A PROJECT REPORT ON

Optimize the time a Mercedes-Benz spends on the test bench based on anonymized features

A project report submitted in fulfillment for the
Diploma Degree in AI & ML Under

Applied Roots with University of Hyderabad

Project submitted by

Het Shah

Under the
Guidance of
Mentor: Harish
Ravi

Approved by: Mentor: Harish Ravi

University of Hyderabad

Declaration of Authorship

We hereby declare that this thesis titled “Optimize the time a Mercedes-Benz spends on the test bench based on anonymized features” and the work presented by the undersigned candidate, as part of Diploma Degree in AI & ML.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name: Het Shah

Thesis Title: Optimize the time a Mercedes-Benz spends on the test bench based on anonymized features

CERTIFICATE OF RECOMMENDATION

We hereby recommend that the thesis entitled “Optimize the time a Mercedes-Benz spends on the test bench based on anonymized features” prepared under my supervision and guidance by Het Shah be accepted in fulfilment of the requirement for awarding the degree of Diploma in AI & ML Under applied roots with University of Hyderabad. The project, in our opinion, is worthy for its acceptance.

Mentor: Harish Ravi

Under Applied roots with

University of Hyderabad

ACKNOWLEDGEMENT

Every project big or small is successful largely due to the effort of a number of wonderful people who have always given their valuable advice or lent a helping hand. I sincerely appreciate the inspiration; support and guidance of all those people who have been instrumental in making this project a success. I, Het Shah student of applied roots, is extremely grateful to mentors for the confidence bestowed in me and entrusting my project entitled “Optimize the time a Mercedes-Benz spends on the test bench based on anonymized features” with special reference.

At this juncture, I express my sincere thanks to Mentor: Het Shah of applied roots for making the resources available at right time and providing valuable insights leading to the successful completion of our project who even assisted me in completing the project.

Name: Het Shah

Contents

Abstract:

Mercedes-Benz is one of the world's leading premium car manufacturers. They give paramount importance to safety and reliability of each and every car produced by them. For this they have developed a robust testing system which considers a plethora of independent features into consideration. Therefore, conducting these tests manually is time consuming, tedious and at some degree hazardous to the environment due to excessive carbon emission. In this project we will use the testing data provided by Mercedes-Benz and try to tackle the curse of dimensionality and reduce the time that cars spend on the test bench. Use various AI/ML methods to try and help company to reduce the time the car spends on the test floor to give them a dual benefit of time and hence cost saving along with saving the environment.

Introduction:

Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include, for example, the passenger safety cell with crumple zone, the airbag and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premium car makers. Daimler's Mercedes-Benz cars are leaders in the premium car industry. With a huge selection of features and options, customers can choose the customized Mercedes-Benz of their dreams. .

To ensure the safety and reliability of each and every unique car configuration before they hit the road, Daimler's engineers have developed a robust testing system. But, optimizing the speed of their testing system for so many possible feature combinations is complex and time-consuming without a powerful algorithmic approach. As one of the world's biggest manufacturers of premium cars, safety and efficiency are paramount on Daimler's production lines

Problem Statement:

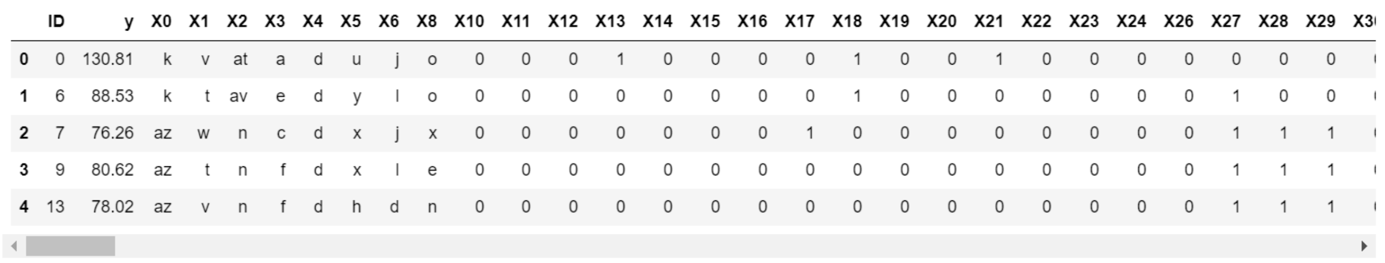
The Mercedes-Benz Greener Manufacturing project is undertaken with and objective of developing a machine learning model which can accurately predict the time a car will spend on the test bench based on the vehicle configuration. Based on the set of customization options and the features a users needs to be added on the vehicle the vehicle configuration is determined. The objective behind the problem is that an accurate model will be able to reduce the total time spent testing vehicles by allowing cars with similar testing configurations to be run successively.

Business/Real-world impact of solving this problem:

The stated goal of the competition is to reduce the time vehicles spend on the test stand which consequently will decrease carbon dioxide emissions associated with the testing procedure. Although the reduction in carbon dioxide may not be noteworthy on a global scale, improvements to Mercedes's process can be passed along to other automakers or even to other industries which could result in a significant decrease in carbon dioxide emissions. Moreover, one of the fundamental tenets of machine learning is that the efficiency of current systems can be improved through the use of the massive quantities of data now routinely collected by companies.

Data Description:

Two data files are provided by Mercedes-Benz for use in the competition: a training dataset, and a testing dataset in csv format. The training and testing data both contain 4209 vehicle tests obtained by Mercedes for a range of vehicle configurations. The training data also contains the target variable, or testing duration in seconds, for each vehicle test.Each vehicle test is defined by the vehicle configuration, which is encoded in a set of features. Both the training and the testing dataset contain 376 different vehicle features with names such as 'X0', 'X1', 'X2' and so on.All of the features have been anonymized meaning that they do not come with a physical representation. The description of the data does indicate that the vehicle features are configuration options. There are 8 categorical features, with values encoded as strings such as 'a', 'b', 'c', etc. The other 368 features are binary, meaning they either have a value of 1 or 0. Each vehicle test has also been assigned an ID which was not treated as a feature for this analysis.An image of a representative subset of the training data obtained by the head() function is shown below:



	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X26	X27	X28	X29
0	0	130.81	k	v	at	a	d	u	j	o	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
2	7	76.26	az	w	n	c	d	x	j	x	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1
3	9	80.62	az	t	n	f	d	x	l	e	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
4	13	78.02	az	v	n	f	d	h	d	n	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Data Acquisition:

Open-Source data - The dataset for this project is retrieved from Kaggle, the home of Data Science.

Source: Kaggle

Link: https://www.kaggle.com/competitions/mercedes-benz-greener-manufacturing/data

☐ Data Size: 166.13 MB

☐ Data Shape: 4209 rows and 376 columns

We can see there are three formats of data types in given features.

• float64

• int64

• object

Tools (Pandas, SQL, numpy etc) that you will use to analyze and process this data Here, we will be using Python along with the below-listed libraries

- ☐ Pandas,
- ☐ NumPy,
- ☐ matplotlib,
- ☐ seaborn
- ☐ Python 3.8, Jupyter Notebook,
- ☐ Scikit-learn, Scipy

Approach, Methodology and Results

The workflow of the project is as follows:

1. Requirement Analysis
2. Data Exploration and Preparation
3. Data Pre-processing
4. Model Selection
5. Model Optimization
6. Model Testing and Evaluation
7. Reporting of Results

Requirement Analysis

Functional Requirements

Purpose

Predict time in seconds the car spends on the test floor based on the features and configurations opted by the customer in the Mercedes Benz car.

Inputs

A dataset is provided that contains anonymous features with the corresponding mapping values.

Output

The model predicts the time that the car spends on test floor based on the features and different configuration present on the car.

Usability

Particularly useful to the car vendor to predict the time spent on test bench

Hardware Requirements

- Computer system
- 16, 32 GB Ram
- Intel i3 and above processor
- Server Overhead
- Port 8000
- Internet connectivity

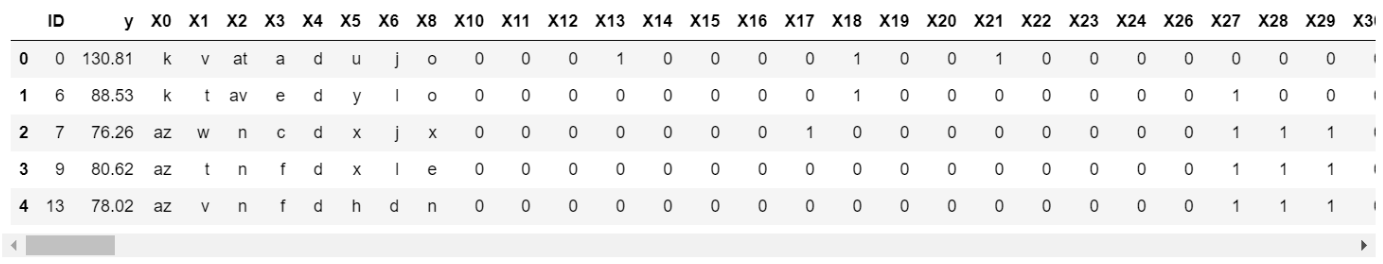
Software Requirements

- Notepad++ or any Code Editor
- Anacoda Navigator , Jupyter Notebook

- MS-Excel
- Python 3
- Python libraries like pandas, NumPy etc.
- scikit-learn
- Client environment may be Windows or Linux
- Web Browser

