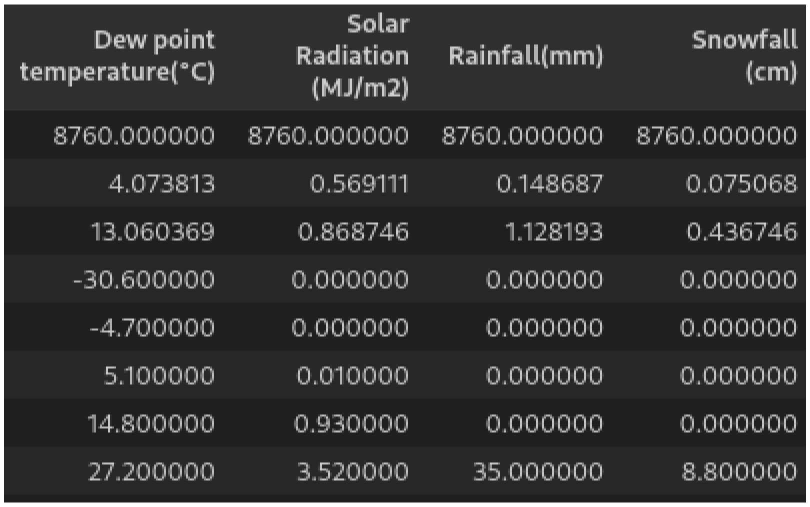
We created a correlation matrix to assess which features would be the most promising when selecting later down the line. When we calculated the VIF we saw that “Temperature” and “Functioning Day” had the highest scores. The features with highest correlation to the prediction feature are “Temperature” and “HourOfDay” while “Temperature” and “Dew Point Temp” have a correlation of .91.



This graph shows the distribution of the data for each individual feature. We can see Snowfall, Solar Radiation, Rainfall, Rented bike count, and Wind speed all have a right skew. With visibility and Dew Point Temp being left skewed. Humidity and Temperature have a nice bell shape.



	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)
count	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000
mean	704.602055	11.500000	12.882922	58.226256	1.724909	1436.825799
std	644.997468	6.922582	11.944825	20.362413	1.036300	608.298712
min	0.000000	0.000000	-17.800000	0.000000	0.000000	27.000000
25%	191.000000	5.750000	3.500000	42.000000	0.900000	940.000000
50%	504.500000	11.500000	13.700000	57.000000	1.500000	1698.000000
75%	1065.250000	17.250000	22.500000	74.000000	2.300000	2000.000000
max	3556.000000	23.000000	39.400000	98.000000	7.400000	2000.000000



Dew point temperature(°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)
8760.000000	8760.000000	8760.000000	8760.000000
4.073813	0.569111	0.148687	0.075068
13.060369	0.868746	1.128193	0.436746
-30.600000	0.000000	0.000000	0.000000
-4.700000	0.000000	0.000000	0.000000
5.100000	0.010000	0.000000	0.000000
14.800000	0.930000	0.000000	0.000000
27.200000	3.520000	35.000000	8.800000

A lot of the features in this data set are heavily skewed so the description of the DataFrame looks problematic under certain features like the last 4 features Dew Point, Solar Radiation, Rainfall, and Snowfall. They all have standard deviations greater than mean.

In-Sample, Validation (80-20 train-test-split), 5x Cross-Validation

Across the three sampling methods 80-20 TTS testing with Linear Regression performed the best with the R^2 and Adjusted R^2s of .5682 and .5674 respectively. This is a little shocking since it performed marginally better than In-Sample Linear Regression.

```
Linear Regression
R2: 0.5674513444814331
Adjusted R2: 0.5669074447088331

Lasso Regression
R2: 0.5672949656297792
Adjusted R2: 0.5667508692216776

Ridge Regression
R2: 0.5674504571723655
Adjusted R2: 0.5669065562840363

Symbolic Regression
R2: 0.3133610711642624
Adjusted R2: 0.31249767059073785

Transformed ridge Regression
R2: 0.4952383163623114
Adjusted R2: 0.49460361374228234
```

Airfoil self-noise

Data preprocessing & Exploratory Data Analysis

The data set "airfoil_self_noise.csv" consists of 5 predictor variables to be used to predict decibels produced by an airfoil blade in a wind tunnel. The data set was somewhat large with 1503 rows of data. None of the data had NaN values so there were no columns removed or data imputed because of bad data collection or management. We also ran a z-score for each data point to determine if there were any outliers. The threshold for our z-score was based on being further than 3 standard deviations away from the mean. Columns, frequency and displacement contained numerous outliers based on our criteria. On further investigation none of the values seemed outlandish, so at this point I did not remove any outliers.

Outliers according to Z score

```
Frequency      0
Angle          0
Chord          0
Velocity       0
Displacement   0
Decibels       0
dtype: int64
------------------------------------------------
Outliers according to Z score
Column name: Frequency
     Frequency  Angle   Chord  Velocity  Displacement  Decibels
12       12500    0.0  0.3048      71.3      0.002663   112.241
13       16000    0.0  0.3048      71.3      0.002663   108.721
28       12500    0.0  0.3048      55.5      0.002831   111.076
47       12500    0.0  0.3048      39.6      0.003101   109.619
78       12500    1.5  0.3048      71.3      0.003367   109.222
79       16000    1.5  0.3048      71.3      0.003367   106.582
96       12500    1.5  0.3048      39.6      0.003921   106.111
173      12500    4.0  0.3048      71.3      0.004978   107.405
198      12500    0.0  0.2286      71.3      0.002143   117.624
199      16000    0.0  0.2286      71.3      0.002143   114.984
200      20000    0.0  0.2286      71.3      0.002143   114.474
313      12500    4.0  0.2286      71.3      0.004006   114.258
314      16000    4.0  0.2286      71.3      0.004006   112.768
315      20000    4.0  0.2286      71.3      0.004006   109.638
466      12500    0.0  0.1524      71.3      0.001599   117.087
```

