Machine Learning Final Project

AUTHORS
Adam Lodrik
Favre Stefan
Macaraeg Jeff

AFFILIATION
University of Lausanne

PUBLISHED
May 19, 2024

ABSTRACT
Our Machine Learning Final Project is aimed at developing a predictive model capable of identifying the make (i.e. the brand) of a vehicle based on its characteristics, including for example consumption, range, and fuel type. To achieve this, we employed advanced machine learning algorithms, specifically classification trees (CT) and neural networks (NN). Recognizing that certain vehicles might exhibit identical characteristics yet differ across other dimensions, we aimed to create a robust model proficient in accurately predicting a vehicle's brand from its features. Additionally, we explored the potential of clustering our models using unsupervised learning techniques. This project not only reinforced our understanding of machine learning concepts but also demonstrated their practical applicability in addressing real-world problems.

```
Warning in fun(libname, pkgname): couldn't connect to display ":0"
```

1 Introduction

During our first year as master's students in Management with a focus on Business Analytics, we had the opportunity to attend lectures on Machine Learning. This course covered a range of machine learning techniques applicable to business contexts. We explored supervised methods such as regressions, decision trees, support vector machines, and neural networks, as well as unsupervised methods like clustering, Principal Component Analysis (PCA), Factor Analysis of Mixed Data (FAMD), and auto-encoders. Additionally, the course addressed essential topics such as data splitting, ensemble methods, and performance metrics.

In the context of this course, our group undertook an applied project aimed at integrating theoretical knowledge with practical application. Starting from scratch, we identified a suitable dataset to apply machine learning techniques learned in class to real-world scenarios. We selected a comprehensive dataset encompassing vehicle metrics such as MPG, range, engine statistics, and more, spanning over 100 different brands.

The primary aim of our research was to develop a predictive model that can identify the make (i.e. the brand) of a vehicle based on its attributes, such as consumption, range, and fuel type. To accomplish this, we utilized advanced machine learning algorithms, specifically classification trees (CT) and neural networks (NN). We acknowledged that some vehicles might share identical characteristics but differ in other aspects, so our goal was to create a robust model capable of accurately predicting a vehicle's brand from its features. Additionally, we investigated the potential of clustering our models using unsupervised learning techniques. This project not only deepened our understanding of machine learning concepts but also demonstrated their practical applicability in solving real-world problems.

The dataset was sourced from data.world, a platform that hosts a diverse range of open-access datasets. It comprises over 45,000 rows and 26 columns, with each row representing a unique vehicle and its corresponding features. The dataset includes comprehensive information on the vehicle's make, model, year, fuel type, consumption per barrel, highway miles per gallon (MPG), among other features. It is available in CSV format, which facilitates ease of access and analysis.

Given the nature of our objective, being to predict the make of the vehicle, we chose classification models.

In the subsequent sections, we will first provide a comprehensive description of our dataset in the `Data Description` section, followed by a detailed Exploratory Data Analysis (EDA) in the `EDA` section. After an in-depth examination of the data, we will proceed to the data cleaning and preparation phase in the `Data Cleaning` section. Subsequently, we will outline the various methodologies employed in the `Methods` section. Utilizing the cleaned dataset, we will implement our initial model, a classification tree, in the `Classification Tree` section to predict the make of the vehicle. This will be followed by the application of a neural network model in the `Neural Networks` section. We will then transition to unsupervised learning techniques such as the Principal Component Analysis and the Clustering in the `Unsupervised Learning` section. Our study will converge in the `Recommendations and Discussion` section, where we will synthesize our results and explore the implications of our findings. The document will conclude with a `References` section, finishing with an `Annex` section.

2 Data description

For this project, we selected a dataset focused on vehicle characteristics, available as a .csv file from data.world. You can access the dataset via the following link: data.world. It includes a total of 26 features describing 45,896 vehicle models from 141 brands released between 1984 and 2023. Below is a table providing an overview of the available features and their descriptions. You can find a deeper description of the data in the Annex 12.3. The analysis of the missing values will be dealt with in the data cleaning section.

2.0.1 Description of the features

Variable Name	Explanation
ID	Number corresponding to the precise combination of the features of the model
Model Year	Year of the model of the car
Make	Brand of the car
Model	The model of the car
Estimated Annual Petroleum Consumption (Barrels)	Consumption in Petroleum Barrels
Fuel Type 1	First fuel energy source, only source if not an hybrid car
City MPG (Fuel Type 1)	Consumption of the car in miles per gallon of fuel when driving in a city, for fuel type 1
Highway MPG (Fuel Type 1)	Consumption of the car in miles per gallon of fuel when driving on a highway, for fuel type 1
Combined MPG (Fuel Type 1)	Combined city and highway car consumption in miles per gallon, for fuel type 1
Fuel Type 2	Second energy source if hybrid car
City MPG (Fuel Type 2)	Consumption of the car in miles per gallon of fuel when driving in a city, for fuel type 2
Highway MPG (Fuel Type 2)	Consumption of the car in miles per gallon of fuel when driving on a highway, for fuel type 2
Combined MPG (Fuel Type 2)	Combined city and highway car consumption in miles per gallon, for fuel type 2
Engine Cylinders	Number of cylinders of the car
Engine Displacement	Measure of the cylinder volume swept by all of the pistons of a piston engine, excluding the combustion chambers
Drive	Type of powertrain distribution system that places rotational propulsion, such as rear-wheel, 4-Wheel Drive,...
Engine Description	Description of some features of the car, such as turbo engine, Stop-Start system, ...
Transmission	Manual/Automatic transmission, with number of gears and/or model of transmission
Vehicle Class	Type of vehicle, such as Minivan, Trucks,...
Time to Charge EV (hours at 120v)	Number of hours required to fully charge an EV car at 120v
Time to Charge EV (hours at 240v)	Number of hours required to fully charge an EV car at 240v
Range (for EV)	Maximum number of miles possible with a fully charged EV car
City Range (for EV - Fuel Type 1)	Maximum number of miles possible with a fully charged EV car in a city
City Range (for EV - Fuel Type 2)	Maximum number of miles possible while only using electricity with a fully charged hybrid car in a city
Hwy Range (for EV - Fuel Type 1)	Maximum number of miles possible with a fully charged EV car on a highway
Hwy Range (for EV - Fuel Type 2)	Maximum number of miles possible while only using electricity with a fully charged hybrid car on a highway

3 Exploratory Data Analysis

Now that we have presented the variables contained in the dataset, let's try to understand the data structure, characteristics and underlying patterns thanks to an exploratory data analysis.

3.0.1 Dataset overview

Find below a preview of our raw dataset.

Show 5 entries Search:

make	ID	model_year	model	estimated_Annual_Petrolum_Consumption_Barrels	fuel_type_1	City_MPG
Acura	1833	1986	Integra	12.39625	Regular Gasoline	
Acura	1834	1986	Integra	11.9004	Regular Gasoline	
Acura	1991	1986	Legend	15.65842105	Regular Gasoline	

make	ID	model_year	model	estimated_Annual_Petrolum_Consumption_Barrels	fuel_type_1	City_MPG
Acura	1992	1986	Legend	14.8755	Regular Gasoline	
Acura	3037	1987	Integra	12.39625	Regular Gasoline	

Showing 1 to 5 of 45,896 entries Previous 1 2 3 4 5 ... 9,180 Next

3.0.2 Columns description

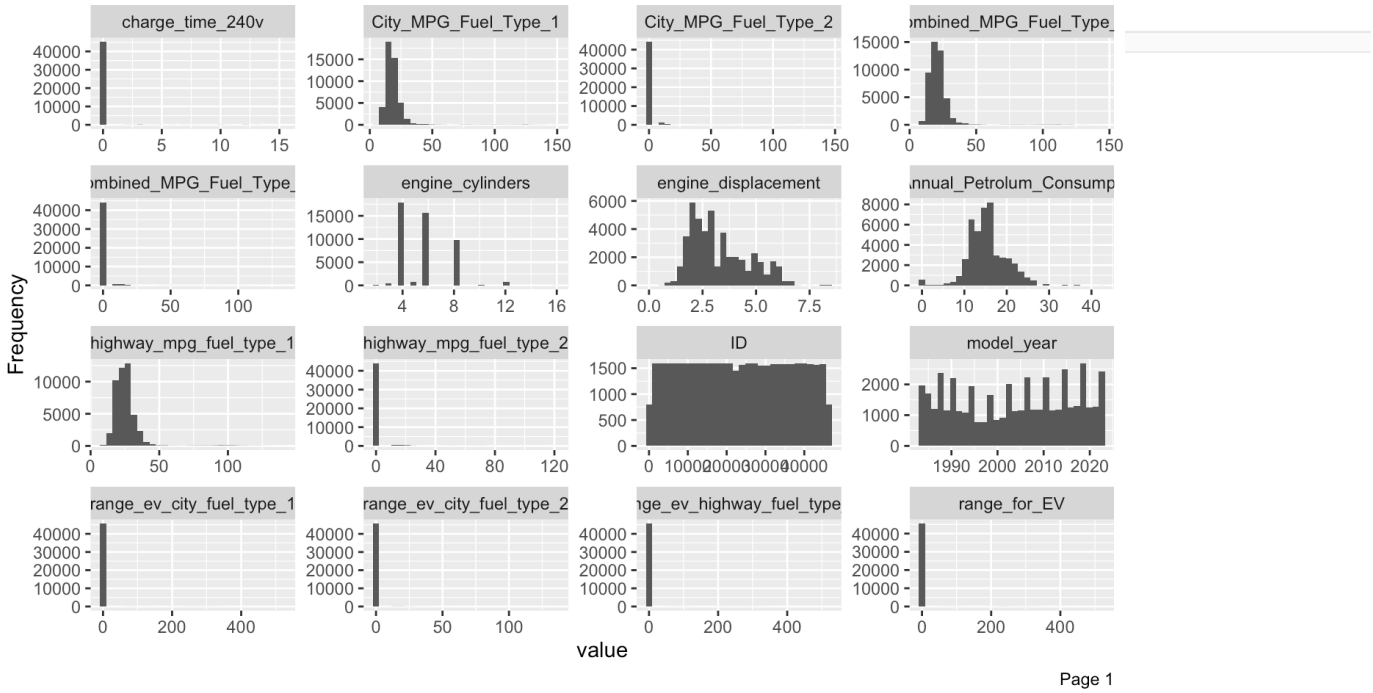
Let's have a quick look at the characteristics of the columns. You will find more statistical details about it in the Annex 12.1.

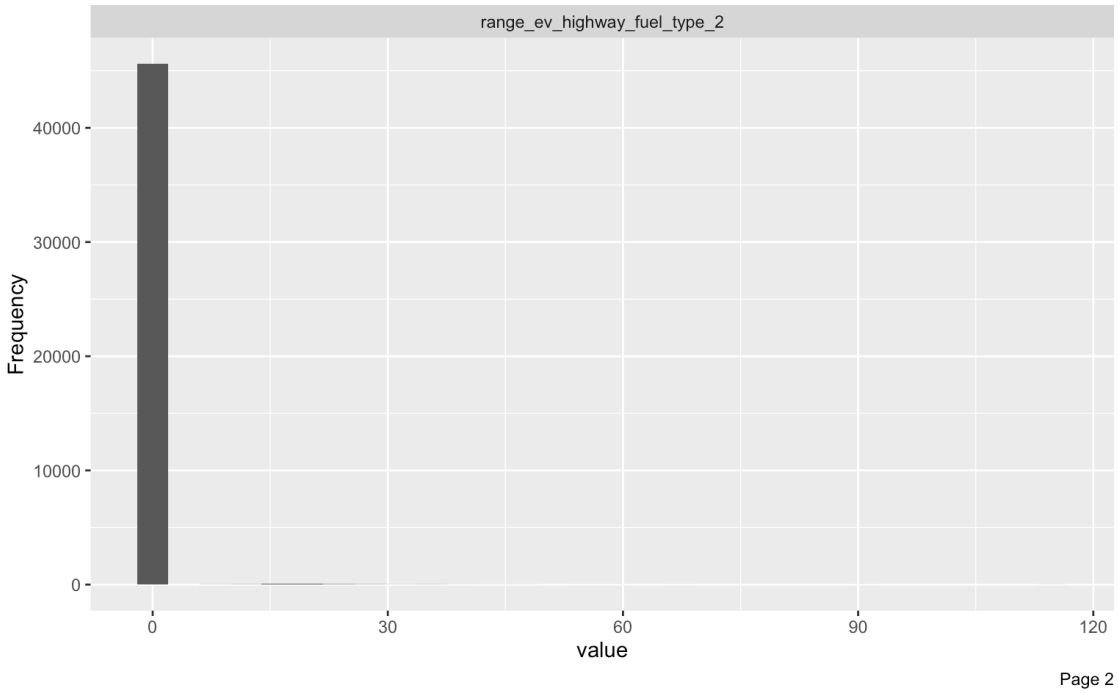
Name	Number_of_rows	Number_of_columns	Character	Numeric	Group_variables
data	45896	26	8	18	None

The dataset that we are working with contains approximately 46'000 rows and 26 columns, each row representing a model from one of the 141 brands. From the data overview, we can see that most of our features are concerning the consumption of the cars. If we now check more in details in the annex, we notice that some variables contain a lot of missing and that the variable "Time.to.Charge.EV..hours.at.120v." is only containing 0s. We will handle these issues in the section "Data cleaning".

3.0.3 Exploration of the distribution

Now let's explore the distribution of the numerical features.

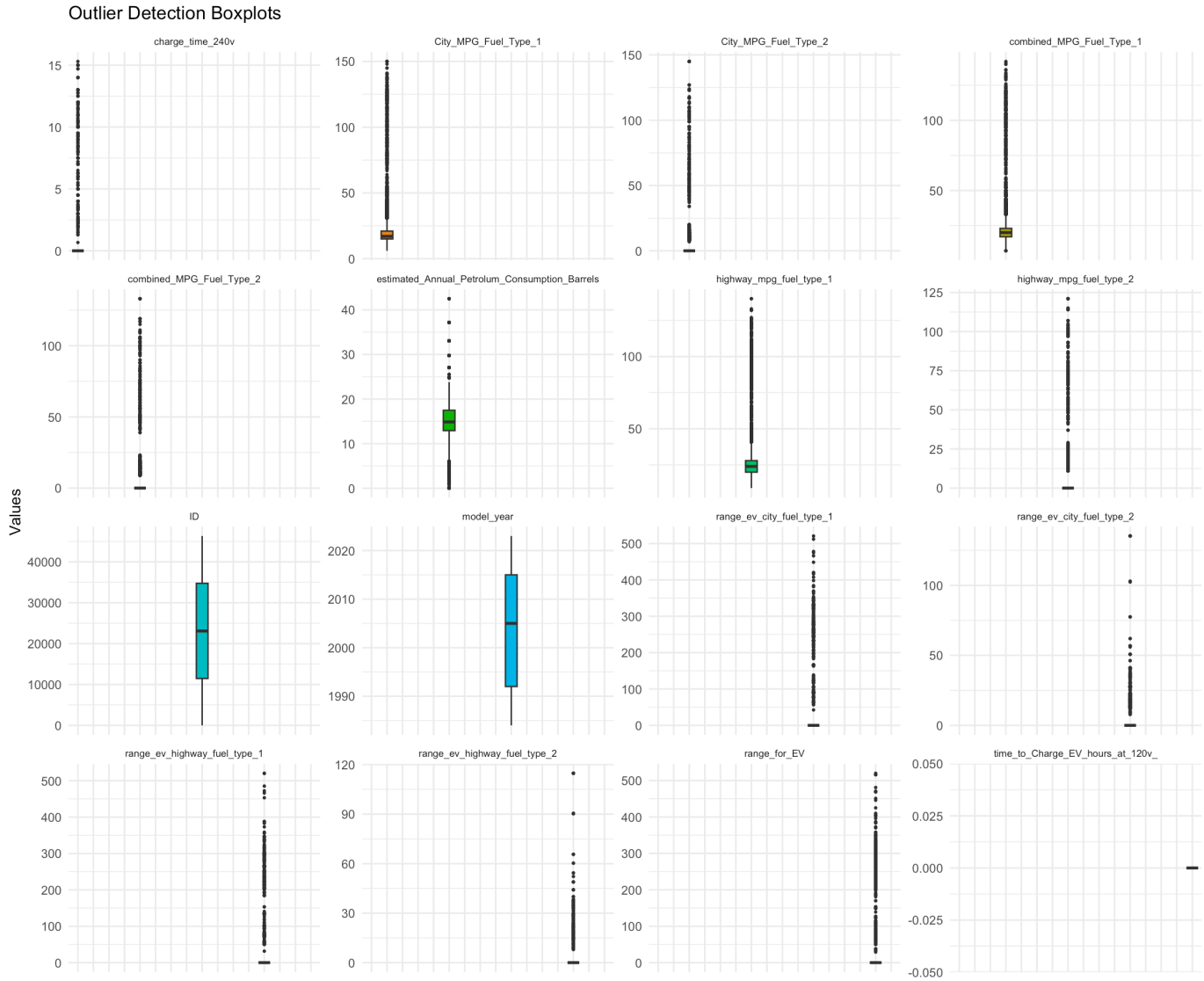




As the majority of models in our dataset are neither electric vehicles (EVs) nor hybrid cars and because of the nature of some column concerning only these two types of vehicles, the results are showing numerous zero values in several columns. This issue will be addressed during the data cleaning process. Additionally, certain features, such as “Engine Cylinders,” are numerically discrete, as illustrated in the corresponding plot.