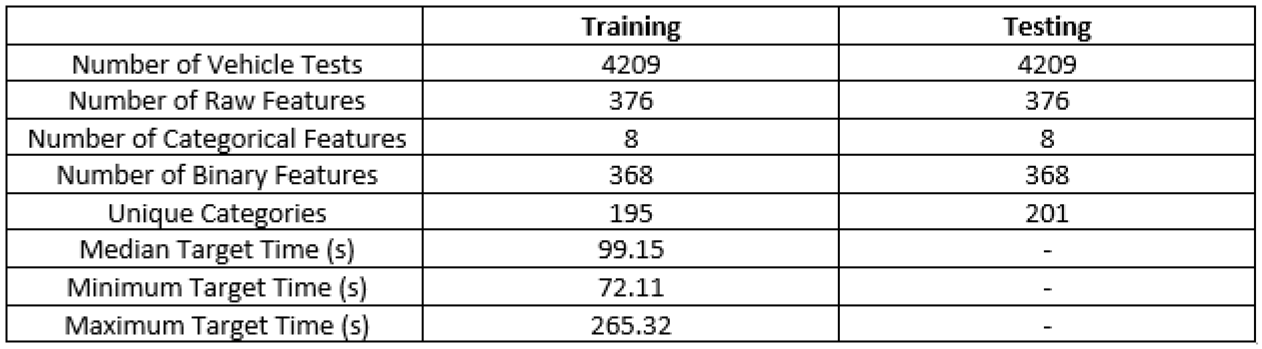
Data Exploration and Preparation

The training and testing data both contain 4209 vehicle tests obtained by Mercedes for a range of vehicle configurations. The training data also contains the target variable, or testing duration in seconds, for each vehicle test.Each vehicle test is defined by the vehicle configuration, which is encoded in a set of features. Both the training and the testing dataset contain 376 different vehicle features with names such as 'X0', 'X1', 'X2' and so on. All of the features have been anonymized meaning that they do not come with a physical representation. The description of the data does indicate that the vehicle features are configuration options such as suspension setting, adaptive cruise control, all-wheel drive, and a number of different options that together define a car model. There are 8 categorical features, with values encoded as strings such as 'a', 'b', 'c', etc. The other 368 features are binary, meaning they either have a value of 1 or 0. Each vehicle test has also been assigned an ID which was not treated as a feature for this analysis.An image of a representative subset of the training data is shown below



	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X26	X27	X28	X29	X3
0	0	130.81	k	v	at	a	d	u	j	o	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	
1	6	88.53	k	t	av	e	d	y	l	o	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	
2	7	76.26	az	w	n	c	d	x	j	x	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	
3	9	80.62	az	t	n	f	d	x	l	e	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
4	13	78.02	az	v	n	f	d	h	d	n	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	

A brief summary table of the testing and training dataset follows:



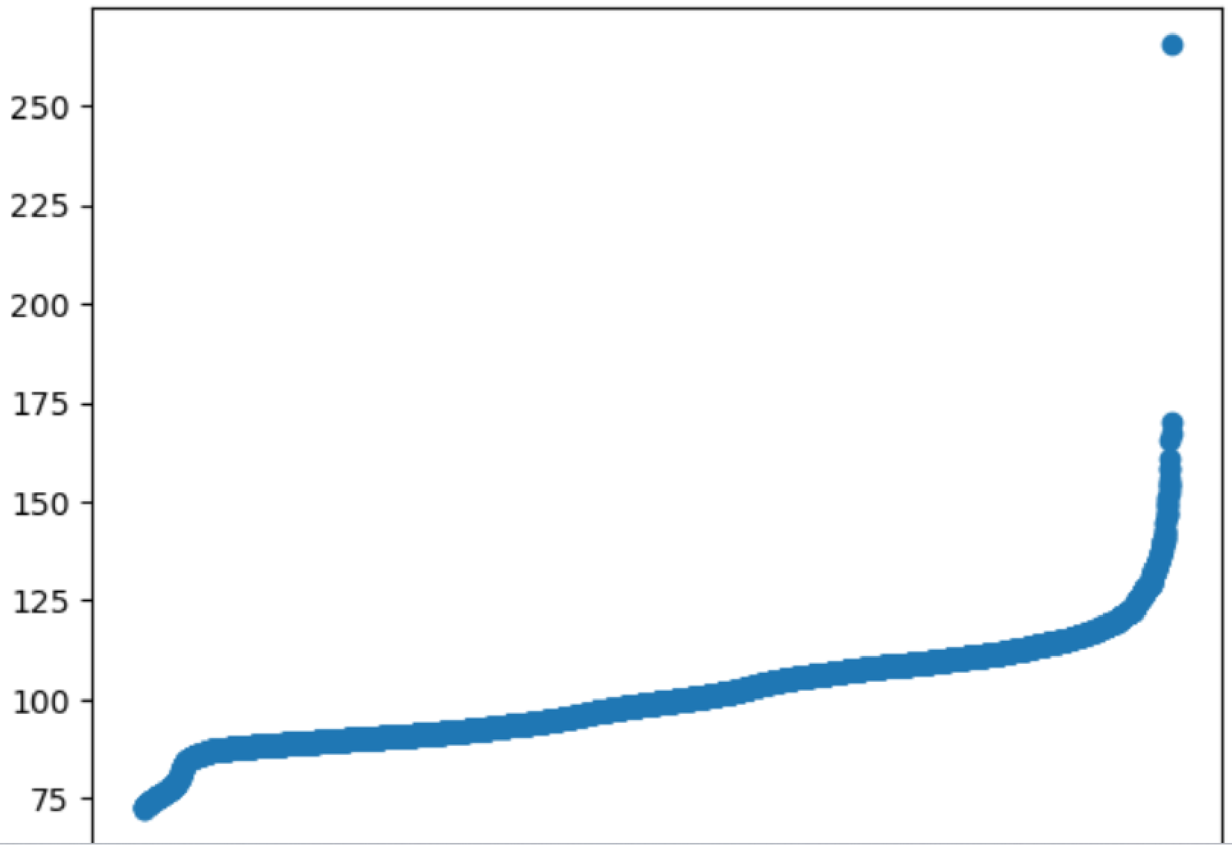
	Training	Testing
Number of Vehicle Tests	4209	4209
Number of Raw Features	376	376
Number of Categorical Features	8	8
Number of Binary Features	368	368
Unique Categories	195	201
Median Target Time (s)	99.15	-
Minimum Target Time (s)	72.11	-
Maximum Target Time (s)	265.32	-

Although the data has been cleaned by Mercedes prior to being made available for the competition and therefore there are no errors or missing entries, the data may still contain outliers with respect to vehicle testing time. These outliers could be valid data but are extreme enough to affect the performance of the model.

The first aspect of the data to investigate was the target variable, the vehicle testing duration. The plot below shows all of the testing times arranged from shortest to longest from left to right.One outliers can be seen at right with the highest value. Plotting the data demonstrates the extreme nature of this data point more effectively than examining the numbers.

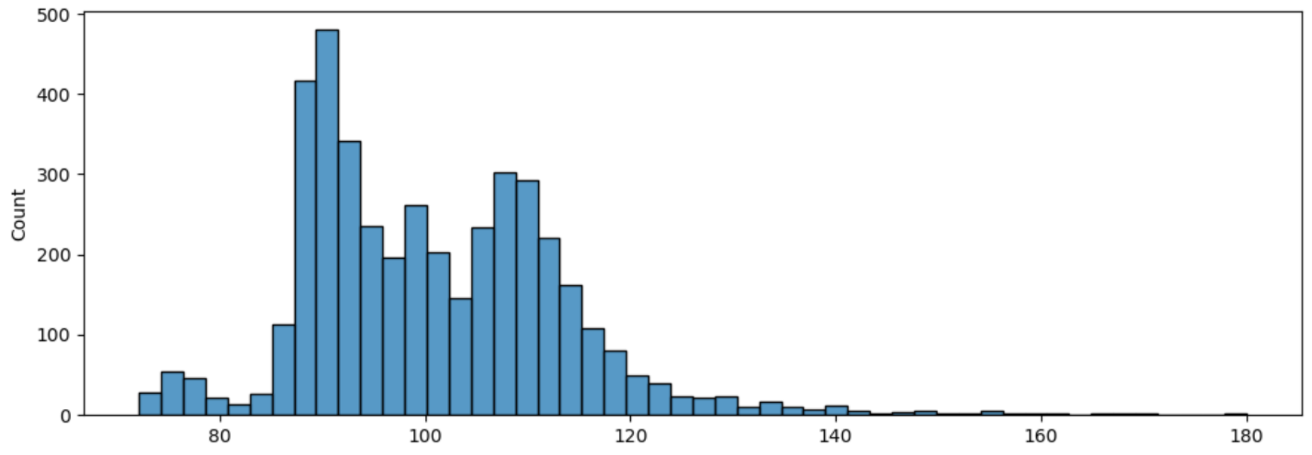
```
plt.scatter(range(train_df.shape[0]), np.sort(train_df.y.values))
```

```
]:  <matplotlib.collections.PathCollection at 0x287c31af0a0>
```



The testing times can also be visualized as a histogram. For this visualization, the outlying data point has been removed in order to better represent the majority of the data.

```
plt.figure(figsize=(12,4))
sns.histplot(train_df.y.values, bins=50, kde=False)
plt.xlabel('y value', fontsize=12)
plt.show()
```



There are several conclusions to be drawn from this histogram:

· The majority of test durations are between 90 and 100 seconds

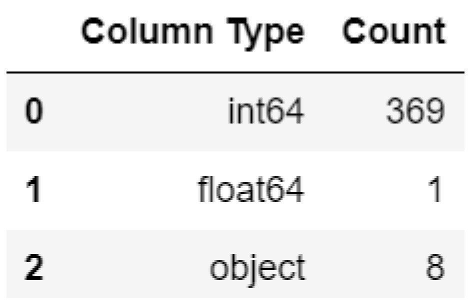
· There are peaks in testing times around 97–98 seconds and near 108 seconds.

· The testing times are bi-modal, with two distinct peaks.

· This data is positively skewed, with a long tail stretching into the upper values.

Based on the target variable visualization, I concluded that I was justified in removing one outlier, the training data point with the greatest testing time. Although it is a valid data point, it is extreme enough that it will adversely affect the performance of the algorithm.

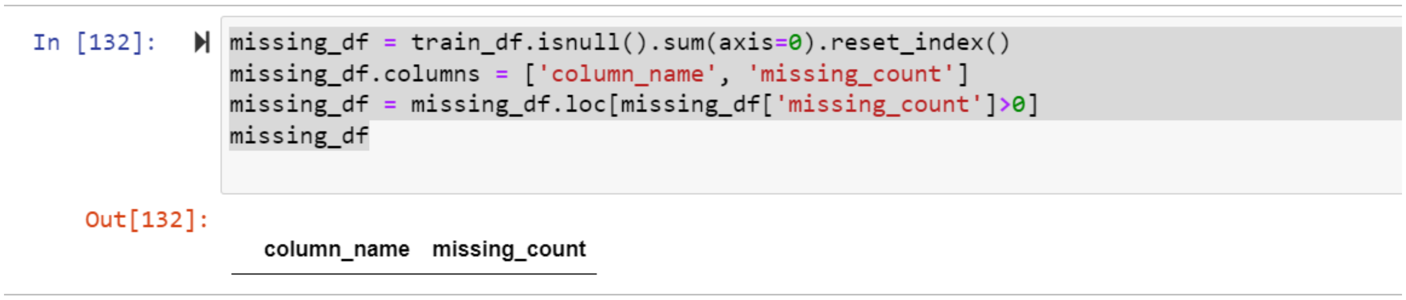
Then i did have a look at the data type of all variables present in the dataset and the result is as follows:



	Column Type	Count
0	int64	369
1	float64	1
2	object	8

So, majority of the columns are integers with 8 categorical columns and 1 float column which is the target variable. X0 to X8 are categorical columns.