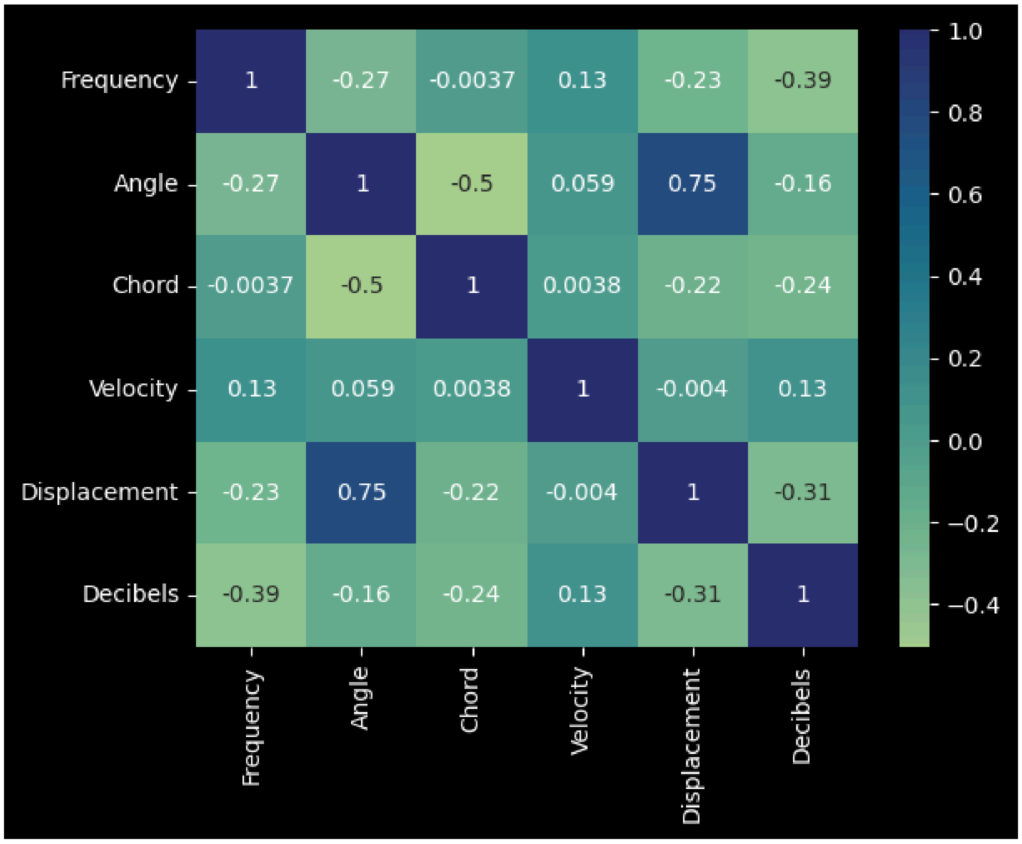
```
467       16000    0.0  0.1524      71.3       0.001599   113.297
482       12500    0.0  0.1524      55.5       0.001727   117.093
483       16000    0.0  0.1524      55.5       0.001727   112.803
524       12500    2.7  0.1524      71.3       0.002439   113.034
525       16000    2.7  0.1524      71.3       0.002439   110.364
540       12500    2.7  0.1524      39.6       0.002948   108.649
737       12500    0.0  0.0508      71.3       0.000740   125.220
750       12500    0.0  0.0508      55.5       0.000762   125.376
763       12500    0.0  0.0508      39.6       0.000792   123.988
776       12500    0.0  0.0508      31.7       0.000812   119.070
786       12500    4.2  0.0508      71.3       0.001428   117.328
969       12500    0.0  0.0254      71.3       0.000401   133.207
970       16000    0.0  0.0254      71.3       0.000401   130.477
971       20000    0.0  0.0254      71.3       0.000401   123.217
980       12500    0.0  0.0254      55.5       0.000412   131.453
981       16000    0.0  0.0254      55.5       0.000412   125.683
982       20000    0.0  0.0254      55.5       0.000412   121.933
992       12500    0.0  0.0254      39.6       0.000428   129.116
993       16000    0.0  0.0254      39.6       0.000428   124.986
1005      12500    0.0  0.0254      31.7       0.000439   128.977
1006      16000    0.0  0.0254      31.7       0.000439   125.627
1015      12500    4.8  0.0254      71.3       0.000849   127.688
1016      16000    4.8  0.0254      71.3       0.000849   124.208
1017      20000    4.8  0.0254      71.3       0.000849   119.708
1027      12500    4.8  0.0254      55.5       0.000873   125.094
1028      16000    4.8  0.0254      55.5       0.000873   124.394
1029      20000    4.8  0.0254      55.5       0.000873   121.284
1253      12500    0.0  0.1016      71.3       0.001211   119.375
1267      12500    0.0  0.1016      55.5       0.001320   119.910

Column name: Angle
No outliers

Column name: Chord
No outliers

Column name: Velocity
No outliers

Column name: Displacement
     Frequency  Angle   Chord  Velocity  Displacement  Decibels
709        200   12.6  0.1524      39.6      0.058411   114.750
710        250   12.6  0.1524      39.6      0.058411   115.890
711        315   12.6  0.1524      39.6      0.058411   116.020
```

```
712        400   12.6  0.1524      39.6      0.058411   115.910
713        500   12.6  0.1524      39.6      0.058411   114.900
714        630   12.6  0.1524      39.6      0.058411   116.550
715        800   12.6  0.1524      39.6      0.058411   116.560
716       1000   12.6  0.1524      39.6      0.058411   114.670
717       1250   12.6  0.1524      39.6      0.058411   112.160
718       1600   12.6  0.1524      39.6      0.058411   110.780
719       2000   12.6  0.1524      39.6      0.058411   109.520
720       2500   12.6  0.1524      39.6      0.058411   106.880
721       3150   12.6  0.1524      39.6      0.058411   106.260
722       4000   12.6  0.1524      39.6      0.058411   104.500
723       5000   12.6  0.1524      39.6      0.058411   104.130
724       6300   12.6  0.1524      39.6      0.058411   103.380
1487       200   15.6  0.1016      39.6      0.052849   123.514
1488       250   15.6  0.1016      39.6      0.052849   124.644
1489       315   15.6  0.1016      39.6      0.052849   122.754
1490       400   15.6  0.1016      39.6      0.052849   120.484
1491       500   15.6  0.1016      39.6      0.052849   115.304
1492       630   15.6  0.1016      39.6      0.052849   118.084
1493       800   15.6  0.1016      39.6      0.052849   118.964
1494      1000   15.6  0.1016      39.6      0.052849   119.224
1495      1250   15.6  0.1016      39.6      0.052849   118.214
1496      1600   15.6  0.1016      39.6      0.052849   114.554
1497      2000   15.6  0.1016      39.6      0.052849   110.894
1498      2500   15.6  0.1016      39.6      0.052849   110.264
1499      3150   15.6  0.1016      39.6      0.052849   109.254
1500      4000   15.6  0.1016      39.6      0.052849   106.604
1501      5000   15.6  0.1016      39.6      0.052849   106.224
1502      6300   15.6  0.1016      39.6      0.052849   104.204

Column name: Decibels
    Frequency  Angle   Chord  Velocity  Displacement  Decibels
723      5000   12.6  0.1524      39.6      0.058411    104.13
724      6300   12.6  0.1524      39.6      0.058411    103.38

--------------------------------------------------
```