3.0.4 Outliers Detection

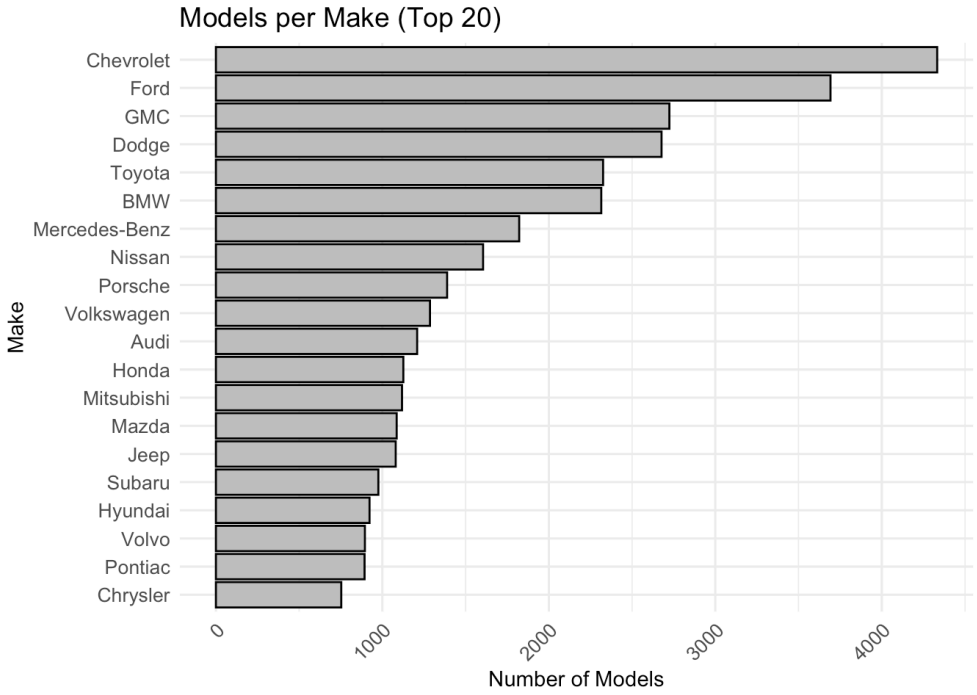
In order to identify occurrences that deviate significantly for the rest of the observations, and in order to potentially improve the global quality of the data, we decided to analyse outliers thanks to boxplots. Here are the result on the numerical features of the dataset:



Most of our boxplots are showing extreme values. Again, this is due to the small amount of EV and hybrid cars in our dataset compared to the rest of the models and due to the nature of some features, concerning only those type of vehicles.

3.0.5 Number of models per make

Now let's check how many models per make we have in our dataset. In order to have a clear plot, we decided to keep only the top 20 brands among all the make in our dataset to create the plot. All the remaining makes are accessible on in Annex 12.2.

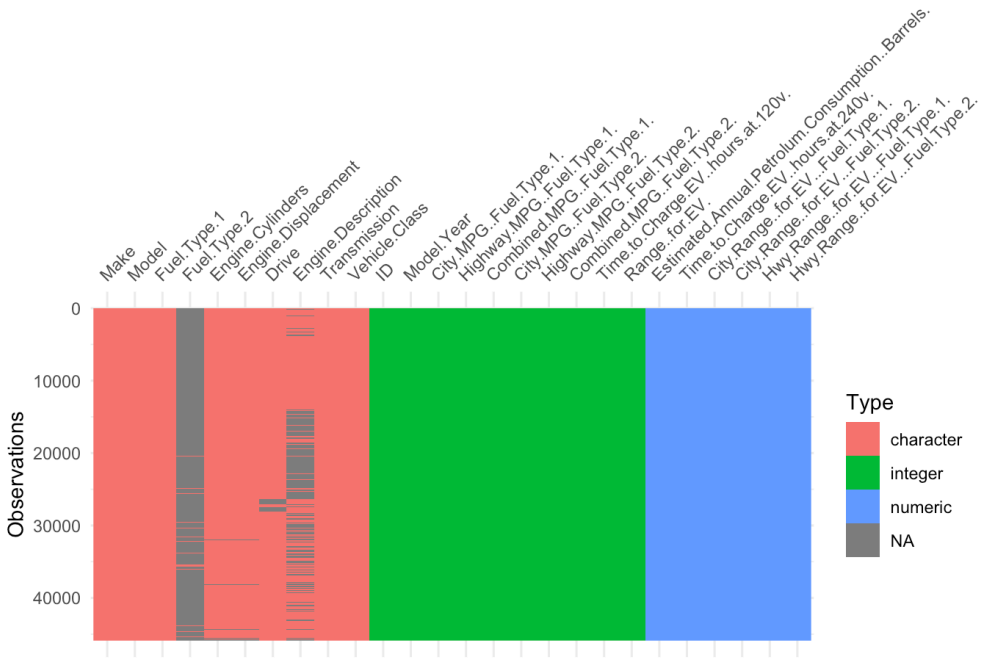


On the 141 brands, we notice that only 13 brands have more than 1000 models in the dataset. Among these, only one of them (Chevrolet) have more than 4000 models presents. In addition, we saw that many car brands have 1 observation, which would lead to class unbalanced and would lead to bias toward majority classes. We will address this issue later.

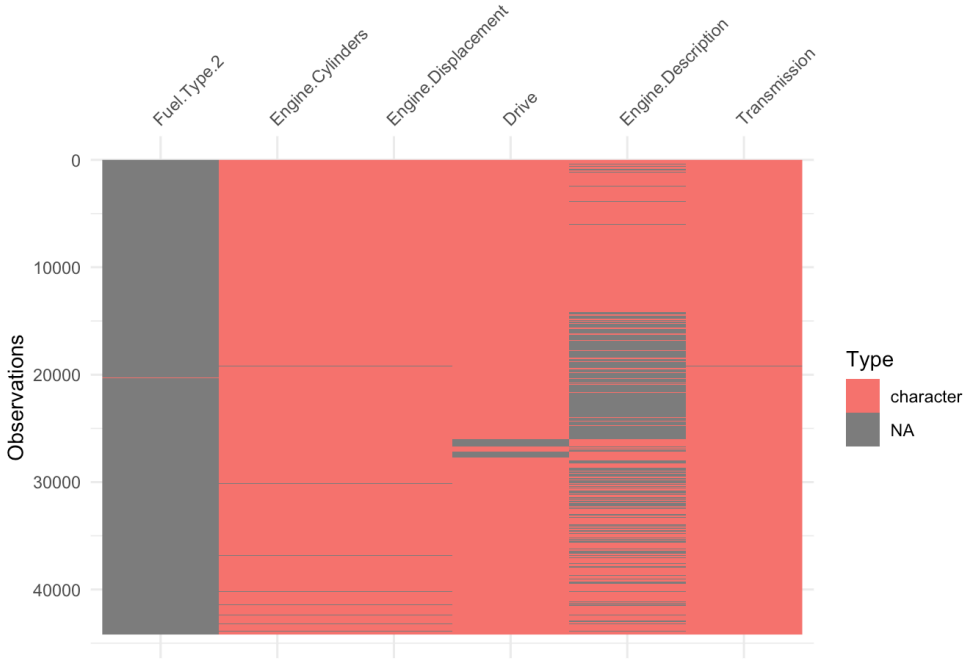
4 Data cleaning

In this section we will handle the missing values of our dataset to make sure that we have a clean dataset to create our model. We will first visualize the missing values of our dataset and then clean the missing values in the columns that we will use for our analysis. We will also remove some rows and columns that are not relevant for our analysis.

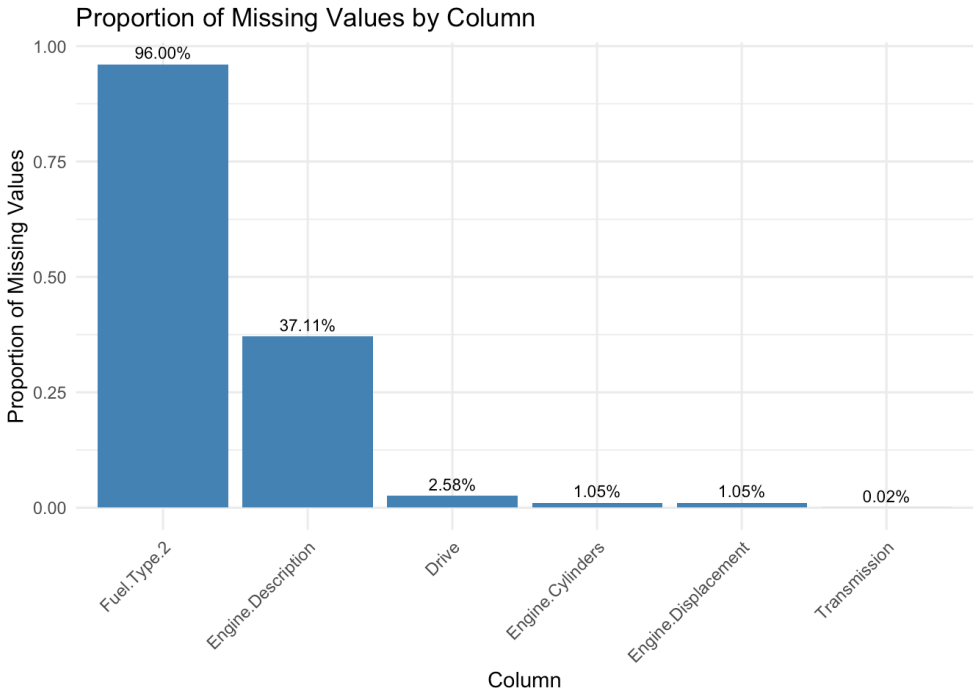
Let's have a look at the entire dataset and its missing values in grey.



We can see that overall, we do not have many missing values in proportion with the size of our dataset. However, we can see that some columns have a lot of missing values. Below we have the detail of the percentage of missing values by columns.



4.1 Dealing with the columns “Engine Cylinders” and “Engine Displacement”



As we can see we have missing in 6 columns. Let's first have a closer look at the engine cylinders and engine displacement columns. They both have 484 missing values. After some data manipulation, we saw that these 484 missing are all electric vehicles and that they all have missing values in the engine cylinders and engine displacement columns. Given that in our dataset we have 484 vehicles, we now that theses missing in these column only concerns electric vehicles. This make sense since electric vehicle do not have an combustion engine and therefore those categories are not really applicable. We will therefore replace all missing values in this two columns with “none”.

Show 5 entries Search:

Column	Missing values	Prop. Missing
Fuel.Type.2	44059	96.00%
Engine.Description	17031	37.11%
Drive	1186	2.58%

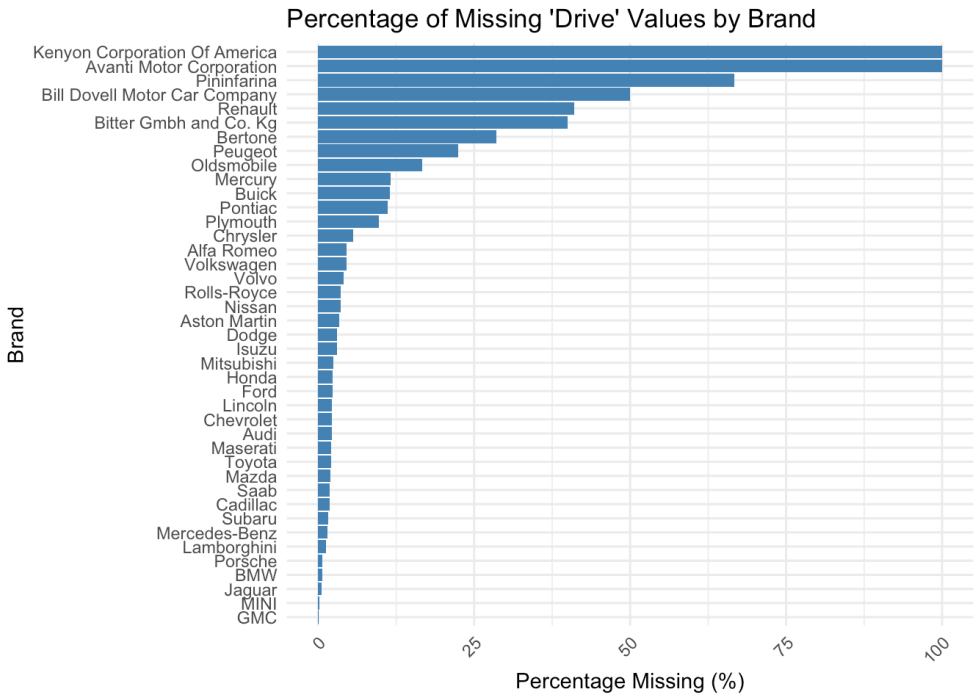
Column	Missing values	Prop. Missing
Transmission	11	0.02%
ID	0	0.00%

Showing 1 to 5 of 26 entries Previous 1 2 3 4 5 6 Next

As we can see, we still have some missing in the columns “Fuel Type 2”, “Engine Description”, “Drive” and “Transmission”. Let’s investigate the missing in the column “Drive”.

4.2 Dealing with the column “Drive”, “Transmission” and “Engine Description”

Let’s have now a closer look at the other features with missing values.



We decided to drop the brand with more than 10% of missing values in the “Drive” column. After this operation, we also removed the 8 observations that remained with missing values in the “Transmission” column. We decided to drop the column engine description since it contains missing values for more than a third of our observations.

Show 3 entries Search:

Column	Missing values	Prop. Missing
Fuel.Type.2	40466	95.80%
Engine.Description	16334	38.67%
ID	0	0.00%

Showing 1 to 3 of 26 entries Previous 1 2 3 4 5 ... 9 Next

4.3 Final dataset

The dataset is now cleaned and does not contain any missing values anymore. It contains 42240 observations, 18 features and 129 brands. We renamed the columns and stored it in a csv file (data_cleaned.csv).