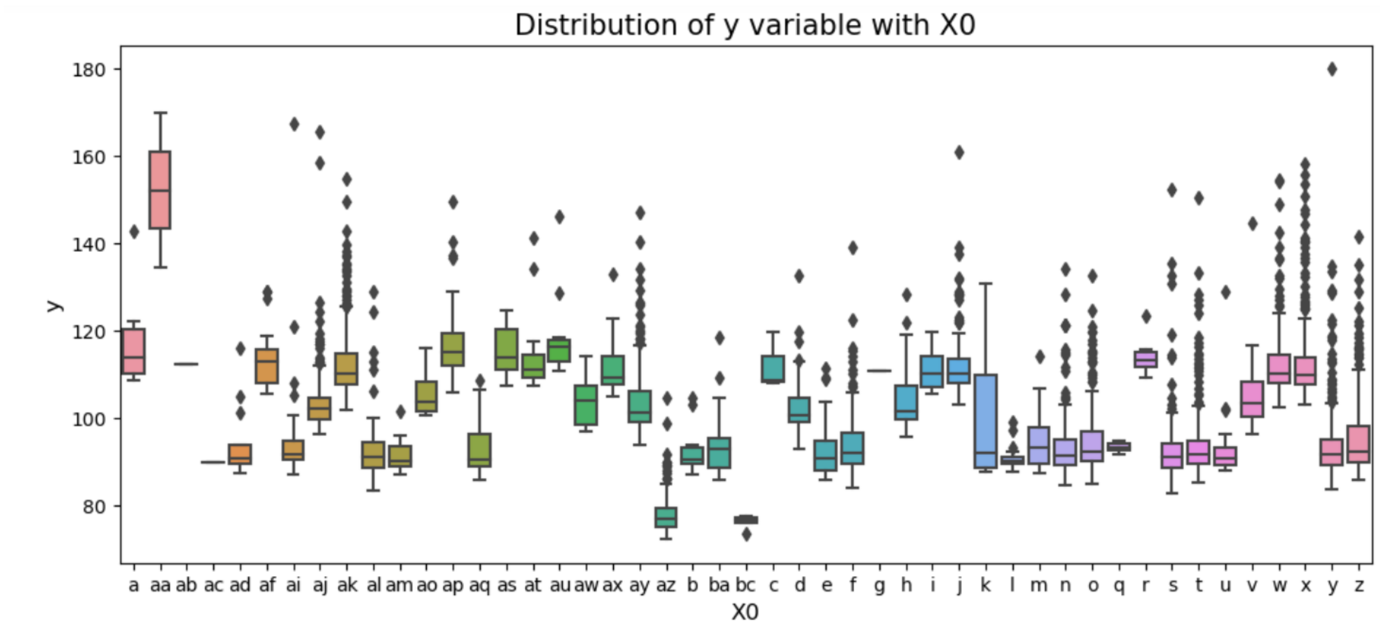
Missing Values:

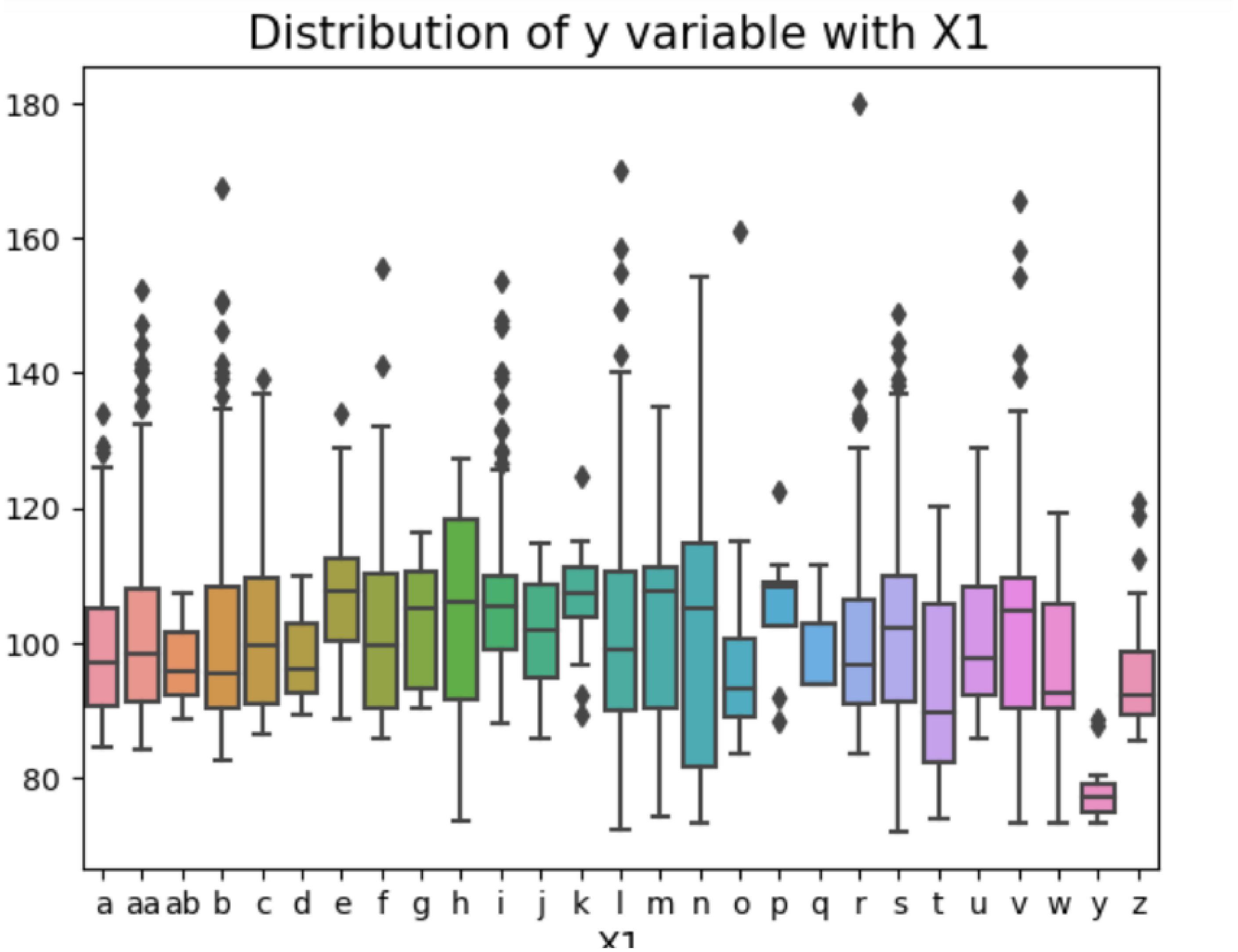


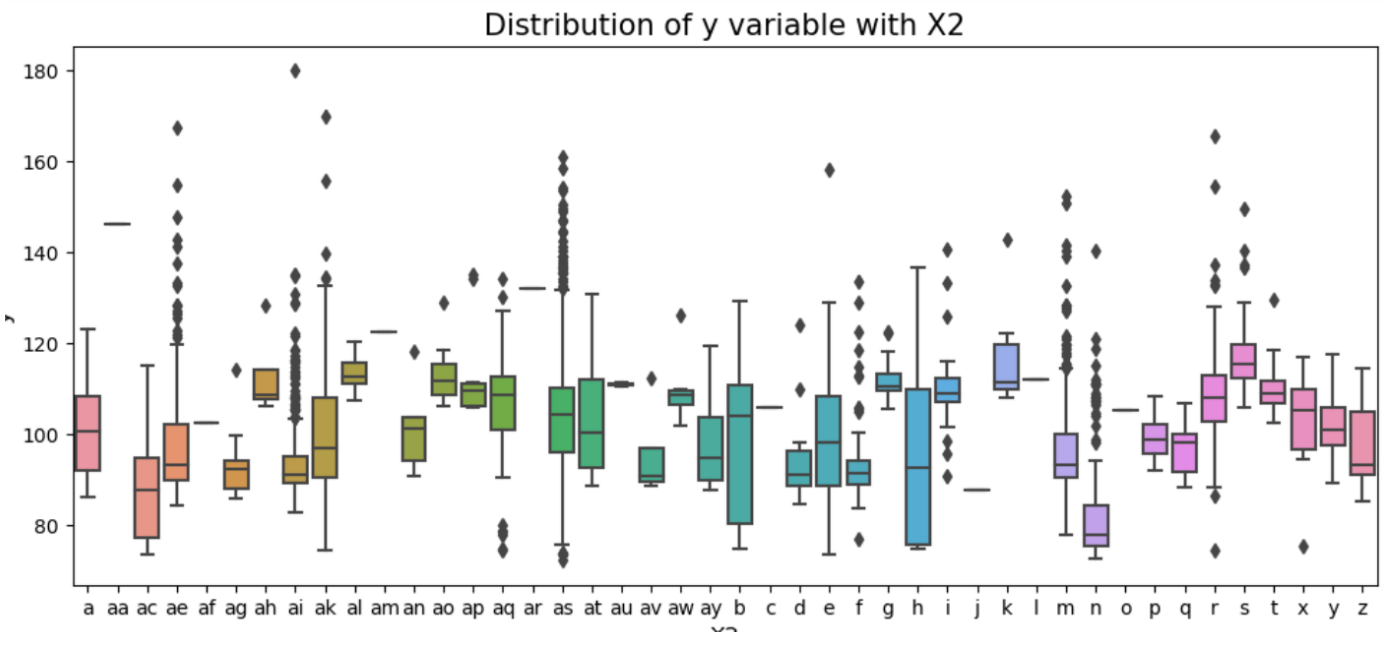
```
In [132]:  missing_df = train_df.isnull().sum(axis=0).reset_index()
            missing_df.columns = ['column_name', 'missing_count']
            missing_df = missing_df.loc[missing_df['missing_count']>0]
            missing_df

Out[132]:
              column_name   missing_count
```