We then went ahead and checked variance, covariance, and for multicollinearity. Variance of our data varied greatly all the way from 9.94e6 in the frequency column to 1.73e-4 in the displacement column. The close to 0 variance in displacement seems worrying but checking our mean of 0.0111 it does not seem unusual. Covariance values between variables did not show anything worrying or to lead me to believe that there was any multicollinearity within our data set.

```
               Variance
Frequency     9.938717e+06
Angle         3.502424e+01
Chord         8.749868e-03
Velocity      2.425116e+02
Displacement  1.729287e-04
Decibels      4.759146e+01
--------------------------------------------------------------------
               Frequency       Angle     Chord     Velocity Displacement
Frequency     9938717.383697 -5089.05844   -1.0795  6562.137785    -9.539578  \
Angle          -5089.05844   35.024241 -0.279488     5.415383     0.058633
Chord              -1.0795   -0.279488   0.00875     0.005516    -0.000272
Velocity       6562.137785    5.415383  0.005516   242.511614    -0.000814
Displacement     -9.539578    0.058633 -0.000272    -0.000814     0.000173
Decibels      -8497.394774   -6.373423 -0.152396    13.439956    -0.028365

              Decibels
Frequency    -8497.394774
Angle         -6.373423
Chord         -0.152396
Velocity      13.439956
Displacement    -0.028365
Decibels       47.591463
```

We ran a correlation matrix using the seaborn statistical package. Most of our predictor variables negatively correlated with our y variable, with decibels (-0.39) and displacement (-0.31) having the strongest correlations. It is likely that these features will be included in our best model.



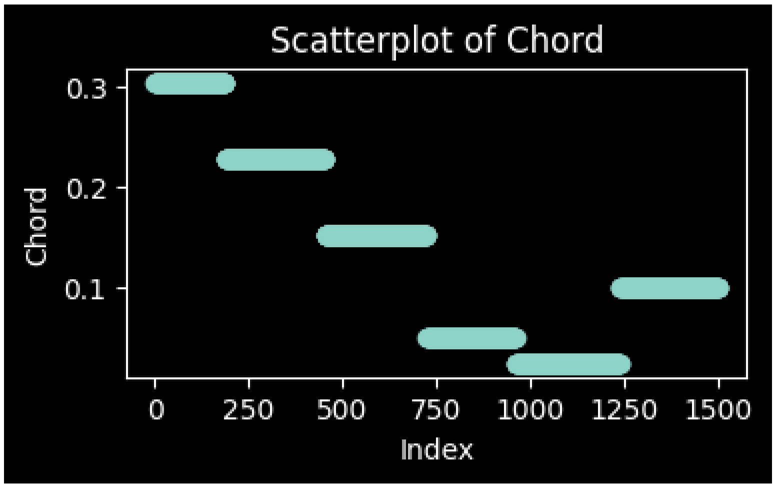
To check for multicollinearity VIF was performed on each feature column. According to VIF scores none of our columns seemed to indicate a particularly high level of multicollinearity. It appeared that the value that we were trying to predict (decibels) seemed to contain the highest degree of multicollinearity at 16.66, which may lead to trouble with our analysis down the road.

```
        feature        VIF
0     Frequency   1.990306
1         Angle   7.655110
2         Chord   4.270062
3      Velocity  12.834525
4  Displacement   4.392963
5      Decibels  16.664049
```