However, for some models, we need to tackle the unbalanced classes in the target variable. For this reason we also created a new .csv file for which we drop the make with less than 10 models (data_cleaned_reduced.csv). This dataset contains 42061 observations, 18 features and 66 brands.

You can find an overview of the cleaned datasets in the Annex 12.4.

5 Methods

This section provides a detailed explanation of the methods used in this study, aimed at ensuring reproducibility. The global strategy and the specific tools employed are described concisely, with relevant properties highlighted. Clear references are provided for further details where necessary.

5.1 Supervised Learning

5.1.1 Classification Tree

As our dataset is dealing with classes, we decided to use a Classification Tree to predict the make of the car.

Classification Trees are constituted of one root node and a hierarchical set of binary rules. Each node represents a decision on an attribute, each branch the outcome of the decision and each leaf a class label (The most important features are located at the top of the graph). Thus, the name is coming from the shape of the representation, that looks like a tree. Even if it can also be applied to regressions, we are using it in our case for classification tasks.

Key parameters and hyperparameters used in Classification Tree include (non-exhaustive):

Parameters

- Data observations
- Method (usually “class”, which specifies that the decision tree being constructed is for classification tasks)

Hyperparameters

- **Minsplit** : minimum number of observations that must exist in a node for a split to be attempted
- **minbucket** : controls the minimum size of terminal nodes
- **complexity parameter (cp)** : controls the size of the decision tree. Also prevents overfitting.
- **Maximum depth** : Set a maximum “length” of the tree, starting from the root node.
- **Random state** : Set a specific number to control the randomness of the estimator.

We had in must of our models to encode categorical variables using Label Encoding to convert them into numerical values. In addition, we had to split the dataset into training (80%) and testing (20%). This step is essential for preventing overfitting, try to select the best model configuration by training the model, then assess the final model performance on our “unseen data” of the testing set. This helps to build a robust model that generalizes well to new data.

5.1.2 Our models

1. Classification Tree without Base model
2. Classification Tree with 10 K-fold CV
3. Classification Tree with Pruning (Setting a limit for the max depth)
4. Classification Tree with Re-Sampling

5.1.3 Classification Tree without constraints

The base model serves as a baseline that will be used to measure the added values of the different techniques that we will apply later.

In our case, the Decision Tree Classifier creates the decision tree. It initializes the process of recursively splitting the dataset into subsets based on the value of the features. It also implicitly defines the loss function, with the two following most common criterias : Gini Impurity (measuring how mixed are the classes in a node, also selecting the best features) and Entropy (different from the Logistic Regression, measure of chaos, indicating the uncertainty reduced by the split).

5.1.4 10 K-fold CV

K-Fold cross-validation is a method to assess the performance of a model. It has the following steps :

- Divide the Data: The dataset is split into k equally sized subsets or “folds.”
- Train and Validate: The model is trained k times. In each iteration, one fold is used as the validation set, and the remaining k–1 folds are used as the training set.
- Evaluate: The model’s performance is evaluated using the validation set in each iteration. Average Performance: The results from each iteration are averaged to provide a single performance metric.

We decided to use a K-fold cross-validation with 10 folds to fight overfitting. The rest remained as in our base classification tree.

5.1.5 Classification Tree with Pruning

Now let's talk about the classification tree using pruning. The idea is to reduce the complexity of the tree by removing unnecessary sections. This helps a lot against overfitting but at the expense of accuracy.

In our case, we have the same components as in the base method at the beginning. Nevertheless, some parts of the process are different. In this case we are using a complexity parameter (max_depth) to control the size of the tree. We tried to play with the tuning of the hyperparameter [5 to 30] to check which value had the best tradeoff between accuracy and overfitting.

5.1.6 Re-Sampling

Re-Sampling is a method used to address class imbalance. As it's name suppose, it creates new observations for the minority classes to match the majority classes, thus balancing the classes.

In our case, we have used a threshold in order to avoid observations with a small number of occurrences in our dataset, which would have reduced the quality of our prediction. We then used the new dataset on our "base" Classification Tree.

5.1.6.1 Software and Tools

- rpart : Used for building classification and regression trees
- pandas : Library used for data manipulation and analysis.
- Scikit-learn: Used for data preprocessing, splitting the dataset, and calculating class weights.
- matplotlib : Used for creating static, animated, and interactive visualizations
- seaborn : Provides a high-level interface for drawing attractive and informative statistical graphics

5.1.6.2 Sequence of Analysis

After loading and importing our necessary functions R and Python functions, we prepared the data by first defining the make as the target variable, then encoded categorical variables using Label Encoding to convert them into dummy variables. This step is essential as Classification Tree needs numerical value to work.

We then split the dataset into a training set (80%) and a testing set (20%). This step is essential to prevent overfitting. We then tried to select the best model configuration by training the model, then assessed the final model performance on our "unseen data" of the testing set. This helps to build a robust model that generalizes well to new data.

Once the separation done, we finally trained our decision tree, using the parameters but first without tuning the hyperparameters that we talked about earlier. Then, after training the model, we have been able to use the model on the test set to compute the accuracies. This step is essential to control the performance of the model, also to control the presence of overfitting. After verification, we found overfitting after constructing the base Classification Tree model. We then decided to use the K-Fold CV to assess the performance of our "base" model.

Then we pruned the tree, by setting different max_depth value [5 to 30] in order to see which tradeoff between accuracy and performance (at fighting overfitting) was the best.

In addition, we decided to address class unbalances by Re-Sampling our dataset. We established a threshold of 100 observations per make to avoid including brands with insufficient data. The resulting dataset was then utilized for our "base" Classification Tree.

5.1.7 Neural Network

We also used a neural network for the classification task. As already discussed, we want to predict the make of a car utilizing the other features.

A neural network is a computational model inspired by the way biological neural networks in the human brain process information. It consists of interconnected layers of nodes (neurons) where each connection has an associated weight. The network learns to map input data to the desired output by adjusting these weights through a process called training. The model can be tuned using hyperparameters set before the learning process begins that governs the overall behavior and performance of the machine learning model.

Key parameters and hyperparameters used in neural networks include:

- **Learning Rate:** Controls how much the model's weights are updated with respect to the gradient of the loss function.
- **Number of Epochs:** The number of complete passes through the training dataset.
- **Batch Size:** The number of training examples used in one iteration to update the model's weights.
- **Number of Layers:** Determines the depth of the network, including input, hidden, and output layers.

- **Number of Neurons per Layer:** The size of each layer, influencing the capacity of the model to learn from data.
- **Activation Functions:** Non-linear functions applied to the output of each neuron, such as ReLU, sigmoid, or softmax.
- **Optimizer:** The algorithm used to update the weights, such as Adam, SGD, or RMSprop.
- **Loss Function:** Measures the difference between the predicted and actual output, guiding the optimizer, e.g., categorical cross-entropy for classification tasks.
- **Dropout Rate:** The fraction of neurons to drop during training to prevent overfitting.
- **Class Weights:** Used to handle imbalanced datasets by giving more importance to underrepresented classes.

5.1.7.0.1 Preprocessing the data

An important step before training our model has been to separate numerical and categorical columns in our data preprocessing because they require different types of handling to prepare them for machine learning algorithms. The numerical columns need to be scaled by adjusting them so they have a mean of zero and a standard deviation of one, which helps the machine learning algorithm perform better. While the categorical columns need to be one-hot encoded which creates a binary column a format that the machine learning model can understand.

5.1.7.1 Our three primary models

1. A Simple Neural Network Model
2. A Class-Weighted Neural Network Model
3. And a Dropout-Enhanced Neural Network Model (also using class-weight)

All three models used the cleaned data set prepared in the data cleaning section. We chose the version with more than 10 observation for each car brands to avoid over imbalanced classes. Each model's architecture, training process, and evaluation metrics are described in the following sections.

5.1.7.1.1 Simple Neural Network Model

The simple neural network model serves as the baseline. It consists of the following components:

- Input Layer: Matches the number of features in the dataset.
- Hidden Layers: Two dense layers with ReLU activation functions.
- Output Layer: A dense layer with a softmax activation function for classification. The softmax activation function output a probability for each features input to belong to a specific class (brand). The model is trained using the Adam optimizer, categorical cross-entropy loss function, and accuracy as the primary metric.

5.1.7.1.2 Class-Weighted Neural Network Model

To address class imbalances, a class-weighted approach has then been applied. This involves assigning higher weights to the minority classes during training. The architecture of the network remains the same as the simple model. The following steps were taken:

- Class Weights Calculation: Inverse proportionality to class frequencies.
- Model Training: Incorporating class weights into the loss function.

The use of class weights helps in penalizing misclassification of the minority classes more heavily, thereby improving the model's ability to predict the minority classes.

5.1.7.1.3 Dropout-Enhanced Neural Network Model

The third model incorporates dropout layers to mitigate overfitting while keeping the modification applied in our second model. Dropout layers randomly set a fraction of neurons to zero at each update during training, which helps prevent the network from becoming overly reliant on specific neurons. The deactivation of some neurons only happens during the training phases. When the model is used against the validation and test set, all neurons are active. This has the consequence of seeing higher accuracy on the validation set than the training set.

- Model Architecture: Similar to the simple model with additional dropout layers after each dense hidden layer.
- Dropout Rate: Will be tuned during the modeling.

The dropout-enhanced model helps improve generalization by reducing the risk of overfitting to the training data.

5.1.7.2 Software and Tools

The following software and tools were used to implement and evaluate the models (non-exhaustive list):

- Python: The programming language used for all implementations.

- TensorFlow and Keras: Libraries used for building and training neural network models.
- Scikit-learn: Used for data preprocessing, splitting the dataset, and calculating class weights.
- Pandas and NumPy: Libraries used for data manipulation and numerical computations.

5.1.7.3 Metrics

In our machine learning analysis, we used several key metrics to evaluate our model. The main metrics are the training, validation and test accuracies. The validation accuracy shows how well the model performs on the validation set, helping us tune the model during training. Training accuracy indicates the model's performance on the training data, which helps identify overfitting. The difference between the training and validation accuracy is useful to detect the presence of overfitting in our models. The test accuracy measures the model's performance on new, unseen data, providing an unbiased evaluation.

We also used the sensitivity to measure the model's ability to correctly identify positive cases and the specificity that measures the ability to correctly identify negative cases. Even though, when we started to create our model, we already knew that we had unbalanced classes, these metrics confirmed this with numbers. The last metric that has been used is the Cohen's kappa that evaluates the agreement between predicted and true labels, accounting for chance, making it more reliable than simple accuracy because it takes into account the number of classes that we want to predict. These metrics together give a comprehensive view of the model's performance.

5.1.7.4 Sequence of Analysis

We first encode the categorical variables and normalized the numerical features. We then trained the models and evaluated each one. By evaluating and investigating the model at each step we managed to deal with the different challenges that we faced. We also managed to tune the dropout rate to insure to keep a good model performance while highly decreasing the case of overfitting we were having.

This sequence ensures a systematic approach to model development and evaluation, providing a clear understanding of each step involved.

5.2 Unsupervised Learning

In order to check for some dependence between features, we decided to proceed with two unsupervised learning methods which are the Principal Component Analysis and the Clustering methods.

5.2.1 Principal Component Analysis

- Features: The features from the data_cleaned dataframe.
- Scaling: Switching the features from categorical to factor, then factor to numerical. Then, we applied a spectral decomposition on the correlation matrix.
- Variance proportion: We look at the proportion of variance relative to the total variance. Then we can look at the eigenvalues and check the cumulative percentage of variance that explains at least 80% of variance. Then, we obtained the number of dimensions that represent the data well.
- Circle of correlation: The circle of correlation allows to check for the correlation between the features. Also, we can summarize into 2 dimensions.
- Cos2: The cos2 allows to interpret the quality of the representation through the dimensions of the circle of correlation.
- Biplot: Visualization of the observations (or individuals) within the 2 dimensions as well as the features from the circle of correlation.
- Screeplot: Indicates the number of dimensions to select, usually it's by looking at the variance proportion (see above). It also indicates if features are correlated. The more dimensions are required, the more the features are independent.

The aim of doing a PCA is to potentially diminish the number of features as we have several features in our dataset. By doing a dimension reduction, we can only select the first few principal components.

5.2.2 Clustering

Doing a clustering analysis allowed us to find some clusters that share similar characteristics measured by the features. In our case, we have decided to go with a partitioning method.

- Partitioning Method: It begins with assigning each instance to one cluster at random, then to compute the centers and then re-assign with the closest cluster center.
- K-means: Method for numerical data.
- Total Within-Cluster Sum of Squares: Sum of the squared distances between each data point and its corresponding cluster centroid. It measures the compactness of the clusters. For every added cluster, the TWCSS decreases.

- Elbow method: It is the plot to show the optimal number of clusters to select with TWCSS. At the “elbow”, we have the optimal number of cluster k to select.
- Cluster Plot: Plot to visualize the clusters.
- Silhouette Plot: Indicates the how goodly-fitted an instance is within its cluster and also how well-separated it is from the other clusters. A good silhouette shows homogeneity within a cluster.

The clustering was aimed to provide better insights on the similarities of the instances. We started by taking the results from the PCA to add clustering in it. For a better visualization, a 3D biplot was of interest to have the overview of the clusters and the features within 3 dimensions. The final aim is to see the link between the features as well as the similarities of the observations.

5.2.2.1 Software and Tools

The following software and tools were used to implement and evaluate the models (non-exhaustive list):

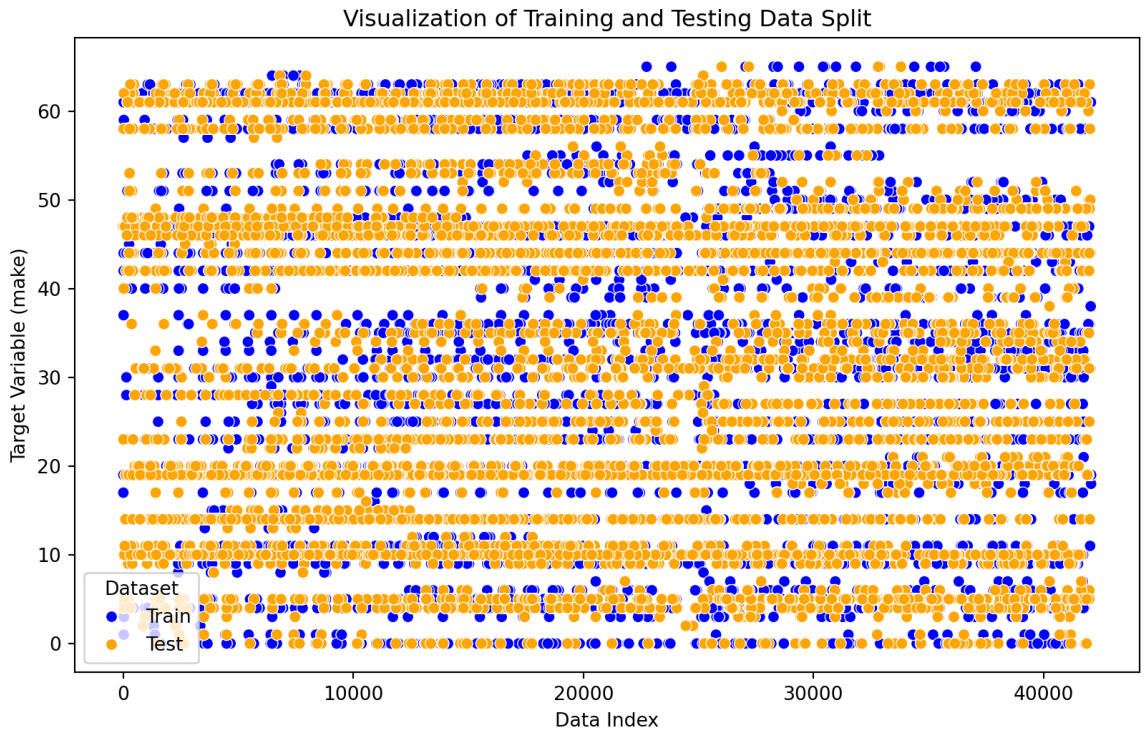
- R Studio: The programming language used to implement the unsupervised learning methods.
- Factoextra: Provides tools to visualize and interpret multivariate data analyses such as the principal component analysis and clustering.
- FactoMineR: Provides functions to visualize and interpret multivariate data analyses such as the principal component analysis and clustering.
- Cluster: Provides methods for cluster analysis such as hierarchical clustering and partitioning methods (K-means, Partitioning Around the Medoid).