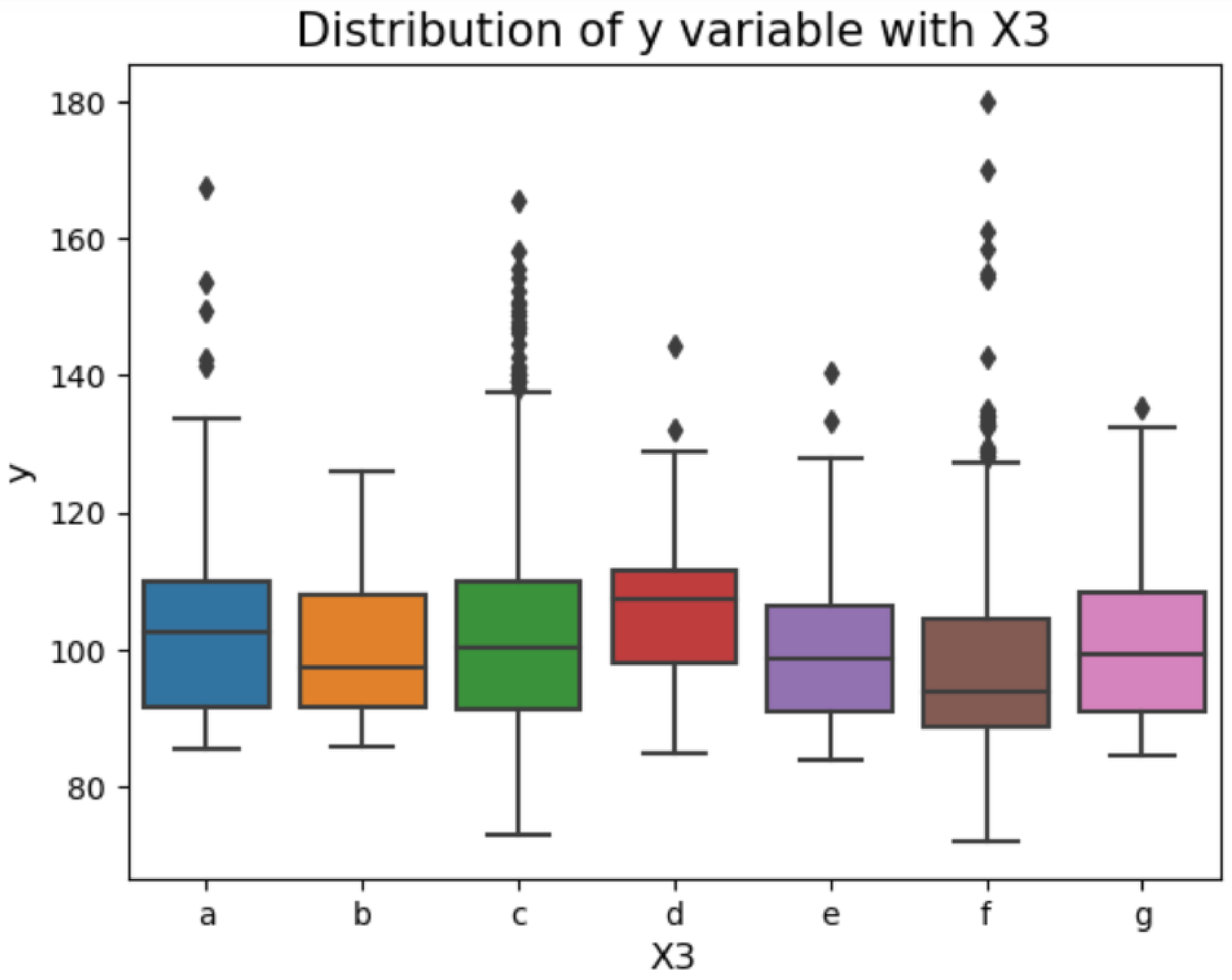
As we can see there is no missing value in the dataset.

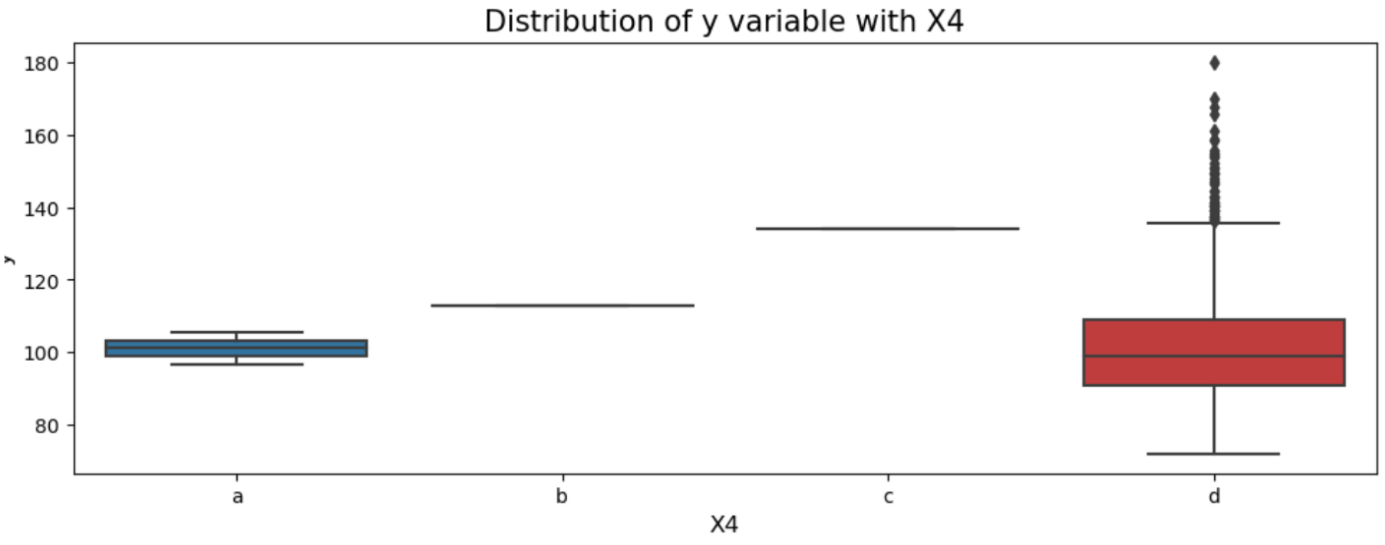
The **second** half of the data exploration phase focused on the vehicle features. I began with the categorical variables by plotting the testing durations versus the unique category in boxplots. This was done for each of the categorical variables on the training data.

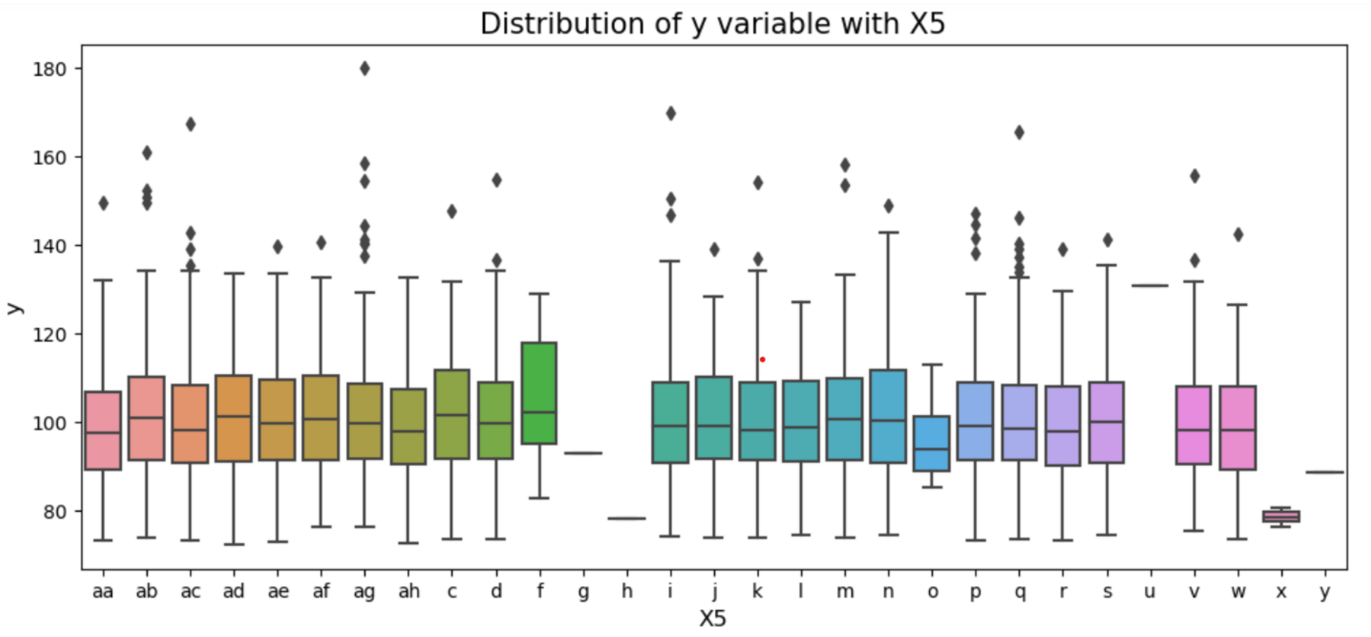


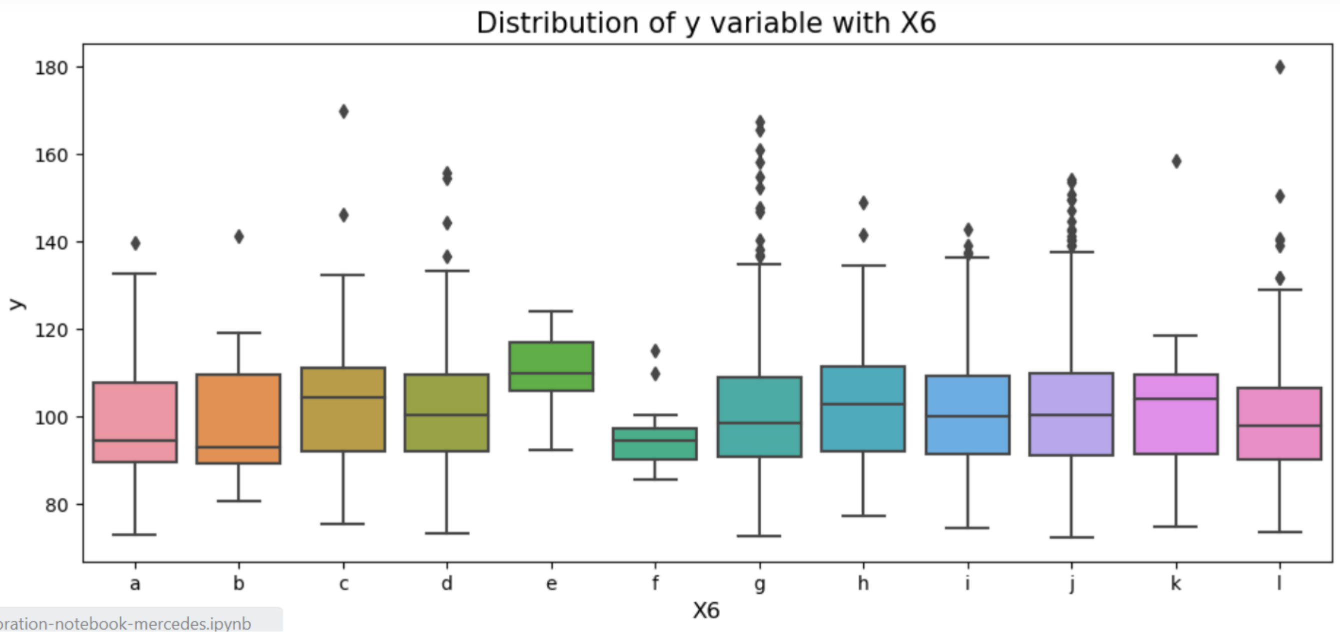

Distribution of y variable with X1
180
160
140
120
100
80
a aa ab b c d e f g h i j k l m n o p q r s t u v w y z
X1