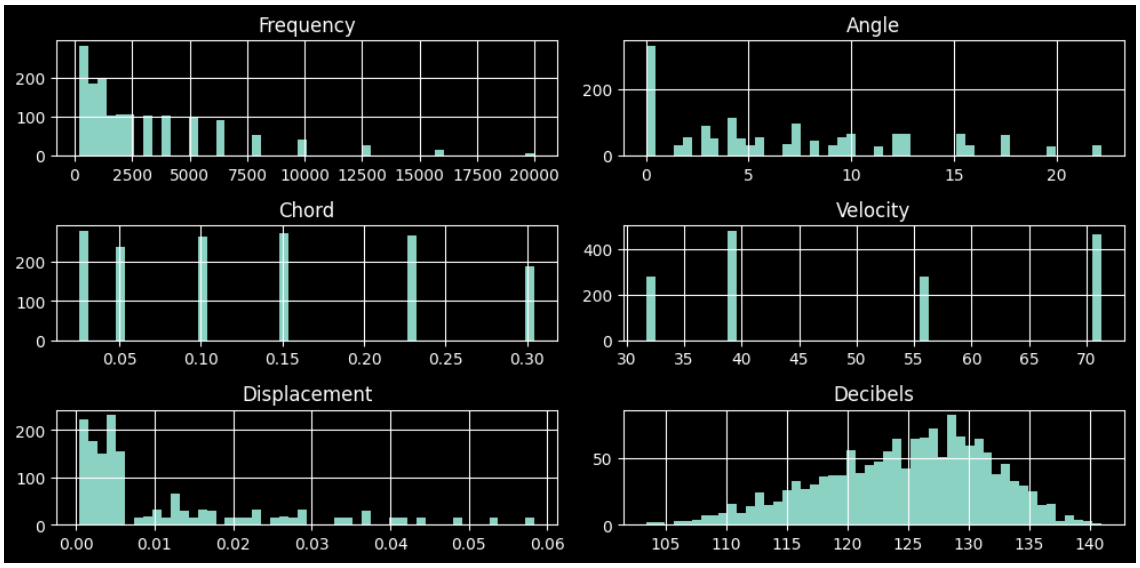
In this portion of the analysis we also performed several transformations of our y variable such as log10, log2, natural log, and square root to use in transformation regression.

Exploratory Data Analysis:

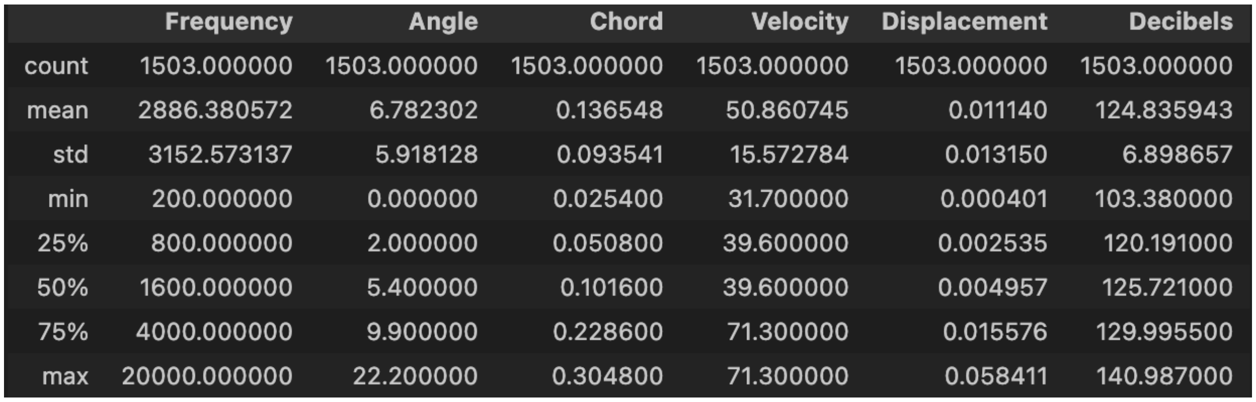
Scatter plots and histograms were constructed of all of the columns. Looking at the distribution of our scatter plots it appears that all variables are continuous, possibly with the exception of chord. At this point I will leave the chord column as is and continue on with analysis.



Moving on to the histograms none of the distribution of data looks close to normal. This may prove troublesome for linear regression down the road. Decibels, our y value, seems to be the only column with a close to normal distribution. Chord again does not seem to be continuous, with the addition of velocity, and might be good candidates to switch to categorical data.



Using the describe() function we can see some useful information about the data. Frequency stands out because it has a mean of 2886.38 and standard deviation of 3152.57. The third quartile has a value of 4000. This leads me to believe that because most of this data is within one standard deviation of the mean, that the highest values have significant weight on this column of data. There is a good chance this predictor variable won't be included in our best models or that some of these outliers should be removed during our analysis.



	Frequency	Angle	Chord	Velocity	Displacement	Decibels
count	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000
mean	2886.380572	6.782302	0.136548	50.860745	0.011140	124.835943
std	3152.573137	5.918128	0.093541	15.572784	0.013150	6.898657
min	200.000000	0.000000	0.025400	31.700000	0.000401	103.380000
25%	800.000000	2.000000	0.050800	39.600000	0.002535	120.191000
50%	1600.000000	5.400000	0.101600	39.600000	0.004957	125.721000
75%	4000.000000	9.900000	0.228600	71.300000	0.015576	129.995500
max	20000.000000	22.200000	0.304800	71.300000	0.058411	140.987000

In-Sample, Validation (80-20 train-test-split), 5x Cross-Validation

Across the three sampling methods our 80-20 train test split performed the best in terms of adjusted r-squared. This was surprising because I would expect in-sample validation to produce the best results because it is training and testing on the same data points, which should lead to an overfit model. I am not sure what the reason for this is. In Python linear, ridge, lasso, and symbolic regression was used for each sampling method.

In our in-sampling method linear regression performed the best with an r squared and adjusted r squared value of 0.516 and 0.514 respectively. For our symbolic regression we used the parameters:

```
SymbolicRegressor(population_size=5000, function_set=('add', 'sub', 'mul', 'div',
'sqrt', 'log', 'abs', 'neg', 'inv', 'max', 'min'), metric='rmse')
```

This produced the best results for us, but severely underperformed at an r squared and adjusted r squared value of 0.182 and 0.179 respectively.