Note that this section has been written and is based on the course website (ML_BA). The parts on the dropout layers is based on the following articles: towardsdatascience and python-course

6 Classification Tree

Let’s now dive into the implementation of a classification tree model to predict the make of a vehicle based on its attributes. We first load the necessary packages and the dataset. We then prepared the data to use for the classification tree models.

You will find here an plot showing the actual data splitting effectuated. The x axis represent all the observations in our dataset and the y axis the different brand.



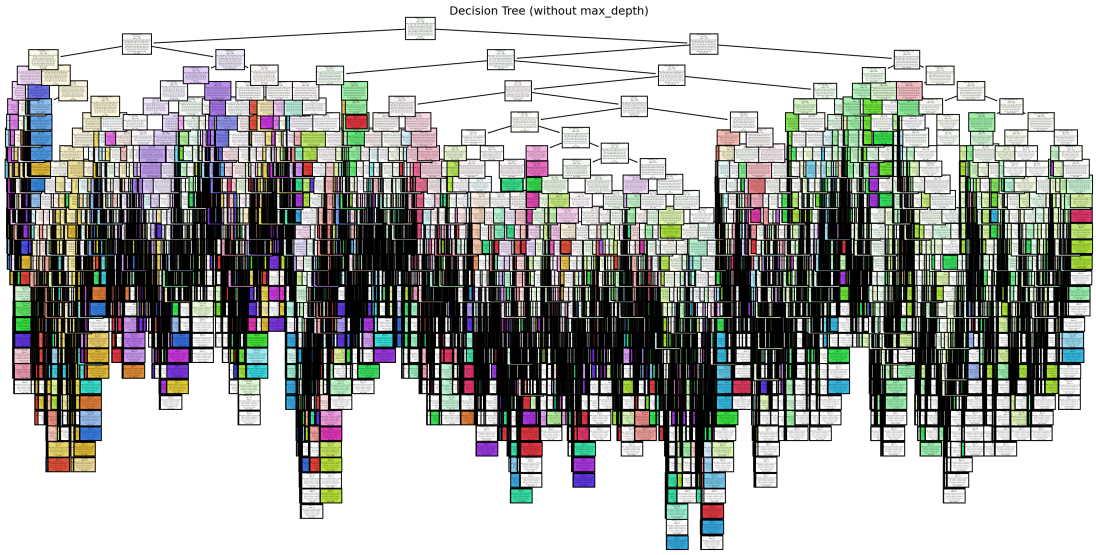
6.0.1 Training the Decision Tree without Pruning

We then proceeded to train the base Decision Tree model. This will allow us to get a baseline. You can find below the accuracy of the training and the visualization of the tree.

DecisionTreeClassifier

```
DecisionTreeClassifier(random_state=42)
```

```
Training Accuracy (without max_depth): 0.8944

Test Accuracy (without max_depth): 0.6940

Test Cohen's Kappa (without max_depth): 0.6796
```



As we can see, the tree resulting is complex, takes a lot of time to train and is difficult to interpret. We can also see that the accuracy of the training set is higher than the test set accuracy and cohen's kappa accuracy, attesting that our model is overfitting. Therefore, let's see the the assessment of the model performance with the K-fold Cross Validation method.

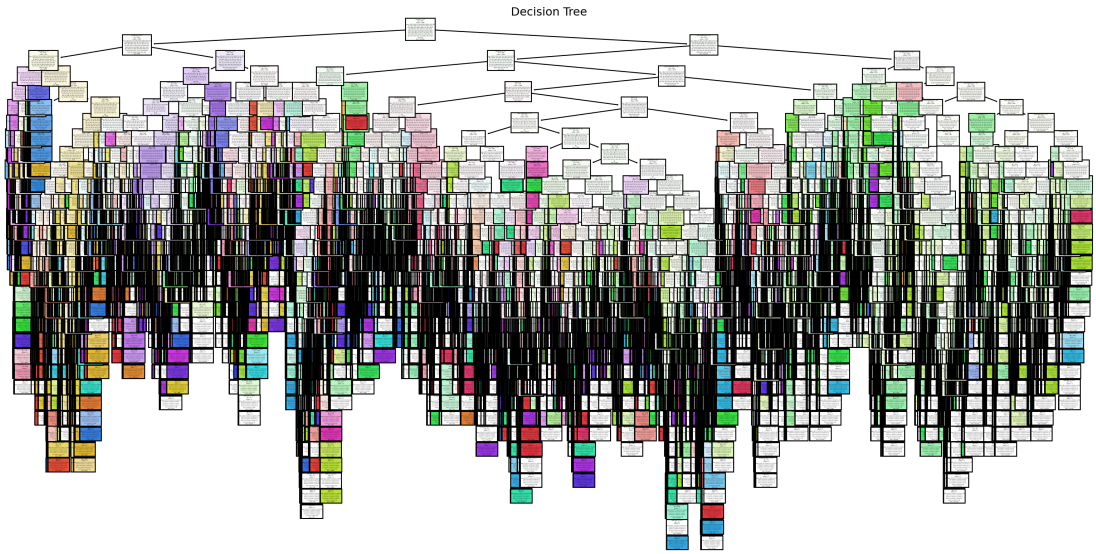
6.0.2 Training the Decision Tree with 10-fold Cross Validation

Here are our results of a 10-fold Cross Validation.

```
Mean cross-validation score:  0.6957039989845513
```

DecisionTreeClassifier

```
DecisionTreeClassifier(random_state=42)
```

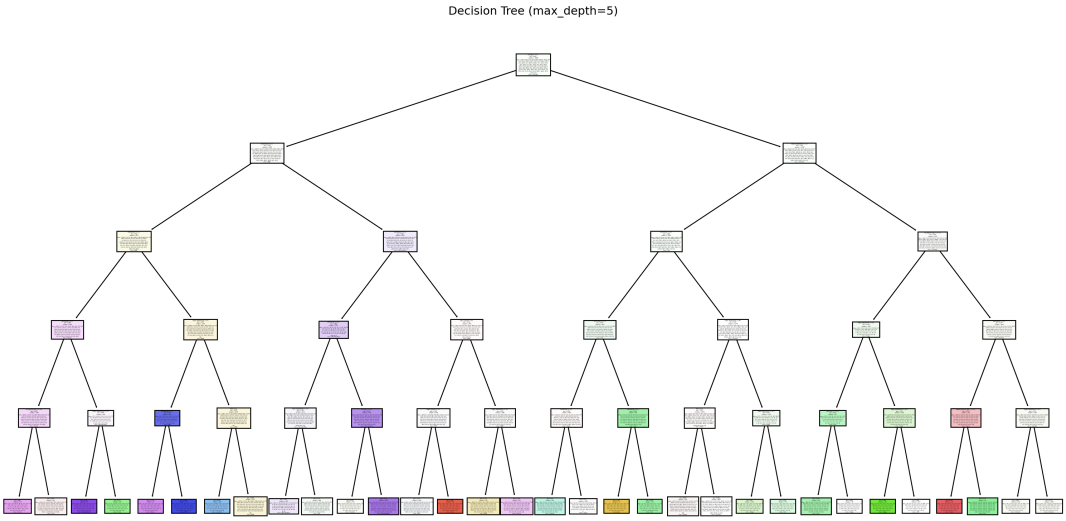


With ten folds, we have not been able to fight overfitting, as the training accuracy remaines higher than the test accuracy, but we notice that our mean accuracy is approx. 70%, which is good. nevertheless, in order have a better model, we need to address overfitting.

6.0.3 Training the Decision Tree with Pruning method

Now we will show you the results with a pruned tree. This one is less complex than the previous trees and fight overfitting. We have set different max_depth parameter values to control the tree's growth (5, 10, 15, 20, 25, 30). We want here to find the optimal tree depth that balances between training and test accuracy. for visibility reasons, we have decided to represent a classification tree with only 5 max_depth. Here are our results:

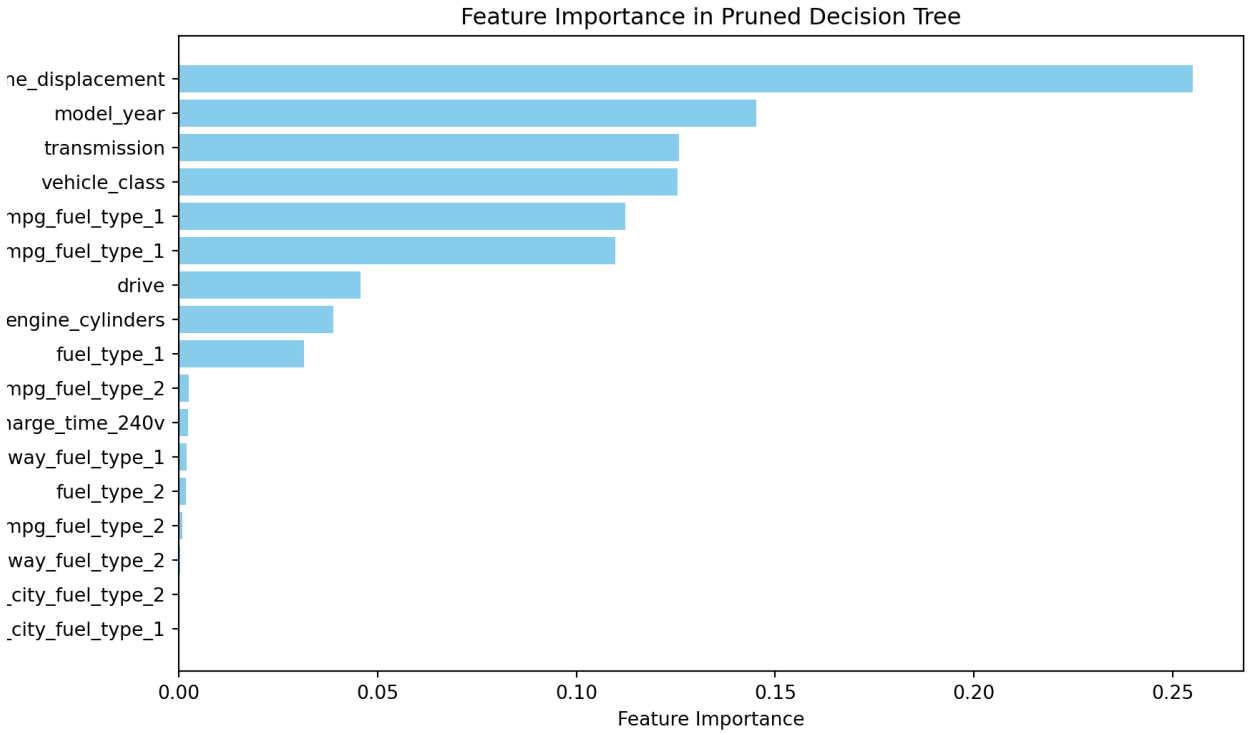
	Train Accuracy	Test Accuracy	Test Cohen's Kappa
5	0.264354	0.259955	0.206755
10	0.497444	0.465946	0.436326
15	0.719508	0.621182	0.602991
20	0.850600	0.682753	0.667804
25	0.889473	0.692856	0.678283
30	0.894199	0.692500	0.677918



The accuracy of the model improved as the depth of the tree increased, with a max_depth of 25 or 30 providing the best test accuracy and Cohen's kappa reaching up to approx. 70%. A max_depth of 5 resulted in significantly lower accuracy, indicating that it is not the optimal choice. A max_depth of 10 or 15 seems to be the best compromise between overall accuracy and avoiding overfitting. This pruning helps to improve the generalization performance of the model by preventing it from becoming too complex and overfitting the training data.

6.0.4 Variable Importance

As we have used a classification tree for the prediction, the most important variable in the model are the ones at the top of the graph. Let's visualize the ranking of the features importance.



We can see that among the top 3 most important features figure in order : the engine displacement, the model of the year and the class of the vehicle. On another hand, we can see that some features such as range_ev_city_fuel_type_2, range_ev_highway_fuel_type_2 and range_ev_city_fuel_type_1 are not important for our model for making predictions. This is certainly due to the quantity of EV and hybrid cars in the dataset and these features that are only concerning these types of vehicles.

6.0.5 Classification Tree with re-sampling

We had seen during our EDA that the classes of the dataset were not well balanced. Now, we are fixing this by re-sampling our classes. Thus, you will find here the number of models per brand equally distributed.

```
         Audi          BMW    Chevrolet        Dodge         Ford
         4239         4239         4239         4239         4239
          GMC        Honda         Jeep        Mazda Mercedes-Benz
         4239         4239         4239         4239         4239
   Mitsubishi       Nissan      Porsche       Toyota   Volkswagen
         4239         4239         4239         4239         4239
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
Training Accuracy (without max_depth): 0.9522

Test Accuracy (without max_depth): 0.8962

Test Cohen's Kappa (without max_depth): 0.8888
```

We notice that thanks to the re-sampling method, the overfitting seems to have disappeared even if it was not the first intention. In addition, as shown, the classes are now balanced. Therefore, the chances of returning any class is equally distributed.

7 Neural Networks

In this section, we will build a neural network model to predict the make of a car based on the features at our disposal. We will preprocess the data, split it into training and testing sets, define the neural network architecture, compile the model, train it and evaluate its performance.

7.1 Preprocessing and splitting the data

The dataset contains different types of data. Some columns are numerical (like “city_mpg_fuel_type_1” or “charge_time_240v”), and some are categorical (“vehicle_class” or “fuel_type”). We identify and differentiate these two types of columns, subsequently preprocessing them accordingly.

The data is split into a training set and a testing set. The training set is used to train the model, and the testing set is used to evaluate its performance. This split ensures that we can test how well the model generalizes to new, unseen data.

7.1.1 Building the neural network models and training them

7.1.1.1 Base Neural Network

We chose to use a neural network. This neural network consists of layers of neurons, where each layer applies transformations to the data. The first layer takes the input features. Then some Hidden layers help the model learn complex patterns. In the end, the output layer predicts the probability of each car manufacturer. The first layer, the input layer, takes the preprocessed input features. The second layer is set to 128 neurons, the third to 64 neurons and the last layer, the output layer, has as many neurons as there are car manufacturers (66 in our case). The activation function used in the hidden layers is the Rectified Linear Unit (ReLU), and the output layer uses the Softmax activation function. The model is compiled with the Adam optimizer and the categorical cross-entropy loss function.

```
        # Base neural network model
        model_base = Sequential([
            Input(shape = (X_train.shape[1],)),
            Dense(128, activation = 'relu'),
            Dense(64, activation = 'relu'),
            Dense(y_train.shape[1], activation = 'softmax')])
```

We used activation functions in the hidden layers to introduce non-linearity into the model. The ReLU activation function is used in the hidden layers because it is computationally efficient and helps the model learn complex patterns in the data. The Softmax activation function is used in the output layer because it converts the model’s raw output into probabilities that sum to one. This allows us to interpret the model’s output as the probability of each car manufacturer.