Distribution of y variable with X2
180
160
140
120
100
80
a aa ac ae af ag ah ai ak al am an ao ap aq ar as at au av aw ay b c d e f g h i j k l m n o p q r s t x y z

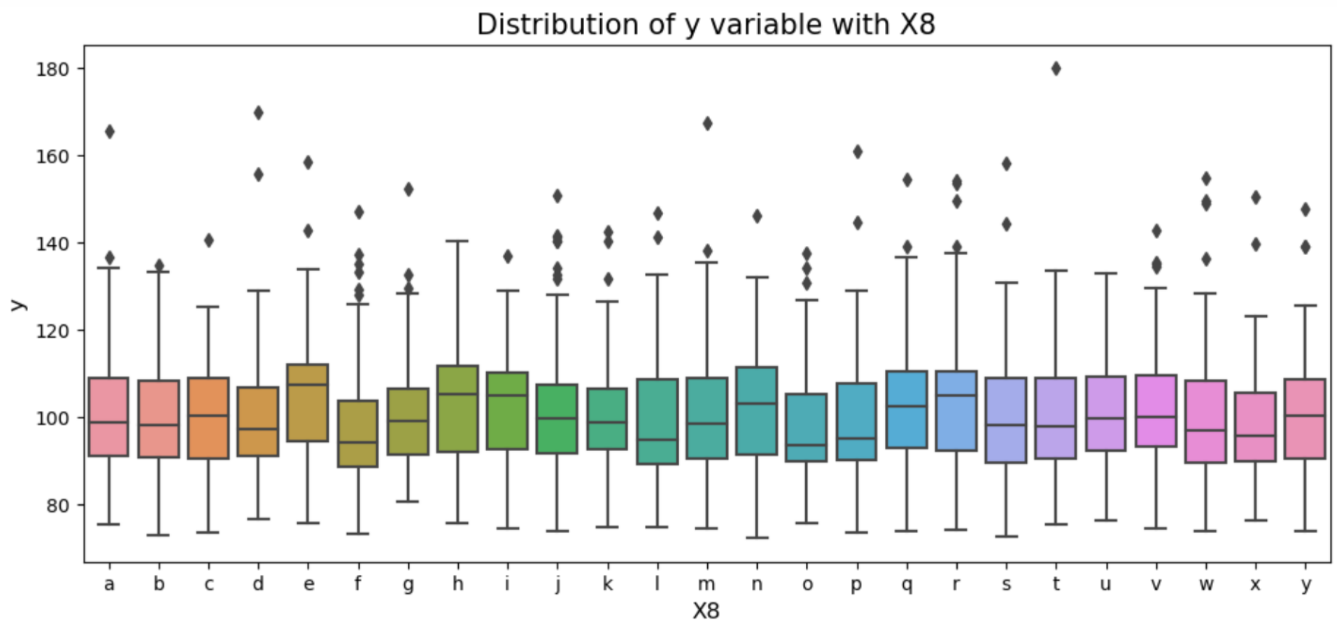

Distribution of y variable with X3
y
180
160
140
120
100
80
a
b
c
d
e
f
g
X3


Distribution of y variable with X4
y
180
160
140
120
100
80
a
b
c
d
X4

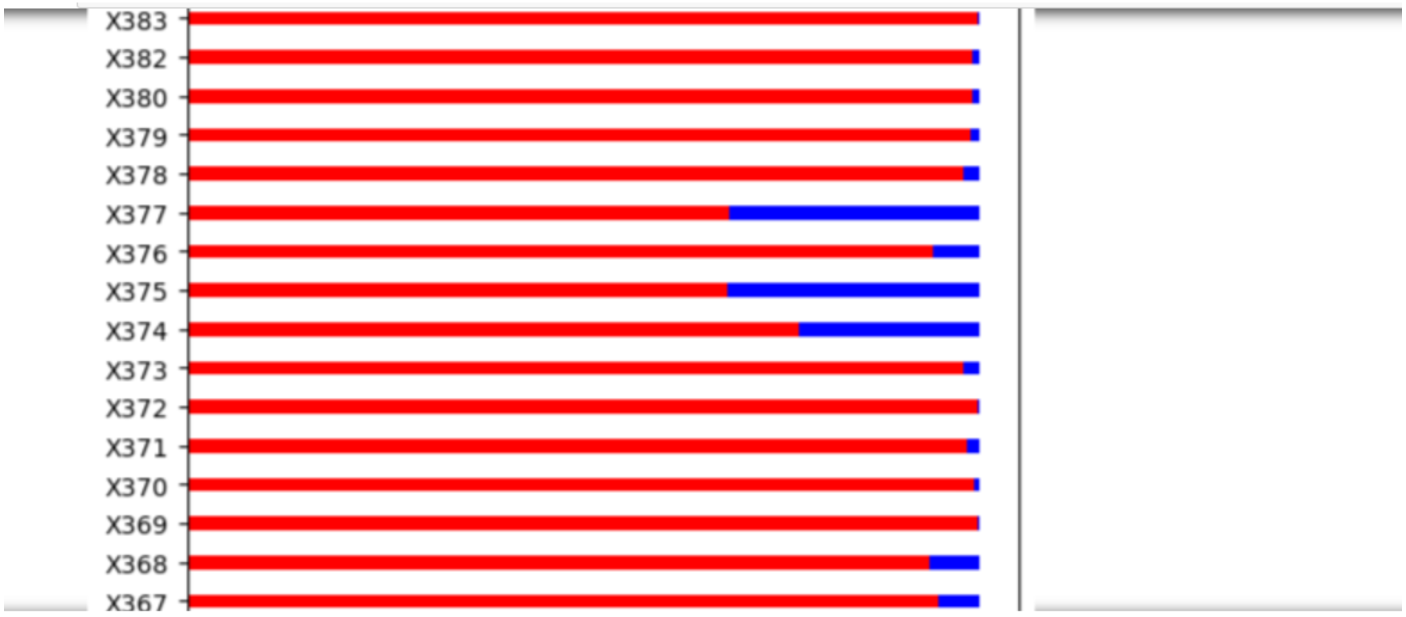




xploration-notebook-mercedes.ipynb



After focusing on vehicle features in the second half, now we moved out attention to the binary variables present in the dataset and start with getting the number of 0's and 1's in each variable. Lets plot the number of 0's and 1's present in each column.



With this particular problem, it was difficult to draw any conclusions from the visualizations of the explanatory variables. The above figure demonstrates that some of the binary features are shared by all of the vehicles and some are shared by none. Looking further into the numerical data for the testing set, I identified 12 binary variables where the values were either all 1 or 0. As there is no variation in these features, they have no predictive power and therefore, these 12 features may be removed. The data exploration has underscored the need for dimensionality reduction. Fortunately, one common technique for reducing the number of features, Principal Components Analysis (PCA), is an unsupervised technique that does not require an understanding of the physical representation of the features. The conclusions from data exploration stage is as follows:

· One outlier, as determined by vehicle testing duration, needs to be removed

· 12 binary variables encode no information and should be removed

· There are no noticeable trends within the categorical or binary variables

· Unsupervised dimensionality reduction (PCA) will need to be performed on the data

Data Preprocessing:

There are several interrelated steps in the data preprocessing workflow.

· Obtaining and cleaning data

· Data preparation

· Feature engineering/ Dimensionality reduction

The first step requires obtaining the data in a usable format. For this project, no “data wrangling” was required because the data was already provided in an organized format. Data preparation mainly consisted of one-hot encoding the categorical variables, removing the outliers, and removing features that do not encode any information. In the training data, there was a single outlier with a testing time far greater than that of the next highest value. This point was removed so that it would not skew the predictions of the regression. There were also 12 binary features with values that were either all 1’s or 0’s. These variables (they cannot even be called variables because they do not change, but should be referred to as constants) do not contain any information and were removed. After aligning the training and testing datasets to ensure they contained the same number of features, I was left with a training set of 4208 data points with labels and 531 features. The testing set has 4209 data points and 531 features without labels.

The final step in the data preprocessing pipeline is feature engineering. This encompasses a range of operations including adding new features or removing features to reduce the dimensionality of the data. Based on the data exploration, I decided that I would not be able to create brand new features by combining existing features because I did not observe any trends within the variables. Therefore, I concentrated on reducing the number of features.

Dimensionality Reduction/ Feature Engineering

One implementation is using a subset of features selected by ranking the variables in terms of correlation coefficient with respect to the target variable.

Another method involves selecting the most relevant features for a given algorithm. Within Scikit-learn, there are a number of algorithms that will display the feature importances after training. Both of the ensemble methods explored in this report have a feature importances attribute. However, I received substantially different results when comparing the weights given to each feature for both models. Consequently, I decided that selecting a smaller number of features based on feature importances was not the correct decision.