Linear Regression

R2: 0.5157097420928731

Adjusted R2: 0.5140922061613196

Lasso Regression

R2: 0.45057075303655514

Adjusted R2: 0.4487356520112932

Ridge Regression

R2: 0.4885692672601716

Adjusted R2: 0.4868610817800786

Symbolic Regression

R2: 0.13656748607108637

Adjusted R2: 0.13368360993905937

Transformed Regression

R2: -1.6028270356952636e+274

Adjusted R2: -1.6081804994083406e+274

We tested transformation regression in the in-sampling method with transforming decibels, our y value, with log10, log2, natural log, and square root. The resulting r-squared and adjusted r-squared values were so poor we did not continue to use this method in other sampling methods.

Something that stuck out was how much worse our cross-validation models performed compared to our 80-20 train test split. Models increased r-squared and adjusted r-squared values anywhere from 0.05 to 0.07, which is a large jump. I am unsure of why the jump is so large because at 5 fold cross validation each subset should contain ~300 data rows. Maybe if our data set was larger the contrast between the two sampling methods would be reduced.

The best performing model across all sampling methods was linear regression with an r-squared value of 0.558 in 80-20 which was a small increase from in-sampling at 0.516. The flexibility and accuracy of this model lead us to believe that this is the best model for this data set. Considering how poor our symbolic regression performed here at an r squared value of 0.037, I wonder what parameters would be able to create a decent model or if there is an issue with gplearn's algorithm and our data set.

TnT Linear Regression

R2: 0.5582979754897279   Adjusted R2: 0.5564513951197018

```
TnT Lasso Regression
R2: 0.44938671435397204        Adjusted R2: 0.44708481934709066

TnT Ridge Regression
R2: 0.5075506158529834         Adjusted R2: 0.5054918809694258

TnT Symbolic Regression
R2: 0.03650757012260075
Adjusted R2: 0.032479591736825575
```

Statistical Summaries and Quality of fit

As mentioned previously linear regression was the best performing model at r-squared

and adjusted r-squared values of 0.558 and 0.556 respectively in Python. The flexibility and simplicity of this model paired with its high r-squared values leads me to believe that this is the best model to use for this data set and will be the model we use for our analysis from here on forward.

The next part of our analysis iterates through all the combinations of our predictor variables using linear regression and compares r-squared, adjusted r-squared, and cross validation r-squared values. The algorithm then reports what combination of parameters lead to the highest r-squared values. The best performing model used all features and produced the r squared values mentioned previously. The cross validation r squared values of this model was 0.488.

```
---------- Best performances below ---------- --------------------------------------------
Column combination
['Frequency', 'Angle', 'Chord', 'Velocity', 'Displacement'] R2: 0.5582979754897279
--------------------------------------------

Column combination
['Frequency', 'Angle', 'Chord', 'Velocity', 'Displacement'] Adjusted R2: 0.5564513951197018
--------------------------------------------

Column combination
['Frequency', 'Angle', 'Chord', 'Velocity', 'Displacement'] CV R2: 0.48795379382121196
--------------------------------------------
Mean Squared Error
22.128643318247303 --------------------------------------------
Sum of Squared Error
6660.721638792438 --------------------------------------------
Root Mean Squared Error
4.704109194974889 --------------------------------------------
```

```
Mean Absolute Error
22.128643318247303 --------------------------------------------
```

Other summary statistics related to this model are a mean squared error of 22.13, sum of squared error of 6660.72, root mean squared error of 4.70, and mean absolute error of 22.12.

In scala the best performing model was symbolic regression. There were r squared, adjusted r squared, and cross validated r squared values of 0.651, 0.645, and 0.650 respectively. The root mean squared error of 4.07. This model performed much better than linear regression in python. Interestingly, scikit learn in Python performed linear regression better than in scala. Linear regression in scala had r squared, adjusted r squared, and cross validation r squared values of 0.516, 0.514, and 0.478 respectively. Root mean squared error of 4.80.

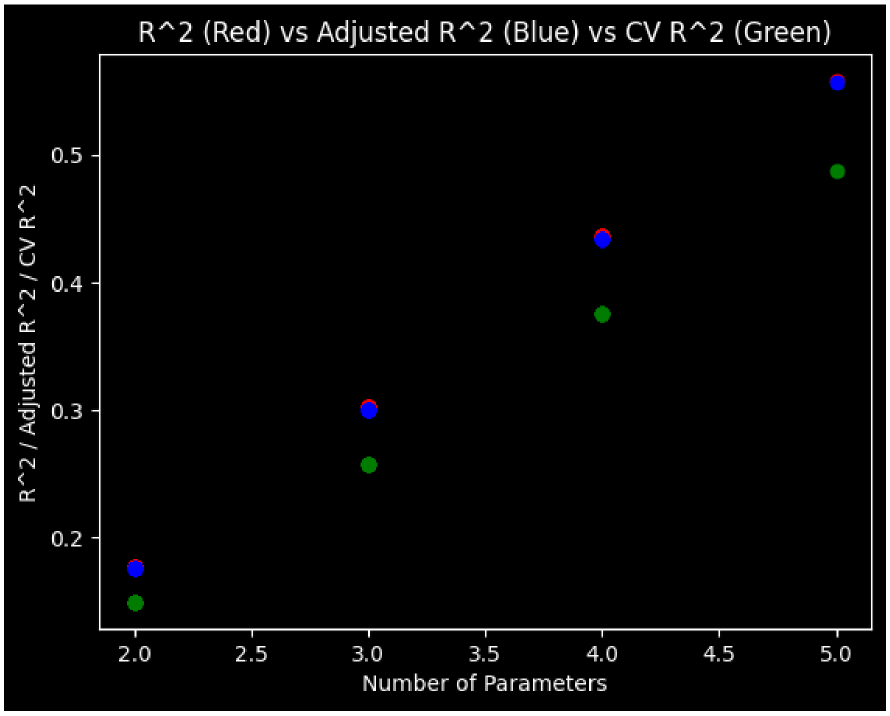
Scala summary statistics of symbolic regression:

```
fitMap qof = LinkedHashMap(rSq -> 0.651050, rSqBar -> 0.645144, sst -> 71482.377701, sse -> 24943.777222, mse0 -> 16.595993, rmse -> 4.073818, mae -> 3.168094, dfm -> 25.000000, df -> 1477.000000, fStat -> 110.227913, aic -> -4191.749310, bic -> -4053.553632, mape -> 2.543380, smape -> 2.539128)
```