We used the following hyperparameters for the base model (non-exhaustive list):

- **epochs:** 150 (Corresponds to the number of times the model sees the entire dataset during training.)
- **batch_size:** 32 (Corresponds to the number of samples that the model processes before updating the weights.)
- **validation_split:** 0.2 (Corresponds to the fraction of the training data to be used as validation data.)

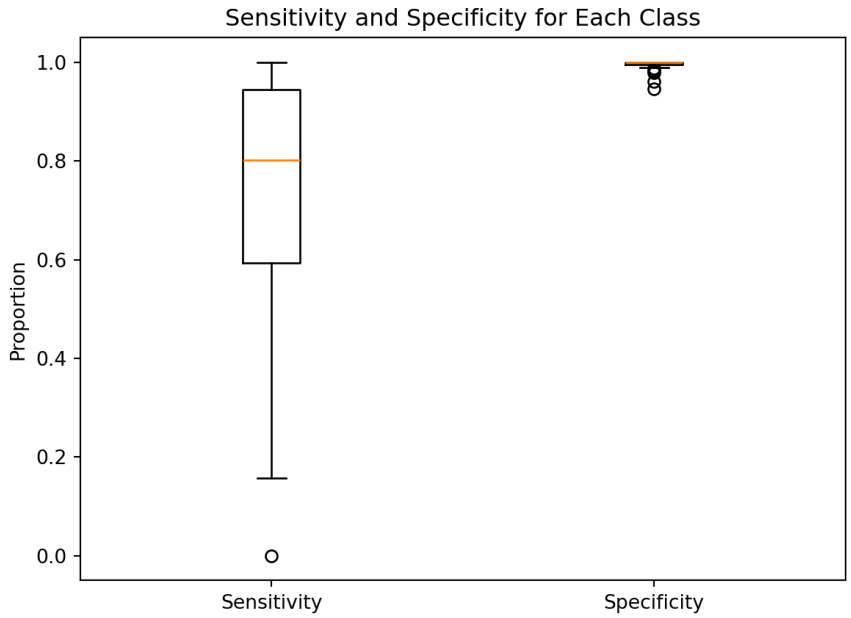
The model is trained for 5 epochs with a batch size of 32. The validation split is set to 0.2, which means that 20% of the training data is kept to be used as a validation set.

Overall, the model performs well but we haven’t dealt with the issue of unbalanced classes yet. Let’s have a look at the distribution of the sensitivity and specificity for each class.

7.1.1.1.1 Issue of unbalanced classes

The issue of unbalanced classes, as explained previously, can highly weaken the model ability to generalize to new data. The model will automatically prefer to predict the most frequent classes. We can see in the boxplots below the distribution of the sensitivity and specificity for the classes. Even though, we already dealt in part with the unbalanced class during the cleaning process, as seen in the plot in section **? @sec-make_n_plot**, there are still big differences between the classes.

```
{'whiskers': [<matplotlib.lines.Line2D object at 0x2c61f4110>, <matplotlib.lines.Line2D object at 0x2c61f7530>,
<matplotlib.lines.Line2D object at 0x2c61f7bf0>, <matplotlib.lines.Line2D object at 0x2c61f75f0>], 'caps':
[<matplotlib.lines.Line2D object at 0x2c61f4260>, <matplotlib.lines.Line2D object at 0x2c61f4e00>,
<matplotlib.lines.Line2D object at 0x2c61f7140>, <matplotlib.lines.Line2D object at 0x2c993a8d0>], 'boxes':
[<matplotlib.lines.Line2D object at 0x2d04def30>, <matplotlib.lines.Line2D object at 0x2c2fc89b0>], 'medians':
[<matplotlib.lines.Line2D object at 0x2c993b830>, <matplotlib.lines.Line2D object at 0x2c9938530>], 'fliers':
[<matplotlib.lines.Line2D object at 0x2c9939dc0>, <matplotlib.lines.Line2D object at 0x2c99387a0>], 'means': []}
```



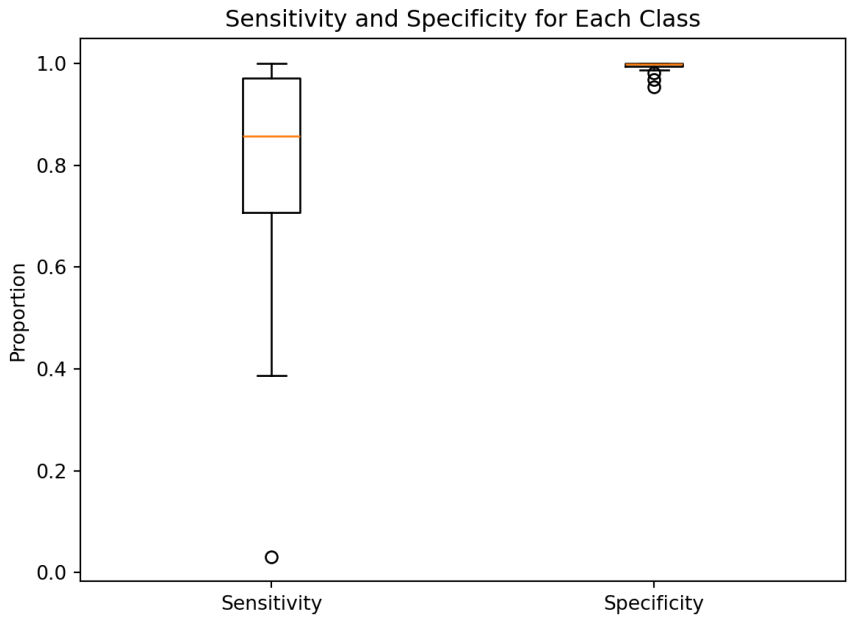
By examining the boxplot representing the distribution of sensitivity and specificity across the classes, we observe clear evidence of class imbalance. Sensitivity and specificity are not consistent across the classes. Specificity, which measures how well the model identifies true negatives, is very high for every class. This indicates that the model is effective at detecting instances that do not belong to a given class. However, sensitivity, which measures the true positive rate and reflects how well the model correctly predicts the make of a car for a specific brand, is not as high. This suggests that the model is not performing well for all classes.

For some classes with more vehicle models, the model tends to predict those classes more frequently, leading to higher accuracy but lower sensitivity for rarer classes. To address this issue, we will use class weights to ensure the model performs more evenly across all classes.

7.1.1.1.2 Adding class weights to the model

This technique, detailed in the methods section, essentially penalizes the model more for misclassifying the minority class than the majority class. By doing so, the model is encouraged to learn the patterns of the minority class more effectively, thereby enhancing its performance on the test set.

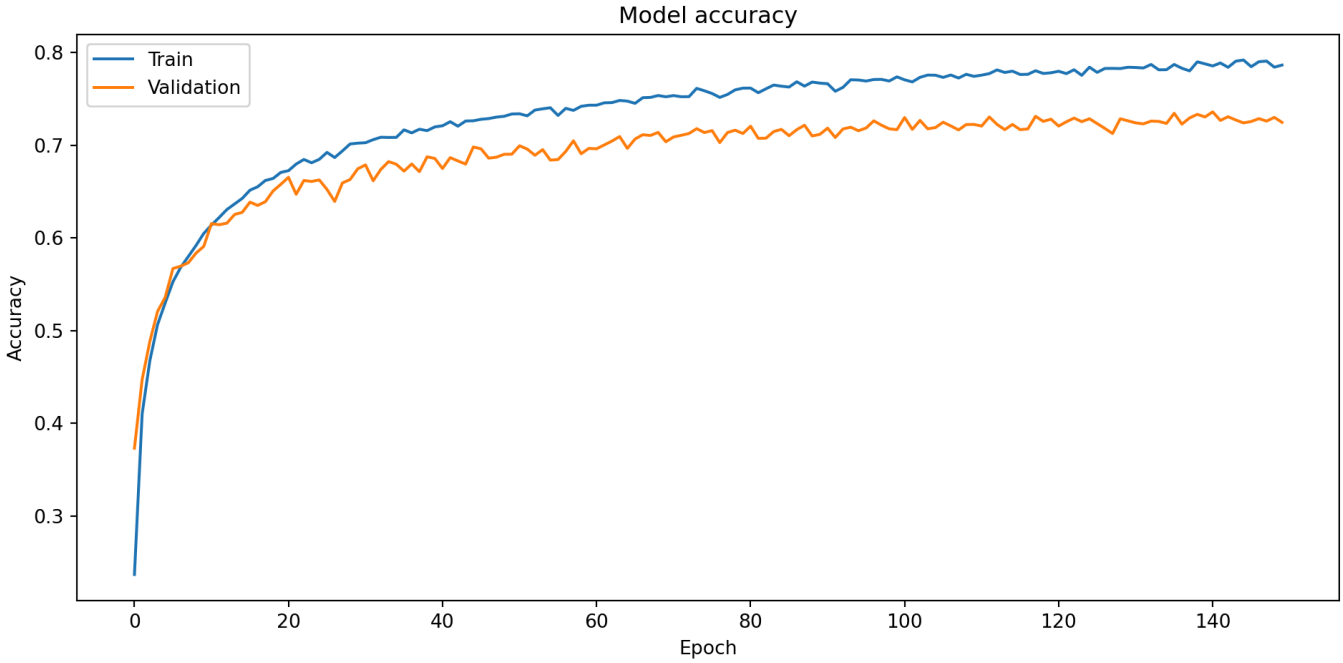
```
{'whiskers': [<matplotlib.lines.Line2D object at 0x2c65c0b30>, <matplotlib.lines.Line2D object at 0x2c65c1a60>,
<matplotlib.lines.Line2D object at 0x2c65c2120>, <matplotlib.lines.Line2D object at 0x2c7901bb0>], 'caps':
[<matplotlib.lines.Line2D object at 0x2c65c03b0>, <matplotlib.lines.Line2D object at 0x2c65c0890>,
<matplotlib.lines.Line2D object at 0x2c7900bc0>, <matplotlib.lines.Line2D object at 0x2c7900dd0>], 'boxes':
[<matplotlib.lines.Line2D object at 0x2c65c0fe0>, <matplotlib.lines.Line2D object at 0x2c65c1790>], 'medians':
[<matplotlib.lines.Line2D object at 0x2c65c0a40>, <matplotlib.lines.Line2D object at 0x2c7903cb0>], 'fliers':
[<matplotlib.lines.Line2D object at 0x2c65c13d0>, <matplotlib.lines.Line2D object at 0x2c65c21e0>], 'means': []}
```

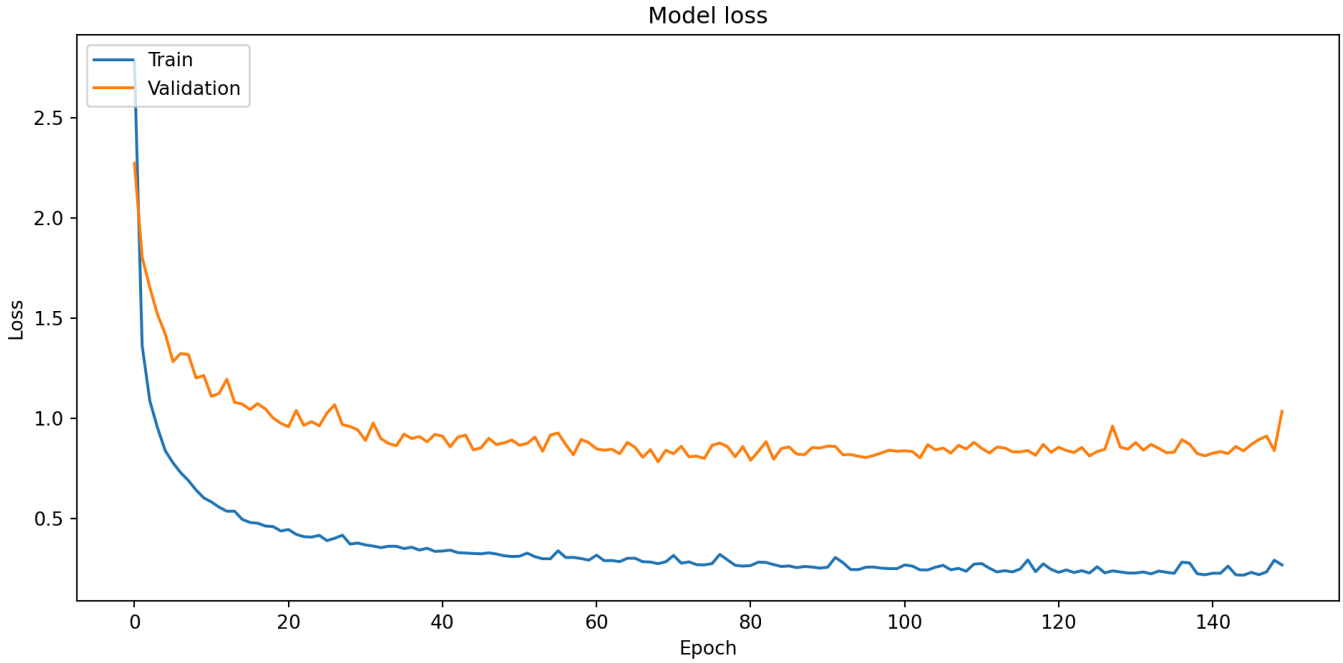


As we can see the model is taking better care of the minority classes and overall the sensitivity is higher across the classes. The sensitivity and specificity are more consistent across the classes. The model is better at generalizing to new data. In our case, this method does not eliminate completely the issue of unbalanced classes. Given the structure of our data and the discrepancy of our classes, we will use this technique for the following neural networks and move on.

7.1.1.1.3 Model performance

We can now look at the evolution of the accuracy of our model during the training process with the following plot.





As we can see, at each epoch, the accuracy is increasing and the loss is decreasing. The model is learning from the training data and improving its predictions.

But, in the end, we have a case of overfitting. The model performs well on the training data but not as well on the testing data. This is an issue because it limits the possibility of generalizing the model to new data.

Performance of the model with weighted class:

```
Final Training Accuracy: 78.64%
```

```
Final Validation Accuracy: 72.47%
```

```
Test Set Accuracy: 71.38%
```

Overall, the performance of the model is still good. However the quality can be improved. To address the issue of overfitting, we will introduce Dropout layers in the neural network.

7.1.1.2 Neural Network with Dropout layers

Dropout layers randomly set a fraction of neurons to zero during training, which helps prevent overfitting by forcing the model to learn more robust features. We will tune the dropout rates to find the optimal value that balances training and validation accuracy and that insure to reduce overfitting.