After some experimentation, the dimensionality reduction method that I did implement is an unsupervised technique known as Principal Components Analysis (PCA). PCA creates a set of basis vectors that represent the dimensions along which the original data has the greatest variation. In other words, the original data is projected onto a new axis with the greatest variance in order to reduce the dimensions while preserving the maximum amount of information. The first principal component explains the greatest variance in the data, while the second explains the second most and is orthogonal to the first and so on. A common technique is to keep the number of principal components that explain a given percentage of the variance within a dataset. After some trial and error, I decided to retain the Principal Components that explained 98% of the variation in the data.The meant keeping the first 180 principal components.The 140 principal components selected explain 95.31% of the variance in the data. The PCA algorithm was trained on the training features and then the training features are transformed by projecting the original features along the first 140 principal components. The testing data features are transformed by the already-trained PCA model. Applying PCA considerably reduces the number of dimensions within the data and should lead to improved performance by reducing the noise in the data.

Model Selection:

A benchmark model for this regression task is a linear regression. Linear regression is a method for modeling the relationship between a target variable (also called the response variable) and one or more independent, explanatory variables.When there are more than one explanatory variables, this process is called multiple linear regression. In this situation, the target variable is the testing time of the car and the explanatory variables are the customization options for the car. A linear regression model assumes that the response variable can be represented as a linear combination of the model parameters (coefficients) and the explanatory variables. The benchmark model in this situation is a basic linear regression that predicts the target for an instance based on the instance's features multiplied by the model's parameters (coefficients) and added to a bias term. This is shown as an equation below:

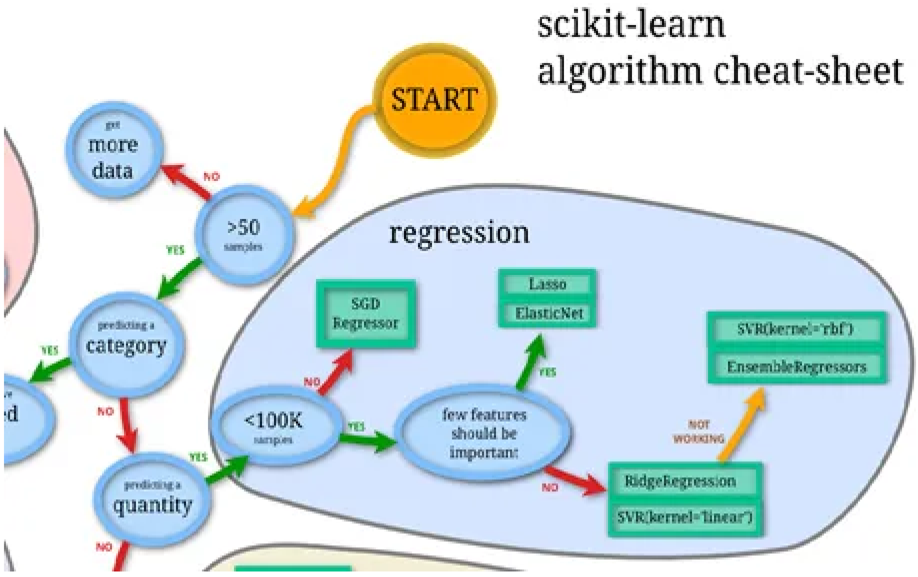
$$y = c_0 + c_1 * x_1 + c_2 * x_2 + \ldots + c_m * x_m$$

where y is the target (predicted) variable, the c terms are the model parameters (coefficients) and the x terms are the feature values. When the model trains, it learns the set of parameters, and when it is asked to make a prediction, it takes the instance's feature values and multiplies by the parameters to generate a prediction. The default method for Linear Regression in Scikit-learn is ordinary least squares, where the model calculates parameters that minimize the sum of the squared difference between the prediction and the known targets.

Initially, I used the entire set of raw features for creating a benchmark model. However, this resulted in a negative coefficient of determination, indicating that the model was performing worse than if it had drawn a constant line at the mean of the data. To develop a more reasonable benchmark, I decided to reduce the dimensionality of the data. I found the Pearson's Correlation Coefficient, or R, between all of the individual features and the target variable. The correlation coefficient describes the strength and direction of the linear relationship between an independent and dependent variable. A value of 1.0 would indicate a perfectly linear positive relationship and thus the Pearson coefficient can be used as one method to determine which features will be useful when predicting a target variable. Once I had identified the correlation coefficients, I trained a number of linear regressors using a range of the most positively correlated features. The top-scoring model used the 47 top correlated features. I selected this as the benchmark model because of the reasonable R^2 value which would take some effort to better. The final benchmark model scored a 0.56232 coefficient of determination on the testing set using five-fold cross validation.The predictions made on the testing set scored a 0.54549 coefficient of determination.The benchmark serves as a comparison for the final model. The final model will need to significantly outperform the benchmark model on the R^2 measure to make it a success. The final model should score higher both on the training dataset, using five-fold cross validation, and on the testing set.

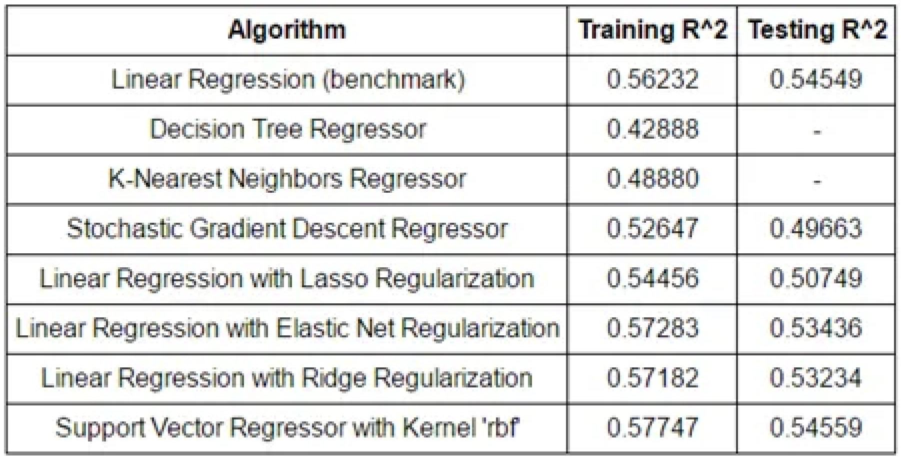
Model Optimization:

The first stage involved testing and evaluating a selection of algorithms. Different algorithms are suited for different tasks, and the best way to determine the correct algorithm is to try them out on the dataset. I began this process as usual by referring to the Scikit-learn algorithm selection flowchart. The relevant section of the chart for choosing a regression algorithm is shown below.



As this was a regression task with fewer than 100,000 samples (training data points), there were several starting options. My approach was to evaluate a number of the simple algorithms recommended, and if none were sufficient, I would proceed to ensemble methods. The results of evaluating a number of simple classifiers are shown below. As is demonstrated in the table, the only simple model that outperformed the benchmark was the Support Vector regressor with an rbf kernel. All of the simple models were evaluated with the

140 principal components from the data preprocessing and were created with minimal hyperparameter tuning. The objective at this stage was simply to evaluate the models to determine if any were acceptable for the problem. I expected that I would have to move on to ensemble methods, but in some machine learning tasks, a simple model is able to capture all of the relationships within the data and can perform very well.



Algorithm	Training R^2	Testing R^2
Linear Regression (benchmark)	0.56232	0.54549
Decision Tree Regressor	0.42888	-
K-Nearest Neighbors Regressor	0.48880	-
Stochastic Gradient Descent Regressor	0.52647	0.49663
Linear Regression with Lasso Regularization	0.54456	0.50749
Linear Regression with Elastic Net Regularization	0.57283	0.53436
Linear Regression with Ridge Regularization	0.57182	0.53234
Support Vector Regressor with Kernel 'rbf'	0.57747	0.54559

Examining the performance of the simple regressors, it was clear I would have to employ a more sophisticated model to significantly improve on the benchmark. The next logical place to turn, as evidenced by the flowchart and on the discussion boards for the Kaggle Competition, [25] was ensemble methods. Ensemble methods work to create a model by combining multiple or building upon many simple models. [26] There are two commonly identified classes of ensemble methods:

· Bagging

· Boosting

Bagging [27] (short for bootstrap aggregating) uses the "wisdom" of the crowd to create a better model than an individual regressor. If training a single model is like asking one business analyst to make a prediction given an entire dataset, bagging is like taking the average forecast of an entire roomful of analysts, each of whom have seen a small part of the data. For example, an Extra Trees regressor trains numerous Decision Trees, with each tree fit on a different subset of the data. In the bagging technique, these subsets are randomly chosen, and the final prediction is attained by averaging all of the predictions. The implementation used in this project set the Bootstrap parameter to false meaning that the subsets are not replaced after being used for training, a technique known as pasting [28]). Boosting is based on a similar idea, except each successive simple model is trained on the most "difficult" subset of the data. The first model is trained on the data, and then the data points with predictions that are the furthest away from the true values (as determined by the residuals, or the difference between the prediction and the known target value) are used to trained the second model and so on. Over time, the entire ensemble will become stronger by concentrating on the hardest to learn data. The final prediction is a weighted average of all the individual regressors with the most "confident" trees given the highest weighting. One boosting technique that is very popular for machine learning competitions is known as XGBoost, [29] which stands for extreme gradient boosting. Both of the ensemble methods used in this project are based on many individual Decision Trees. Decision Trees are a popular choice to use in an ensemble because of their well-established default hyperparameters and relatively rapid training time. Studies [26][30] have shown ensemble methods are more accurate than a single classifier because they reduce the variance that can arise in a single classifier. Bagging was observed to almost always have better performance than a single model, and boosting generally performed better depending on the dataset and the model hyperparameters.

The final model used both an Extra Trees and an Extreme Gradient Boosting ensemble method in Scikit-learn. [31] The Extra Trees algorithm was part of a stacked model built on top of a Linear Regression with Elastic Net Regularization. There were three major problems that I encountered while developing the model:

· Algorithm training and predicting time

· Diminishing returns to algorithm optimization

· Limited number of testing opportunities

Implementing basic algorithms in Scikit-learn is relatively simple because of the high level of abstraction within the library. The actual theory behind the algorithms is hidden in favor of usability. Moreover, the default parameters are given reasonable values to allow a user to have a usable model up and running quickly.In this project, I found it relatively simple to get the baseline model functioning with a decent coefficient of determination. However, as in most machine learning projects, the law of diminishing returns soon crept in with regards to hyperparameter tuning. [33] Improving the performance of any model even a few tenths of a percentage point required a significant time investment in tuning the algorithm. The full effect of these diminishing returns can be seen in the complexity of the final model. As the development of the model progressed, it became difficult to determine what steps to take to improve the performance. Mostly, I tried to combat this by using Grid Search to find the optimal parameters. This problem was an example of the effectiveness of data, a phenomenon where the quality and preprocessing of the training data has much greater importance to the final model performance than the tuning of the algorithm.