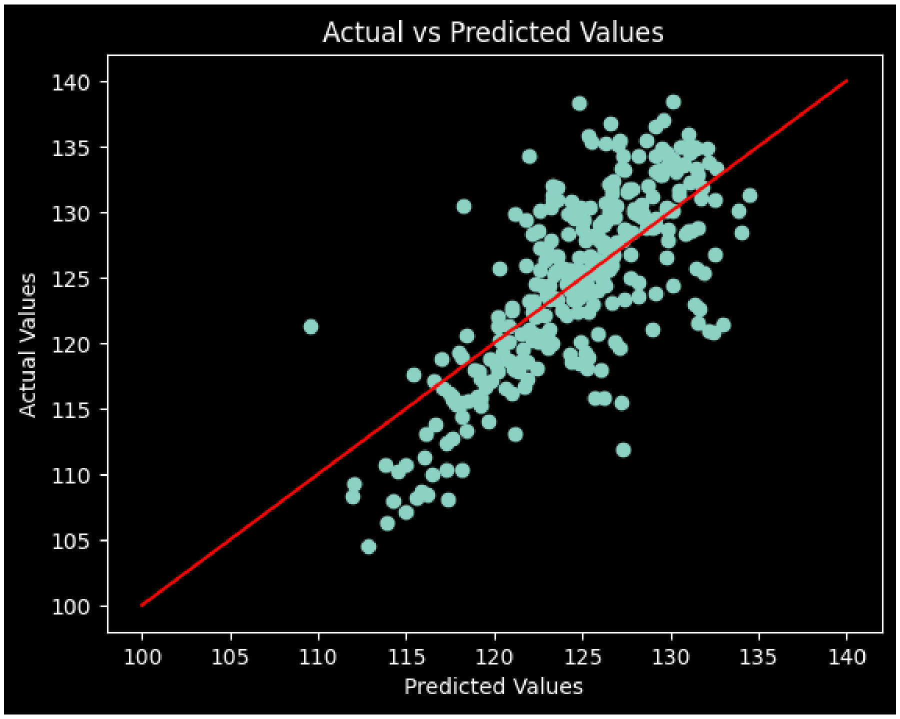
Scala summary statistics of linear regression:

```
fitMap qof = LinkedHashMap(rSq -> 0.515710, rSqBar -> 0.514092, sst -> 71482.377701, sse -> 34618.219133, mse0 -> 23.032747, rmse -> 4.799244, mae -> 3.728517, dfm -> 5.000000, df -> 1497.000000, fStat -> 318.824288, aic -> -4478.057739, bic -> -4446.166428, mape -> 2.994523, smape -> 2.988658)
```

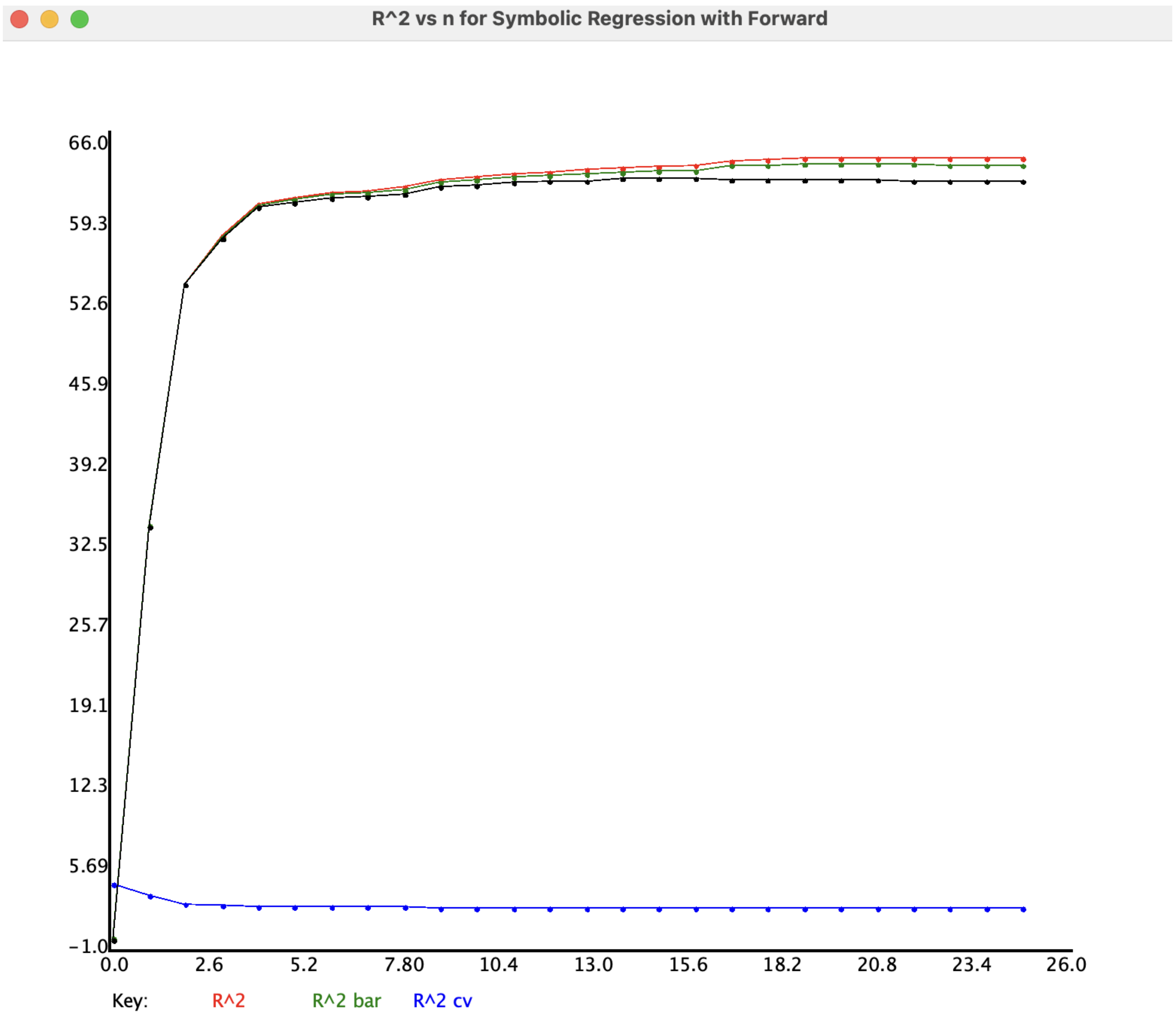
Plots:

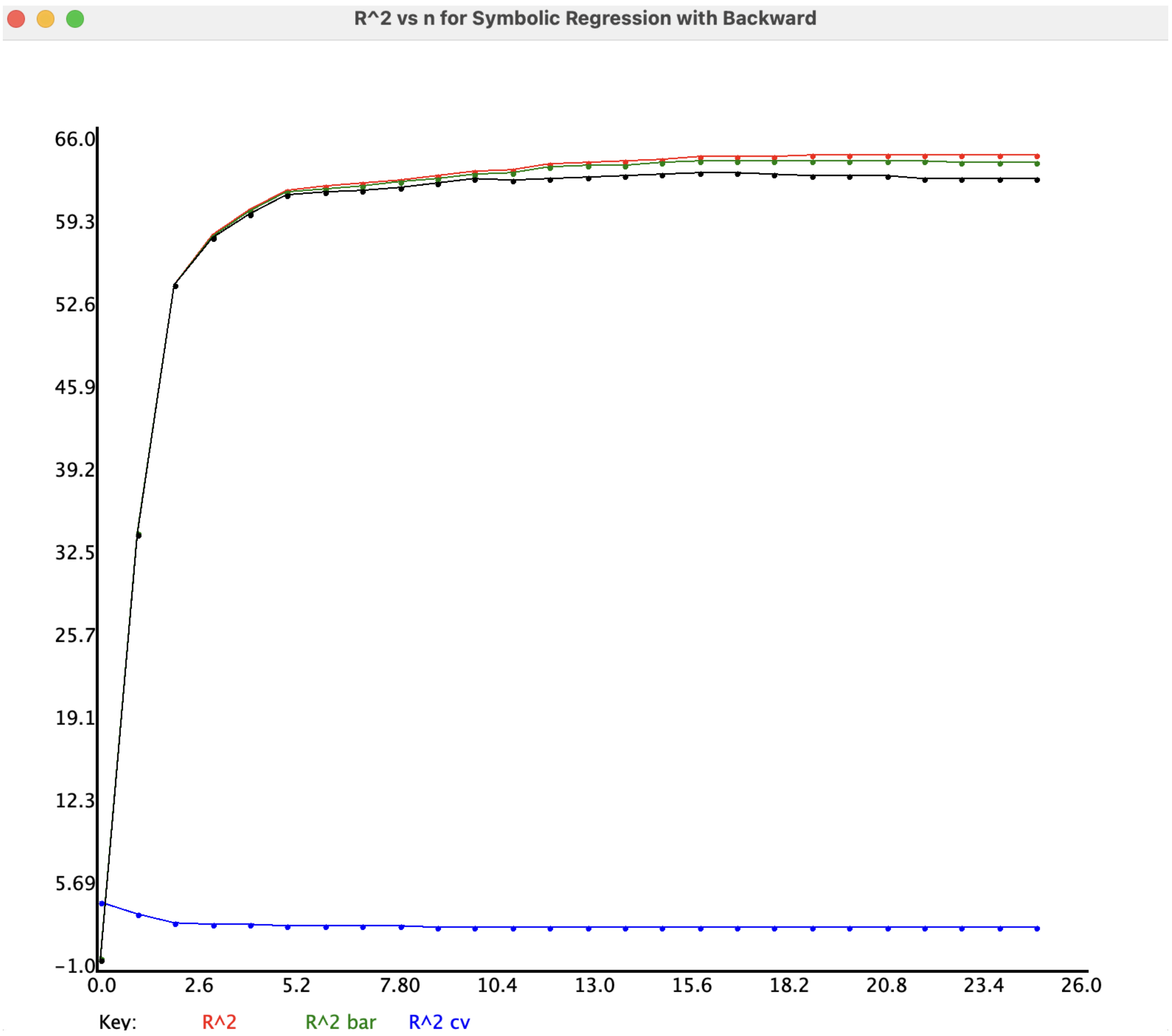


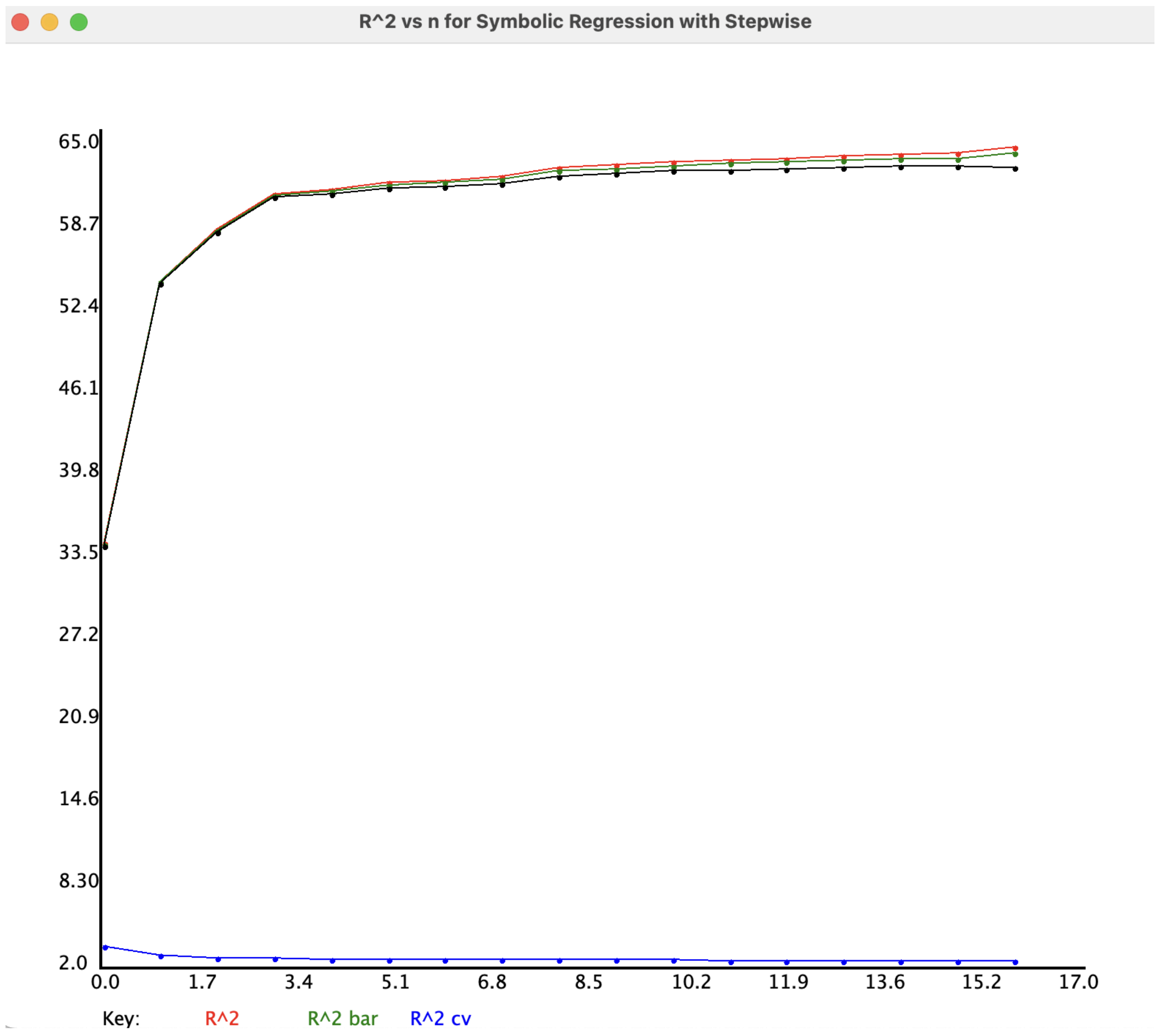
This figure shows the increase of r-squared values as the number of parameters increases. With few parameters in this data set the trend of increasing r-squared values appears consistent. I assume if the data set was larger and more parameters were added the increase in r-squared values would taper off and start to decrease eventually.



Here is a plot of our actual vs predicted values from our best performing model. Our linear regression with all features performed well with many of our predicted values not being far from actual values.


R^2 vs n for Symbolic Regression with Forward
66.0
59.3
52.6
45.9
39.2
32.5
25.7
19.1
12.3
5.69
-1.0
0.0
2.6
5.2
7.80
10.4
13.0
15.6
18.2
20.8
23.4
26.0
Key:
R^2
R^2 bar
R^2 cv


R^2 vs n for Symbolic Regression with Backward
66.0
59.3
52.6
45.9
39.2
32.5
25.7
19.1
12.3
5.69
−1.0
0.0
2.6
5.2
7.80
10.4
13.0
15.6
18.2
20.8
23.4
26.0
Key:
R^2
R^2 bar
R^2 cv



Discussion of Results

Through our analysis we came to the conclusion that using all five features in a linear regression model resulted in the best performing model. The resulting r squared, adjusted r-squared and cross validated r squared values of 0.558, 0.556, and 0.488 suggest this is a strong model. It is possible that by fine tuning the data set by the removal of outliers or conversion of some columns to categorical data, or adjusting the parameters of symbolic regression we can further model performance. Unfortunately, I am not experienced in data science or physics and am unsure how to go about doing so properly. For maintaining simplicity and accuracy the linear regression model that we have put forth should suffice. The flexibility of linear regression is desirable for many studies and the most common modeling that I have encountered in biological science.

Although I am unsure of how physics data prefers to utilize specific regression methods, I am confident that linear regression is a good choice to analyze this data set.

Looking back at our summary statistics we can remember we had values of mean squared error of 22.13, sum of squared error of 6660.72, root mean squared error of 4.70, and mean absolute error of 22.12. The standard deviation of the variable we are attempting to predict is 6.90. This means that most of our predicted values lie within a single standard deviation of the actual data, leading us to believe that these models are reliable.

The best results that we were able to achieve in scala used symbolic regression with displacement, frequency, angle, and chord. This led to high r squared values at ~0.65 for all three r squared measurements and a lower rmse at 4.07