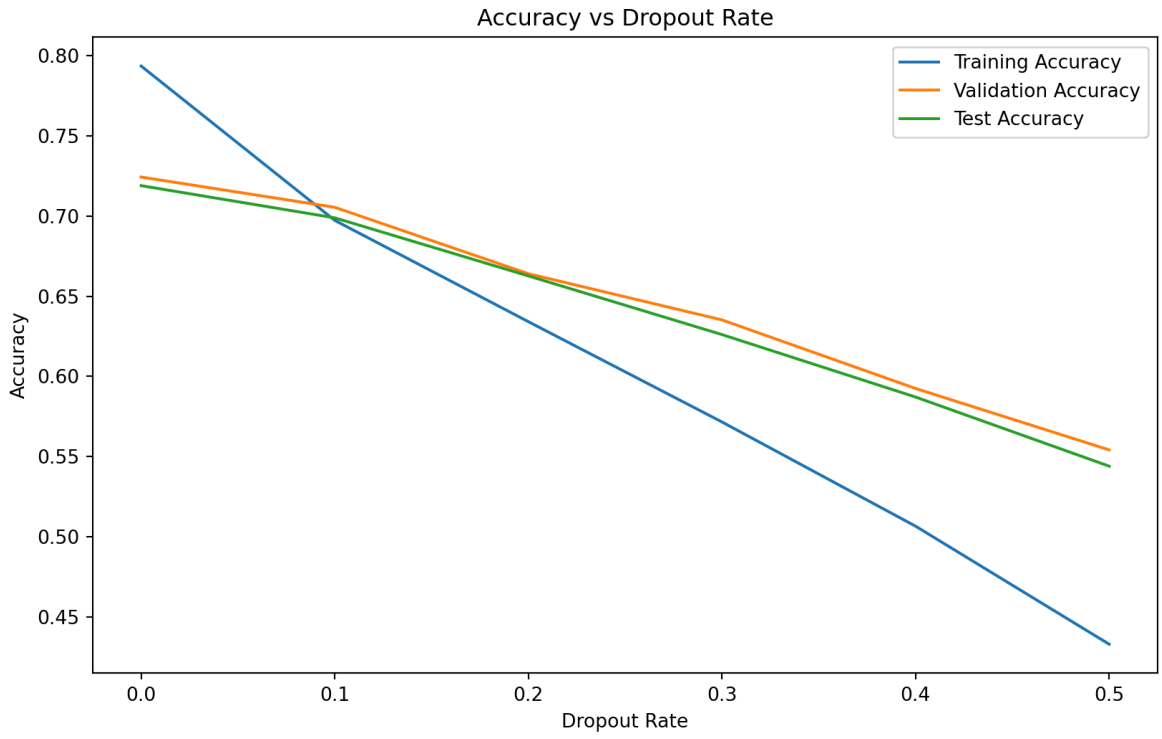
We used the following hyperparameters (non-exhaustive list):

- **epochs:** 150 (Corresponds to the number of times the model sees the entire dataset during training.)
- **batch_size:** 32 (Corresponds to the number of samples that the model processes before updating the weights.)
- **validation_split:** 0.2 (Corresponds to the fraction of the training data to be used as validation data.)
- **dropout_rate:** varies (Corresponds to the fraction of neurons to drop during training.)

We will try 5 different dropout rates in addition of the case of no dropout. We will train the model with each dropout rate and evaluate its performance on the validation and test sets. We will then plot the training, validation, and test accuracies for each dropout rate to find the optimal value.

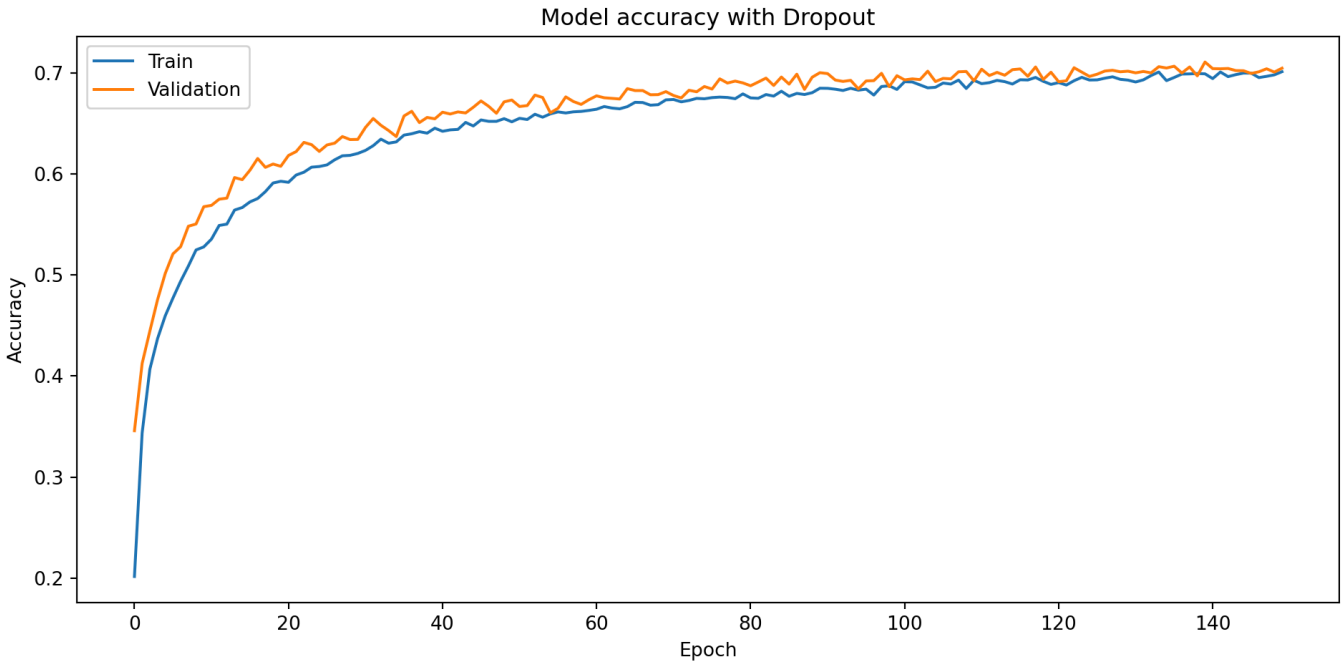
```
# Model with Dropout layers
def create_model(dropout_rate = 0.0):
    model = Sequential([
        Input(shape = (X_train.shape[1],)),
        Dense(128, activation = 'relu'),
        Dropout(dropout_rate),
        Dense(64, activation = 'relu'),
        Dropout(dropout_rate),
        Dense(y_train.shape[1], activation = 'softmax')
    ])
    model.compile(optimizer = Adam(), loss = 'categorical_crossentropy', metrics = ['accuracy'])
    return model
```

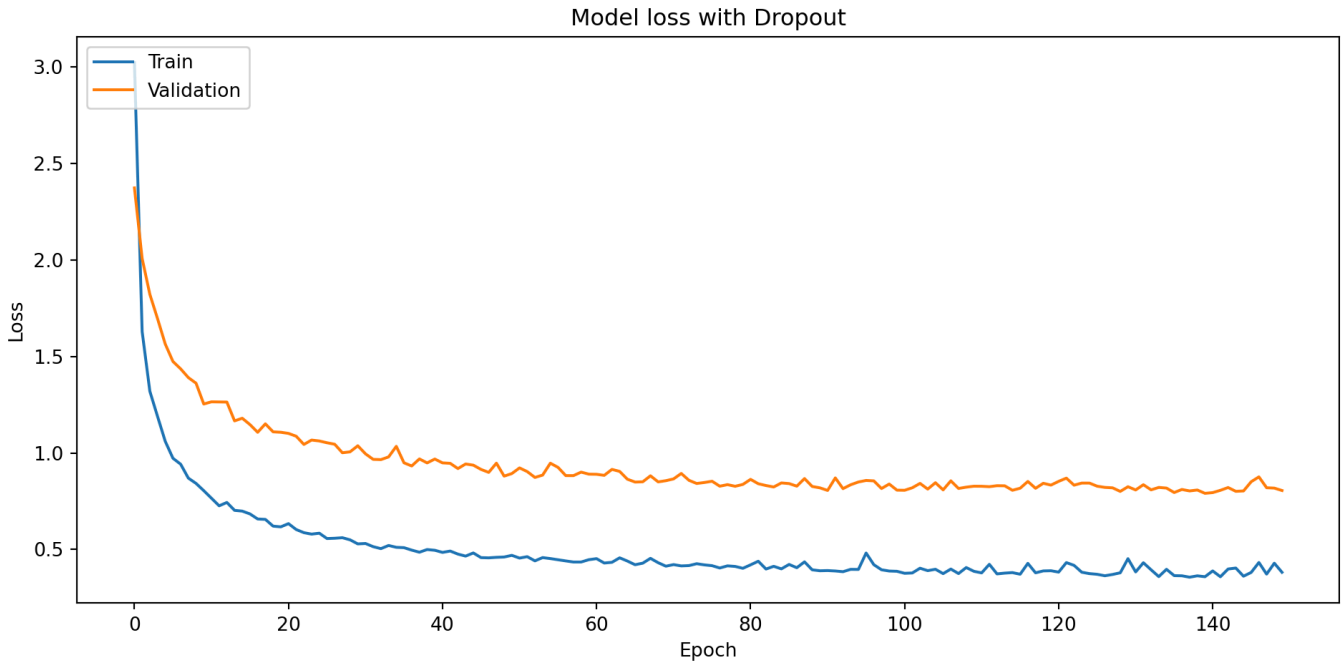
7.1.1.2.1 Selection of the best dropout rate



We can see that the model with a dropout rate of 0.1 has the best balance between reducing drastically the overfitting problem and keeping a good overall accuracy. This model has a good balance between training and validation accuracy, and it generalizes well to new data. It also eliminate the overfitting issue. We will use this dropout rate of 0.1 to train the final model that utilize class weights and dropout layers.

7.1.1.2.2 Model performance





We see that the model with dropout layers performs better that the one without it. We reached a better accuracy on the validation set and the model is clearly not overfitting as much. It is interesting to note that the validation accuracy is higher than the training accuracy. This is a good sign that the model is generalizing well to new data. It is also interesting to note that, as predicted, we see that the validation accuracy is higher than the training accuracy. This is due to the way dropout layers work. The model does not need early stopping in our case (150 epochs) since the accuracies are not decreasing and the loss is not increasing.

Performance of the model with weighted class and dropout layers:

```
Final Training Accuracy: 70.10%
```

```
Final Validation Accuracy: 70.45%
```

```
Test Set Accuracy: 69.94%
```

The final accuracy of our model is not as great as we had with our first model but the model that we are using is at least better at representing the data and generalizing to new data. We also computed the Cohen's Kappa score which is a good indicator of the model's performance. And as we can see, the model performs well.

```
Cohen's Kappa Score: 69.36%
```

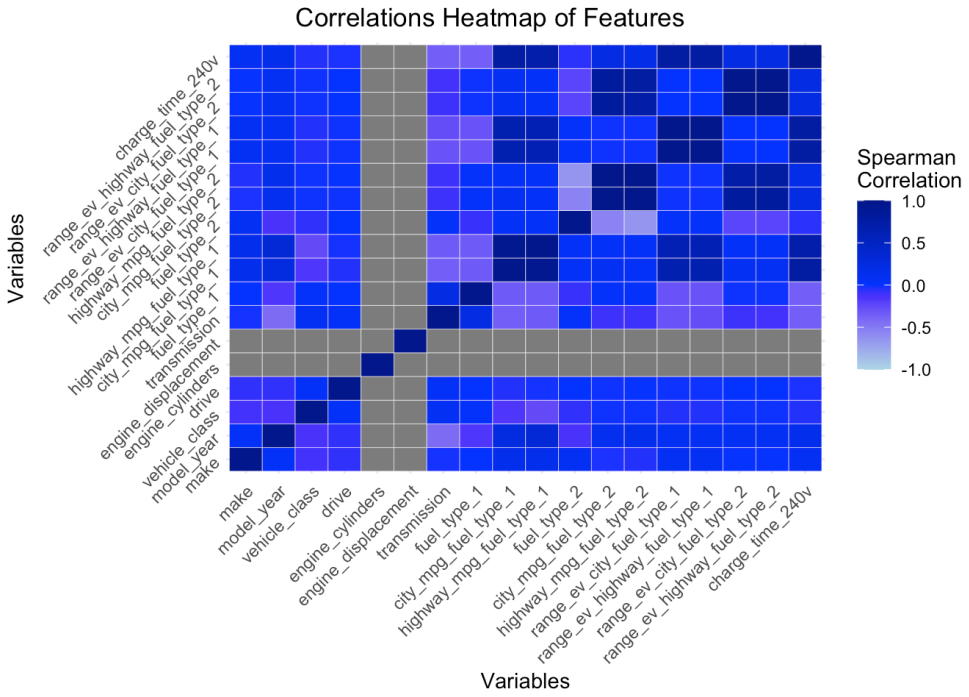
8 Principal Component Analysis

In order to see the link between the features, we can use a dimension reduction technique such as the Principal Component Analysis, aiming to link the features according to their similarities across instances and combine features in fewer dimensions.

We begin by standardizing our data, meaning transforming our character variables into factors and then the factors into numeric.

8.1 Heatmap

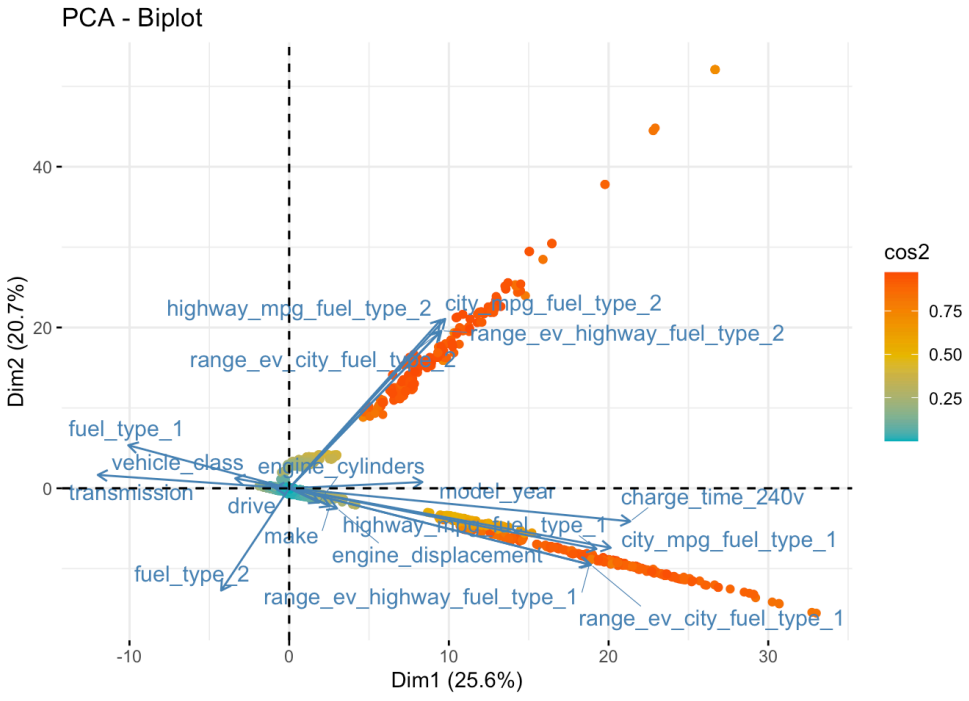
As it is interesting for us to see the relationship between the variables, we decided to start with a heatmap to get a small overview of the correlations between them.



As we can observe, some variables are strongly correlated, but most of them don't appear to be too strongly correlated (with a value lower than 0.5), whether positively or negatively. Let's now look into the link between the features using a biplot, which combines the observations as well as the features.

8.2 Biplot

Using the `PCA` function, we take the standardized data and check for the result. We provided the summary in the Annex 12.5, containing the Eigenvalues and the percentages of cumulative variance per dimension. We then use the `fviz_pca_biplot` function to visualize it.