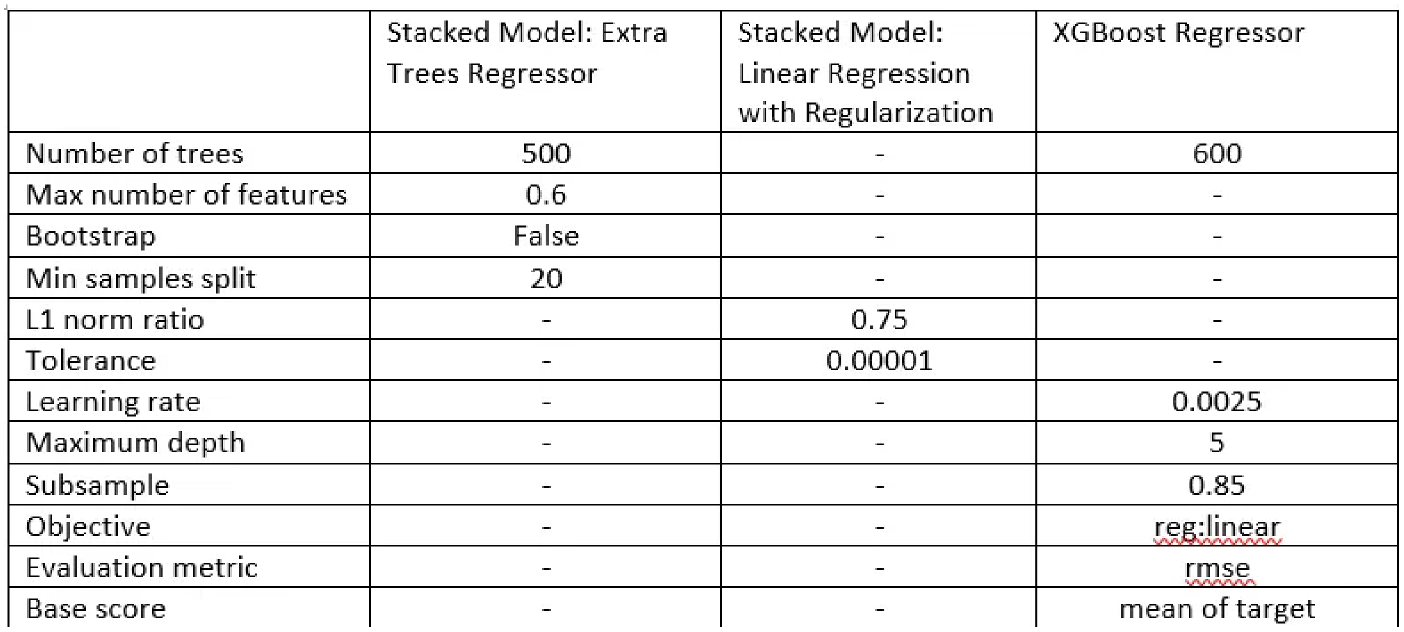
Once the structure of the final algorithm had been selected, the next step was to optimize the model. This mainly consisted of what is known as hyperparameter tuning. (The word 'hyperparameter' refers to an aspect of the algorithm set by the programmer before training, such as the number of trees in a Random Forest, or the number of neighbors to use in a Nearest Neighbors implementation. The word 'parameters' refers to the learned attributes of the model, such as the coefficients in a linear regression).I like to think of this step as adjusting the settings of the model to optimize them for the problem. This can be done manually, by changing one or more hyperparameters at a time and then checking the model performance. However, a more efficient method for parameter optimization is to perform a grid search. There are two primary aspects to this method in Scikit-learn with the GridSearchCV class:

· Hyperparameter search

· Cross validation

The idea of grid search is to define a set of hyperparameters, known as a grid, and then allow an algorithm to test all of the possible combinations. Most of the default hyperparameters in Scikit-learn are reasonable, and my methodology when performing a grid search was therefore to try values for parameters both above and below the default value. Depending on which configuration returned the best score, I would then perform another search with the hyperparameters moving in the direction that increased scores. The CV in GridSearchCV stands for cross validation, which is a method to prevent overfitting on the training data. The concept behind cross-validation is that the data is first split into a training and testing set. Then, the training data is split into 'n' smaller subsets, known as folds. Each iteration through the cross validation, the algorithm is trained on n-1 on these subsets and evaluated on the nth subset. The final score of the algorithm is the average score across all of the folds. This prevents overfitting whereby the model learns the training data very well, but then cannot generalize to new data that it has not seen before. After n iterations of cross-validation have been completed, the model is evaluated one final time on the test set to determine the performance on a previously unseen set of data. Cross-validation is combined with Grid Search to optimize the algorithm for the given problem while also ensuring that the variance of the model is not too high so that the model can generalize to new instances. GridSearchCV allows a wide range of models to be evaluated more efficiently than manually checking each option.

The final hyperparameters for the constituent models that make up the final model are presented Table below. Any hyperparameters not specified in the table were set to the default value in Scikit-learn.



	Stacked Model: Extra Trees Regressor	Stacked Model: Linear Regression with Regularization	XGBoost Regressor
Number of trees	500	-	600
Max number of features	0.6	-	-
Bootstrap	False	-	-
Min samples split	20	-	-
L1 norm ratio	-	0.75	-
Tolerance	-	0.00001	-
Learning rate	-	-	0.0025
Maximum depth	-	-	5
Subsample	-	-	0.85
Objective	-	-	reg:linear
Evaluation metric	-	-	rmse
Base score	-	-	mean of target

GridSearchCV was performed on both intermediate models (the XGBoost model and the stacked model of an Extra Trees regressor stacked on a regularized Linear Regression) individually. The main hyperparameters adjusted for the XGBoost model were the number of decision trees used, the maximum depth of each tree, and the learning rate. The number of trees is simply the number of trees built by the algorithm; the maximum depth of each tree controls the number of levels in each tree and can be used to reduce overfitting or increase the complexity to better learn the relationships within the data; and the learning rate is used in gradient descent to determine how large a step the algorithm takes at each iteration. Compared to the defaults, the learning rate was decreased, the maximum depth of each tree was decreased, and the number of trees was increased. The main parameters adjusted for the other intermediate model were the l1 ratio and tolerance for the Linear Regression and the maximum features, minimum number of samples per leaf, and the number of estimators for the Extra Trees Regressor. The l1 ratio for Elastic Net controls the penalty assigned to the model parameters (it specifies the blend between Ridge and Lasso Regression) and the tolerance is a minimum step size for the algorithm to continue running. In terms of the Extra Trees regressor, the maximum features is the number of features each tree considers, the minimum number of samples per leaf is the minimum number of data points that must be in each leaf node, and the number of estimators is the number of decision trees in the forest. The l1_ratio was increased for Elastic Net which has the effect of increasing the penalty (if the ratio is 1.0, then Elastic Net is equivalent to Lasso Regression which tends to eliminate the least important features by setting the weights close to zero). The maximum number of features for the Extra Trees Regressor was decreased which means that the model did not need to use all the features. Both of these adjustments suggest to me that 140 principal components may have been too many because both models performed implicit feature selection through the hyperparameters. However, keeping too many principal components and then having some not used by the model is preferable to not having enough features and therefore discarding useful information.

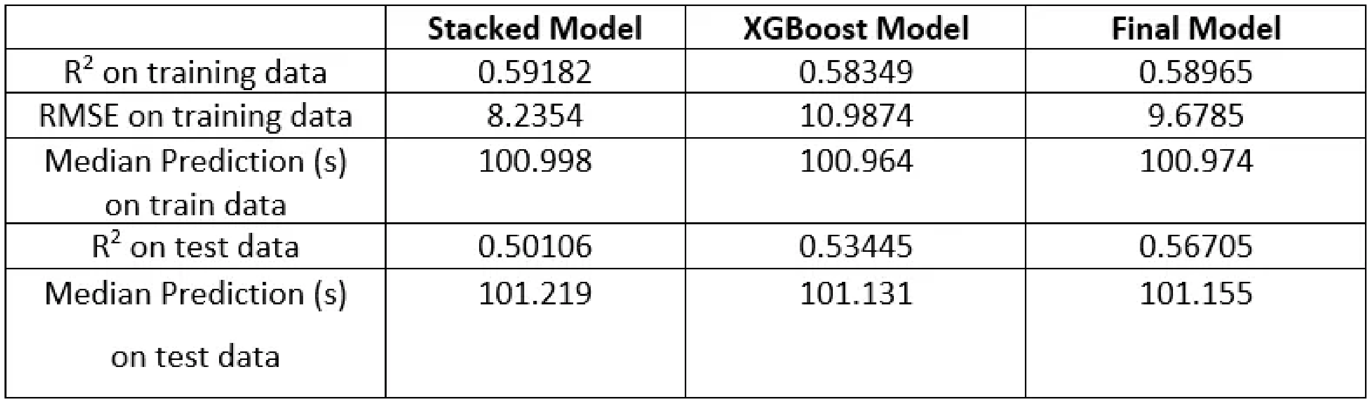
The final and most critical hyperparameter was the weighting for the averaging between the two intermediate models. I determined that the best method was to try a range of ratios and submit the resulting predictions to the competition. The final model placed more weight on the XGBoost predictions. However, increasing the contribution from the XGBoost regressor reached a point of diminishing returns above 0.75. Averaging the predictions from the XGBoost model and the stacked model resulted in a significant improvement in the coefficient of determination compared to each model individually

Model Testing And Evaluation:

The final model was optimized through a combination of Grid Search with cross validation and manually adjusting the final weighting attributed to each of the intermediate models. Both the stacked model and the XGBoost model were tuned using Grid Search cross validation with the training set. The relative weighting of each intermediate model was determined by varying the ratio and submitting the predictions made on the test set to the Kaggle competition. Although I am usually opposed to manually performing an operation, this was the only option for optimizing the weighting for the test data. I could have optimized the weighting for the training data, but as can be seen in the results tables, the coefficient of determination score on the training data did not necessarily correlate with that on the testing data. Moreover, through linear interpolation, it was possible to find the optimal weighting for the test data with a minimal number of submissions.