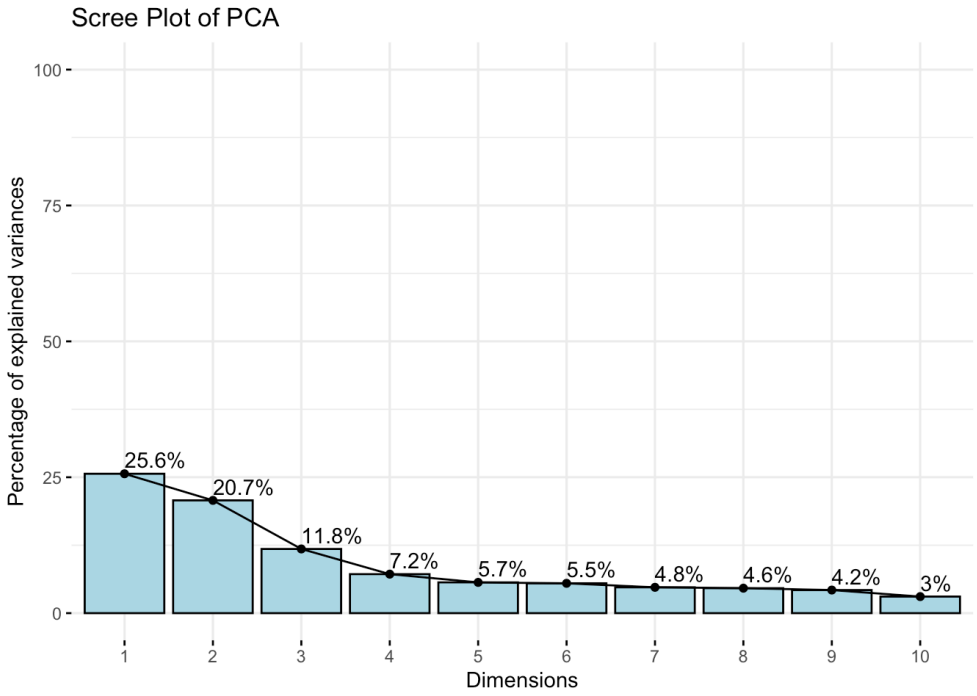


The biplot shows several informations. First, the two dimensions explain almost 50% of the total variance of the data. Then, each point represents an observation and its color represent the quality of the representation of the variables. Looking at the cos2 gradient, the redder the dot, the better the quality of representation. Then, the arrows (or vectors) represent the features. The vectors pointing in similar directions represent some positive correlations, whereas the ones going opposite represent negative correlations. Orthogonal variables are not correlated.

Taking all of these into account, we can interpret this graph in the following way: we notice that the variables linking the mpg and the range for a fuel type (e.g. fuel type 1) go in the same direction and all seem to be positively correlated, and they are uncorrelated to the same characteristics of the other fuel type (e.g. fuel type 2). Also, the mpg and range seem to be negatively correlated to their own fuel type. Moreover, fuel type 1 is uncorrelated to fuel type 2. Thus, we can infer that Dimension 1 represents the “non-hybrid” characteristics of vehicles whereas dimension 2 is more difficult to interpret with this graph. It could potentially be linked to “hybrid” characteristics of vehicles, but it is not clear.

8.3 Scree plot

The scree plot allows to see the number of dimensions needed to reach most of the variance. To visualize it, we will use the `fviz_eig` function.



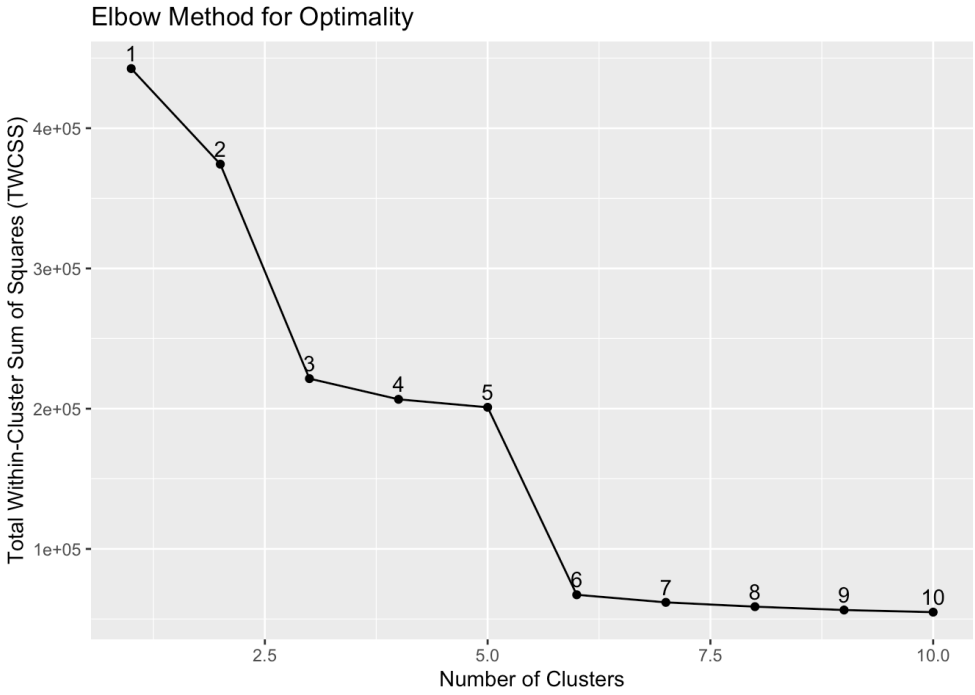
The results of the scree plot show that 7 dimensions are required to reach at least 80% of the variance, meaning the features might be relatively independent. It is already shown in the biplot above, as most arrows in the middle appear to be shorter and the cos2 are low, meaning that the features might be more linked to other dimensions than the first 2 dimensions.

9 Clustering

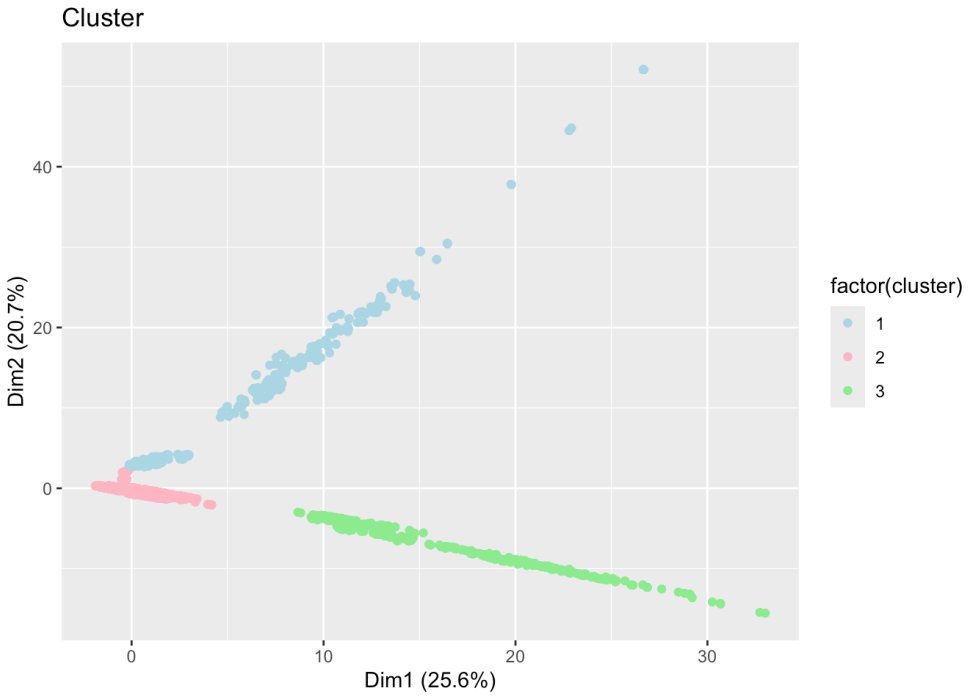
9.1 Clusters Selection

In order to continue with the clustering section, we decided to obtain the coordinates from the PCA.

Then, we proceed with the calculation of the Total Within-Cluster Sum of Squares (TWCSS) and vizualize the results using the Elbow method in order to select the optimal number of clusters.



In our elbow method, we seem to obtain two elbows, indicating that optimality is reached at three clusters, but making it to six clusters makes it even better. However, clustering to only four or five is not relevant. We will begin by working with three clusters and check if increasing it to six would be relevant. We will follow by performing a K-Means method.

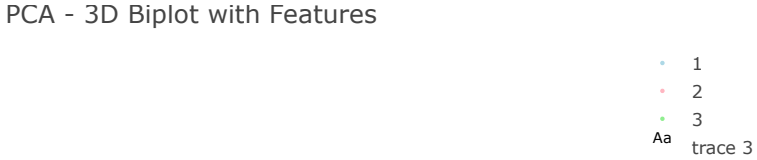


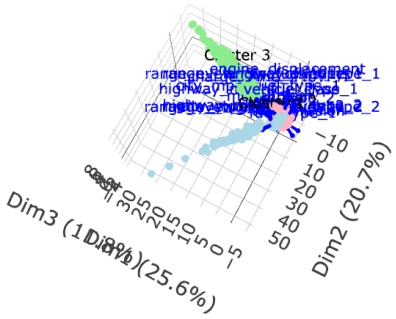
Above is the separation into three clusters. We will provide a 3D biplot for a better visualization, as Dimension 2 doesn't give us much insight.

9.2 3D Biplot

As we notice in the Principal Components Analysis, we require at least 7 dimensions to explain most of our percentages of variance, so adding a third dimension in our visualization would be of benefit. To begin with, we need to provide the cluster centers and get the loadings for the variables. We will provide a `plotly` biplot in order to be able to move around the three dimensions.

▶ Show the code



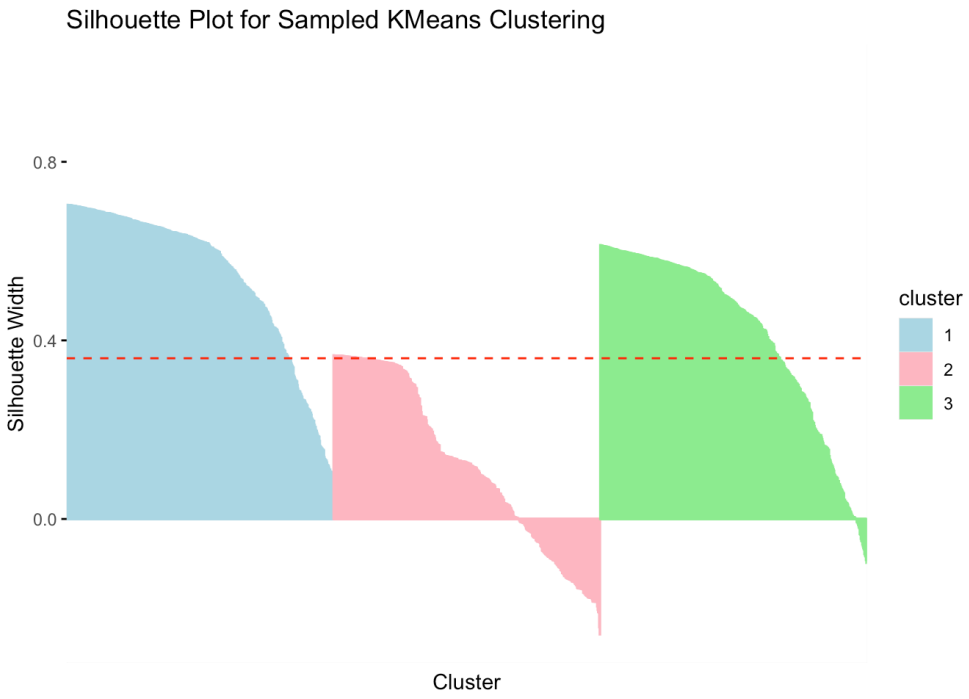


Looking at this 3D biplot, we can clearly see the 3 different clusters. Cluster 1 seems to englobe the observations from characteristics of the vehicle with fuel type 2, meaning the hybrid cars. Then, cluster 2 links observations from the different car characteristics such as the vehicle class, the engine displacement and cylinders, the drive, the make, the fuel types and the model year. Finally, Cluster 3 takes in the characteristics of vehicles with fuel type 1, meaning normal gas cars. Moreover, we can observe that Dimension 3 can be linked to the vehicle characteristics, which allows us to better understand the observations repartition. Finally, we will analyze the goodness-of-fit of the variables within their own clusters.

9.3 Silhouette Plot

The silhouette plot will provide insights on the homogeneity within the clusters. We will use the `fviz_silhouette` function for this.

```
  cluster size ave.sil.width
1       1  517          0.54
2       2  517          0.12
3       3  517          0.41
```



In order to provide the silhouette plot, we considered taking only a sample of the dataset, as it would have been to heavy to run otherwise. Here, we can notice that Clusters 1 and 3 both seem to have a high silhouette width, indicating that they are well-clustered. The widths are also mainly above the average width (the red line), meaning that there is a good consistency within the cluster. On the other hand, there is more variation in the 2nd cluster, indicating some heterogeneity within the cluster. The cluster contains some negative silhouette widths and a lot of the observations are below the red line, all of these meaning that some observations would potentially fit better in another cluster.

In the Annex 12.6, we have also provided the same clustering method but for six clusters. We observe that Cluster 2 can be divided into four clusters, but it is more difficult to analyze. As this is the unsupervised learning part, the interest is to discover patterns in the data, so for simplification purposes, we decided to stay with three clusters.

To sum the unsupervised learning part, we can clearly say that there seems to be some factors that are linked together and some observations that can be linked into three clusters. These clusters being “Hybrid Vehicles”, “Vehicle Characteristics” and “Non-Hybrid Vehicles” makes sense, as we either have a 1 propulsion type of car or hybrid cars, and the vehicle characteristics being another cluster can be explained in the fact both hybrid and non-hybrid vehicles can share some same vehicle characteristics (not all). As for the features, we need 7 dimensions to explain at least 80% of the total variance, meaning that not all the features are concretely linked together and the links between these are “moderately strong”.

10 Recommendations and Discussions

10.1 Summary of what was observed from the results

On one hand, we performed classification trees to make our predictions. We noticed that with our “Base” Decision Tree, our model was encountering overfitting, with a training accuracy higher than our test accuracy. We then used the 10 K-Fold CV in order to assess the performance of our model and we noticed that our accuracy score reached approximately 70%. In this case, overfitting was not fixed.

Then, we decided to prune the tree fo fight against overfitting. Our results showed that there was a tradeoff between the accuracy and the overfittingness when playing with the max_depth hyperparameter. Overall, a max_depth of 10 seems to be the best tradeoff between accuracy and overfitting.

We then have been able to show the variable importance, showing that the top three most important features figure in order : the engine displacement, the model of the year and the class of the vehicle.

Finally, we have used re-sampling in order to fight class imbalances. We have been able to have a much better model with less overfitting and a good accuracy by applying this method (around 90% of test set accuracy).

For the neural networks model to predict the make (brand) of a vehicle based on its technical characteristics, we managed to train a model with a high kappa score of almost 70%. We have been working with unbalanced classes. Our first neural network model was trained without addressing this problem. For this reason, our first model was very poor at predicting the minority class. To remedy this issue, we applied weighted classes to the loss function so that the model could give more importance to the minority class. We then had to address the presence of overfitting. The model’s validation and test accuracy were noticeably higher than the training accuracy. We applied a technique that we discovered but hadn’t seen in class called dropout layers to our model to mitigate this issue. This technique was interesting to work with. As expected, we saw that the validation accuracy was higher than the test and training accuracy. This is due to the way dropout layers are applied to the model. Some of the neurons are only disabled during the training phase and not during the testing phase. We decided to tune this hyperparameter to reduce the gap between the validation and test accuracy while ensuring not to drop our accuracy too much. In the end, we managed to obtain a model after 150 epochs that had a kappa score and an accuracy on the test set of 70%. We can say that the model is able to predict the make of a vehicle based on its technical characteristics with good accuracy.

To enhance the performance of our neural network models, we should explore novel techniques and alternative methodologies to address the various challenges encountered. In dealing with unbalanced data, techniques such as downsampling or resampling, similar to those utilized in our classification model, may prove beneficial. To address the issue of overfitting, the application of cross-validation or bootstrap methods is recommended. Furthermore, optimizing additional hyperparameters could contribute to further improvements in our model’s performance.