The final model accomplishes the objective of the problem as stated by Mercedes-Benz. The model is able to explain 56.605% of the variance in the testing data which means that it can account for more than half of the variation in vehicle testing times based on the vehicle configuration. Although this percentage may seem low in absolute terms, in relative terms, it is reasonably high. The top scores on the Kaggle leaderboard21 as of June 25 are near 0.58005 indicating that there is an upper limit to the explanatory power of the data provided by Mercedes-Benz. No model will ever be able to capture all of the variance with the provided data, and if anything, this competition demonstrates that if Mercedes wants to improve predictions further, it will need to collect more and higher quality data. The performance of any machine learning model is limited by the quality of the data (again underscoring the "Effectiveness of Data") [34] and in this problem, the data cannot to explain all of the difference in vehicle testing times.

The final results of the model on the training and testing data are shown below in Table

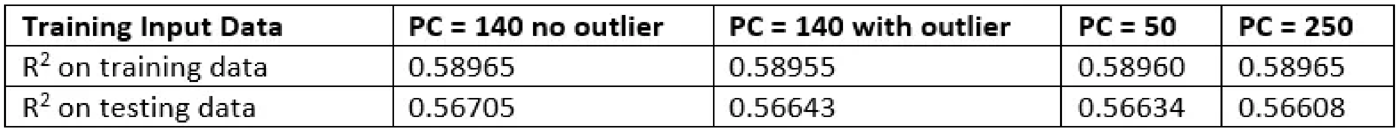


	Stacked Model	**XGBoost Model**	**Final Model**
R^2 on training data	0.59182	0.58349	0.58965
RMSE on training data	8.2354	10.9874	9.6785
Median Prediction (s) on train data	100.998	100.964	100.974
R^2 on test data	0.50106	0.53445	0.56705
Median Prediction (s) on test data	101.219	101.131	101.155

There are number of important conclusions to be drawn from these metrics. The first is that the stacked model performs better on the training data but worse on the testing data than the XGBoost model. This suggests that the stacked model is overfitting the training data and the XGBoost model is better able to generalize to unseen data. In other words, the XGBoost model has a lower variance and higher bias than the stacked model.

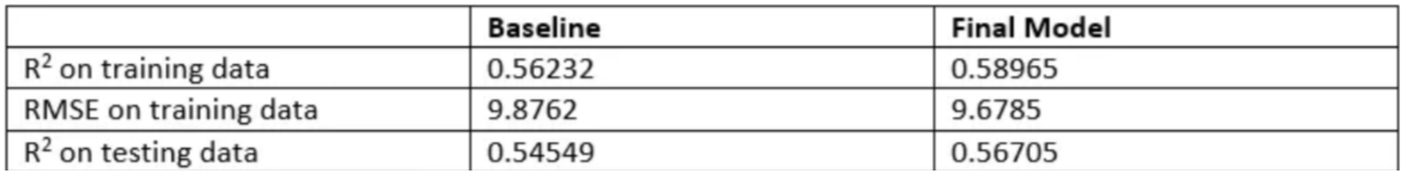
Therefore, averaging the two models means that the weaknesses of one are partly cancelled out by the strengths of the other. The final model does not perform as well on the training data as the stacked model, but it significantly outperforms both intermediate models on the testing data. The averaging of the two intermediate models is an acceptable solution to the problem because it leads to a model that is better equipped to handle novel vehicle configurations.

The final model is also more robust to changes in the input because it is an average of two different models. In order to determine the sensitivity of the final model, I manipulated the training inputs and observed the change in coefficient of determination on the testing and training set. I trained with and without the outlier using 140 principal components, and then altered the number of principal components and again recorded the coefficient of determination.



Training Input Data	**PC = 140 no outlier**	**PC = 140 with outlier**	**PC = 50**	**PC = 250**
R^2 on training data	0.58965	0.58955	0.58960	0.58965
R^2 on testing data	0.56705	0.56643	0.56634	0.56608

Across all training inputs, the R^2 score is very consistent. The model is therefore robust to variations in changes in the training data. If the results were highly dependent on the training data, it would indicate that the model is not robust enough for the task which is not the observed situation. Final results from the model can be trusted as indicated by the coefficient of determination score on the testing set. Although the model does not explain all of the variation in vehicle testing times, it explains more than half the variance and performs in the top 25% of the 3000 models submitted to the competition. The model would not be able to predict vehicle testing times for other companies because it depends on the precise features measured by Mercedes. However, given the problem objective of predicting Mercedes-Benz vehicle testing durations based on vehicle configuration, the model is useful.

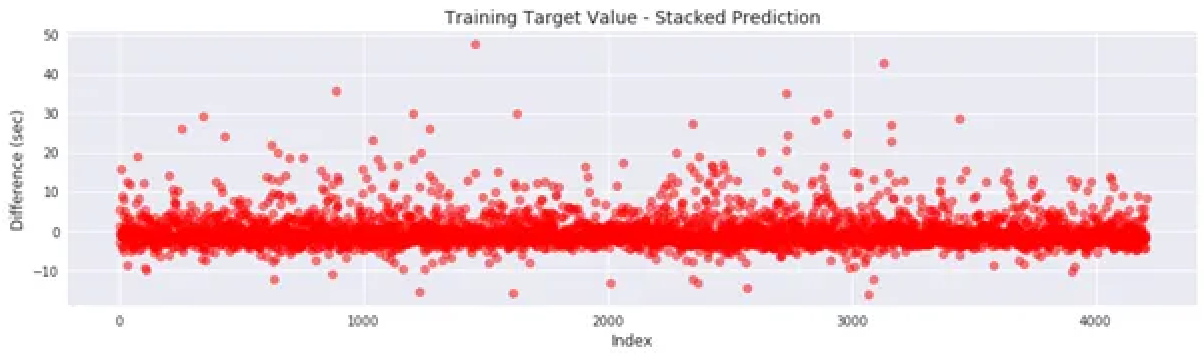


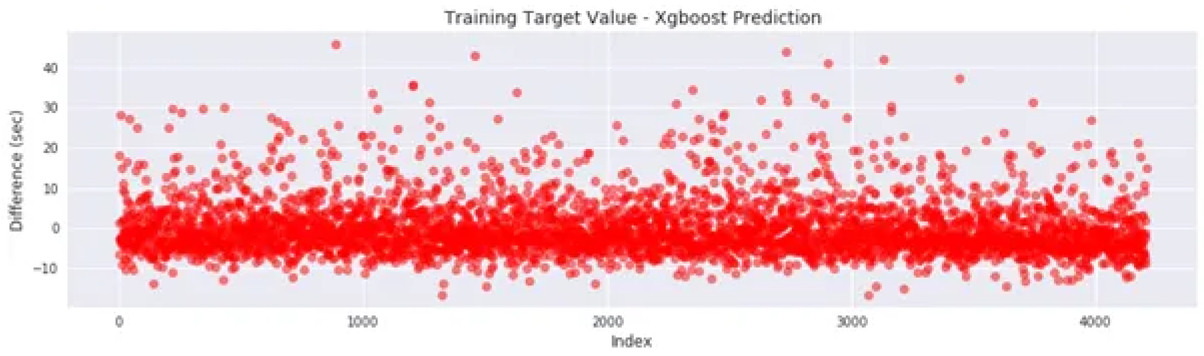
	Baseline	Final Model
R^2 on training data	0.56232	0.58965
RMSE on training data	9.8762	9.6785
R^2 on testing data	0.54549	0.56705

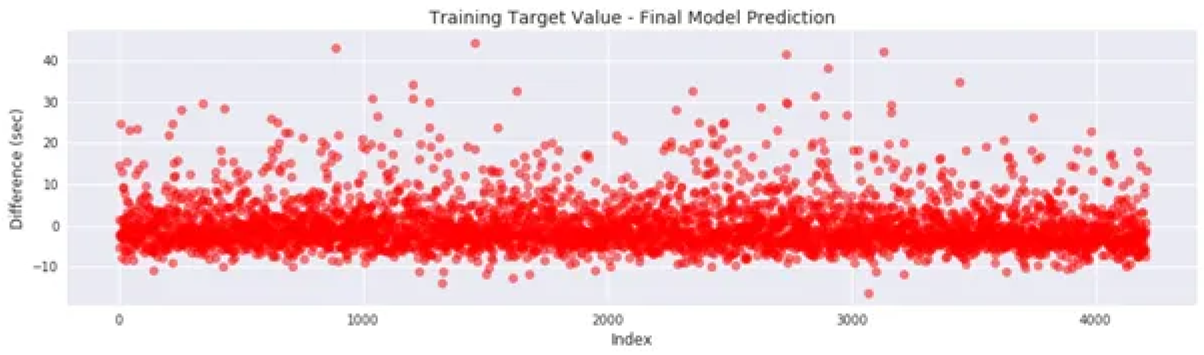
The final model outperforms the baseline model by 4.86% in terms of coefficient of determination on the training set and by 3.95% in the same metric on the testing set. While these numbers may not seem large in absolute terms, they represent a significant improvement. These results again show the diminishing returns of model optimization and the need for more quality and quantity of data. Even with a vastly more complex model, the overall performance did not vastly outperform the baseline and if Mercedes-Benz wants to obtain a better model, it will need to concentrate on obtaining more data. Nonetheless, a 4% improvement over the baseline could represent millions of dollars saved for a company that tests tens of thousands of cars on a yearly basis. In conclusion, the final model accomplishes the problem objective more successfully than the baseline and would improve the Mercedes-Benz vehicle testing process if implemented. As there is diminishing returns with respect to model optimization, additional improvements in the problem realm will only come from using more data.

Reporting of Results:

In order to visually demonstrate the benefit of averaging models, I graphed the difference between the known target values and the predictions made by the stacked intermediate model, the XGBoost intermediate model, and the combined model.







This figure clearly illustrates the benefit of combining predictions from different models. The R^2 measure for the stacked model on the training data was much higher than that of the XGBoost model suggesting the stacked model overfits the training data. This can be seen by comparing the top graph of above Figure to the middle graph as the spread of differences between the known target values and the predicted target values is greater for the XGBoost model. The XGBoost model therefore has greater bias and less variance than the stacked model. Subsequently, the XGBoost model does not overfit the training data to the same degree as the stacked model, but it also might ignore some of the underlying relationships between the features and the target. While the stacked model does overfit to an extent, averaging in the prediction from the stacked model resulted in a better evaluation metric on the testing set. This indicates that the stacked model may be capturing a correlation on the training data that the XGBoost model did not incorporate while training. Overall, the final model is better able to handle new data because each weakness of the intermediate models is partially cancelled by averaging their predictions.