Then, using unsupervised learning methods had the purpose of finding patterns that are interesting to see links between the features. We have observed that seven dimensions were required to fulfill at least 80% of the variance and it has been confirmed with the screeplot. We observed while adding an extra dimension in the biplot that Dimension 1 links the “non-hybrid” characteristics of vehicles, Dimension 2 seem to take into account to some extent the “hybrid” characteristics and Dimension 3 more the technical aspects of a vehicle. In addition to that, the observations seem to somewhat follow in a way. Clustering into three shows a clear differentiation between hybrid and non-hybrid cars, but with a less interpretable cluster that seemingly groups the observations related to the vehicle characteristics.

Regarding the limitations for the unsupervised learning, it seems that it could be of benefit to push the analysis by looking at all seven dimensions and six clusters. This all means that the features are not that much correlated and could be analyzed further.

11 References

Azizi, I., & Boldi, M.-O. (2024). *MLBA - S24 - Machine Learning in Business Analytics 2023-2024*. MLBA - S24. Retrieved May 19, 2024, from https://do-unil.github.io/mlba/

Yadav, H. (2023, 31 mai). Dropout in Neural Networks - Towards Data Science. *Medium*. Retrieved May 19, 2024, from https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9

Klein, B. (2024, 19 avril). *21. Dropout Neural Networks in Python | Machine Learning. Retrieved May 19*, 2024, from https://python-course.eu/machine-learning/dropout-neural-networks-in-python.php#:~:text=The%20dropout%20approach%20means%20that,learn%20set%20with%20this%20network.

12 Annex

12.1 Data Columns Detailed

Name	data
Number of rows	45896
Number of columns	26
_____	
Column type frequency:	
character	8
numeric	18
_____	
Group variables	None

Data summary

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Make	0	1	3	34	0	141	0
Model	0	1	1	47	0	4762	0
Fuel.Type.1	0	1	6	17	0	6	0
Fuel.Type.2	0	1	0	11	44059	5	0
Drive	0	1	0	26	1186	8	0
Engine.Description	0	1	0	46	17031	590	0
Transmission	0	1	0	32	11	41	0
Vehicle.Class	0	1	4	34	0	34	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	
ID	0	1.00	23102.11	13403.10	1.00	11474.75	23090.50	34
Model.Year	0	1.00	2003.61	12.19	1984.00	1992.00	2005.00	2(
Estimated.Annual.Petrolum.Consumption..Barrels.	0	1.00	15.33	4.34	0.05	12.94	14.88	
City.MPG..Fuel.Type.1.	0	1.00	19.11	10.31	6.00	15.00	17.00	
Highway.MPG..Fuel.Type.1.	0	1.00	25.16	9.40	9.00	20.00	24.00	
Combined.MPG..Fuel.Type.1.	0	1.00	21.33	9.78	7.00	17.00	20.00	
City.MPG..Fuel.Type.2.	0	1.00	0.85	6.47	0.00	0.00	0.00	
Highway.MPG..Fuel.Type.2.	0	1.00	1.00	6.55	0.00	0.00	0.00	
Combined.MPG..Fuel.Type.2.	0	1.00	0.90	6.43	0.00	0.00	0.00	
Engine.Cylinders	487	0.99	5.71	1.77	2.00	4.00	6.00	
Engine.Displacement	485	0.99	3.28	1.36	0.00	2.20	3.00	
Time.to.Charge.EV..hours.at.120v.	0	1.00	0.00	0.00	0.00	0.00	0.00	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50
Time.to.Charge.EV..hours.at.240v.	0	1.00	0.11	1.01	0.00	0.00	0.00
Range..for.EV.	0	1.00	2.36	24.97	0.00	0.00	0.00
City.Range..for.EV...Fuel.Type.1.	0	1.00	1.62	20.89	0.00	0.00	0.00
City.Range..for.EV...Fuel.Type.2.	0	1.00	0.17	2.73	0.00	0.00	0.00
Hwy.Range..for.EV...Fuel.Type.1.	0	1.00	1.51	19.70	0.00	0.00	0.00
Hwy.Range..for.EV...Fuel.Type.2.	0	1.00	0.16	2.46	0.00	0.00	0.00

12.2 Number of models per make

Show 5 entries Search:

make	Number
Audi	1183
BMW	2299
Chevrolet	4239
Dodge	2596
Ford	3604

Showing 1 to 5 of 15 entries Previous 1 2 3 Next

12.3 Data summary

The table below provides an overview of the dataset.

Variable	N	Mean	Std. Dev.	Min	Pctl. 25	Pctl. 75	Max
ID	45896	23102	13403	1	11475	34751	46332
Model.Year	45896	2004	12	1984	1992	2015	2023
Estimated.Annual.Petrolum.Consumption..Barrels.	45896	15	4.3	0.047	13	18	43
Fuel.Type.1	45896						
... Diesel	1254	3%					
... Electricity	484	1%					
... Midgrade Gasoline	155	0%					
... Natural Gas	60	0%					
... Premium Gasoline	14138	31%					
... Regular Gasoline	29805	65%					
City.MPG..Fuel.Type.1.	45896	19	10	6	15	21	150
Highway.MPG..Fuel.Type.1.	45896	25	9.4	9	20	28	140
Combined.MPG..Fuel.Type.1.	45896	21	9.8	7	17	23	142
Fuel.Type.2	45896						
...	44059	96%					
... E85	1513	3%					
... Electricity	296	1%					
... Natural Gas	20	0%					
... Propane	8	0%					
City.MPG..Fuel.Type.2.	45896	0.85	6.5	0	0	0	145
Highway.MPG..Fuel.Type.2.	45896	1	6.6	0	0	0	121
Combined.MPG..Fuel.Type.2.	45896	0.9	6.4	0	0	0	133

Variable	N	Mean	Std. Dev.	Min	Pctl. 25	Pctl. 75	Max
Engine.Cylinders	45409	5.7	1.8	2	4	6	16
Engine.Displacement	45411	3.3	1.4	0	2.2	4.2	8.4
Time.to.Charge.EV..hours.at.120v.	45896	0	0	0	0	0	0
Time.to.Charge.EV..hours.at.240v.	45896	0.11	1	0	0	0	15
Range..for.EV.	45896	2.4	25	0	0	0	520
City.Range..for.EV...Fuel.Type.1.	45896	1.6	21	0	0	0	521
City.Range..for.EV...Fuel.Type.2.	45896	0.17	2.7	0	0	0	135
Hwy.Range..for.EV...Fuel.Type.1.	45896	1.5	20	0	0	0	520
Hwy.Range..for.EV...Fuel.Type.2.	45896	0.16	2.5	0	0	0	115

Summary Statistics

Variable	N	Mean	Std. Dev.	Min	Pctl. 25	Pctl. 75	Max
ID	45896	23102	13403	1	11475	34751	46332
Model.Year	45896	2004	12	1984	1992	2015	2023
Estimated.Annual.Petrolum.Consumption..Barrels.	45896	15	4.3	0.047	13	18	43
Fuel.Type.1	45896						
... Diesel	1254	3%					
... Electricity	484	1%					
... Midgrade Gasoline	155	0%					
... Natural Gas	60	0%					
... Premium Gasoline	14138	31%					
... Regular Gasoline	29805	65%					
City.MPG..Fuel.Type.1.	45896	19	10	6	15	21	150
Highway.MPG..Fuel.Type.1.	45896	25	9.4	9	20	28	140
Combined.MPG..Fuel.Type.1.	45896	21	9.8	7	17	23	142
Fuel.Type.2	45896						
...	44059	96%					
... E85	1513	3%					
... Electricity	296	1%					
... Natural Gas	20	0%					
... Propane	8	0%					
City.MPG..Fuel.Type.2.	45896	0.85	6.5	0	0	0	145
Highway.MPG..Fuel.Type.2.	45896	1	6.6	0	0	0	121
Combined.MPG..Fuel.Type.2.	45896	0.9	6.4	0	0	0	133
Engine.Cylinders	45409	5.7	1.8	2	4	6	16
Engine.Displacement	45411	3.3	1.4	0	2.2	4.2	8.4
Time.to.Charge.EV..hours.at.120v.	45896	0	0	0	0	0	0
Time.to.Charge.EV..hours.at.240v.	45896	0.11	1	0	0	0	15
Range..for.EV.	45896	2.4	25	0	0	0	520
City.Range..for.EV...Fuel.Type.1.	45896	1.6	21	0	0	0	521
City.Range..for.EV...Fuel.Type.2.	45896	0.17	2.7	0	0	0	135
Hwy.Range..for.EV...Fuel.Type.1.	45896	1.5	20	0	0	0	520
Hwy.Range..for.EV...Fuel.Type.2.	45896	0.16	2.5	0	0	0	115

Summary Statistics

12.4 Data cleaned overview

Cleaned Dataset

Show 5 entries

Search:

make	model_year	vehicle_class	drive	engine_cylinders	engine_displacement	transmission	fuel_ty
Acura	1986	Subcompact Cars	Front-Wheel Drive	4	1.6	Automatic 4-spd	Regular Gasolin
Acura	1986	Subcompact Cars	Front-Wheel Drive	4	1.6	Manual 5-spd	Regular Gasolin
Acura	1986	Compact Cars	Front-Wheel Drive	6	2.5	Automatic 4-spd	Regular Gasolin
Acura	1986	Compact Cars	Front-Wheel Drive	6	2.5	Manual 5-spd	Regular Gasolin
Acura	1987	Subcompact Cars	Front-Wheel Drive	4	1.6	Automatic 4-spd	Regular Gasolin

Showing 1 to 5 of 42,240 entries Previous 1 2 3 4 5 ... 8,448 Next

Name	Number_of_rows	Number_of_columns	Character	Numeric	Group_variables
data_cleaned	42240	18	8	5	None

Cleaned and Reduced Dataset

Show 5 entries Search:

make	model_year	vehicle_class	drive	engine_cylinders	engine_displacement	transmission	fuel_ty
Acura	1986	Subcompact Cars	Front-Wheel Drive	4	1.6	Automatic 4-spd	Regular Gasolin
Acura	1986	Subcompact Cars	Front-Wheel Drive	4	1.6	Manual 5-spd	Regular Gasolin
Acura	1986	Compact Cars	Front-Wheel Drive	6	2.5	Automatic 4-spd	Regular Gasolin
Acura	1986	Compact Cars	Front-Wheel Drive	6	2.5	Manual 5-spd	Regular Gasolin
Acura	1987	Subcompact Cars	Front-Wheel Drive	4	1.6	Automatic 4-spd	Regular Gasolin

Showing 1 to 5 of 42,061 entries Previous 1 2 3 4 5 ... 8,413 Next

Name	Number_of_rows	Number_of_columns	Character	Numeric	Group_variables
data_cleaned_reduced	42061	18	8	5	None

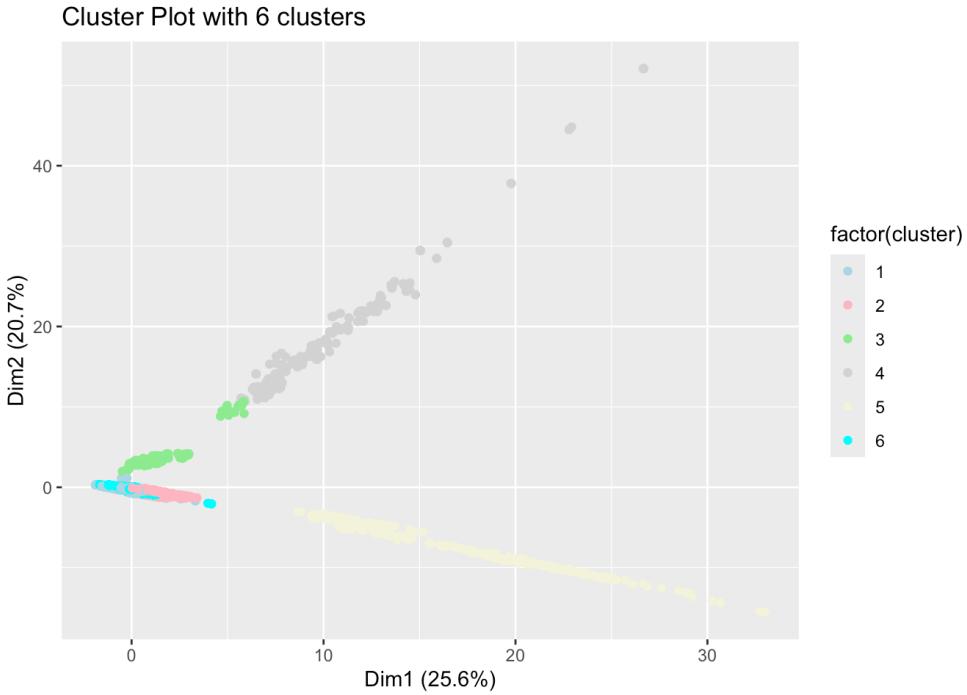
12.5 Eigenvalues for the Principal Components Analysis

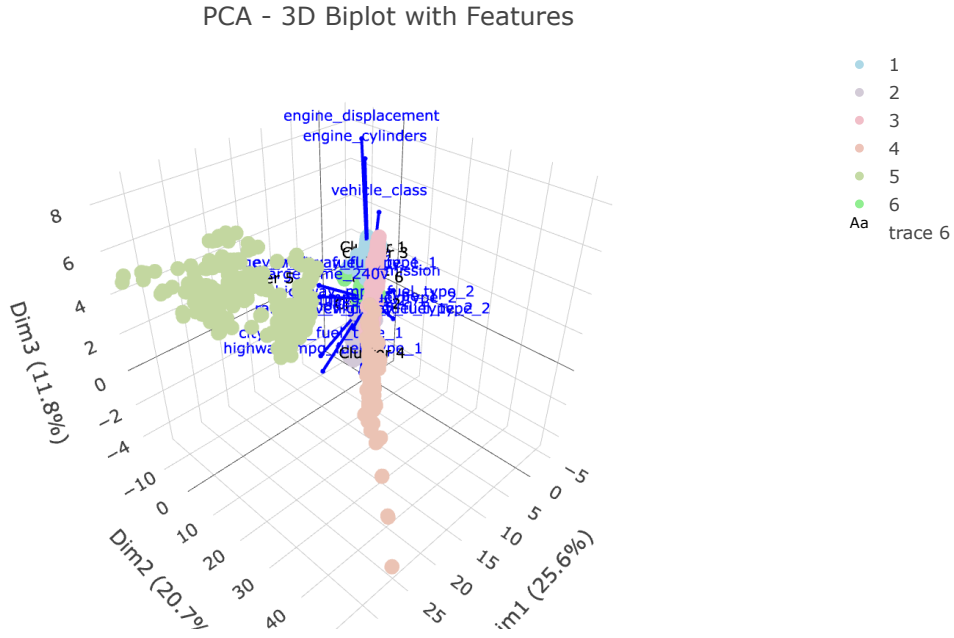
Looking at the eigenvalues, we notice that we have in total 18 dimensions, for which 7 are required to reach at least 80% of the cumulative percentage of variance. We can also see the first ten observations as well as variables, where information such as the distance, cos2 or dimension are shown.

```
Call:
PCA(X = data_prepared, graph = FALSE)


Eigenvalues
                       Dim.1   Dim.2   Dim.3   Dim.4   Dim.5   Dim.6   Dim.7
Variance               4.616   3.735   2.126   1.292   1.019   0.988   0.856
% of var.             25.644  20.748  11.809   7.177   5.660   5.490   4.753
Cumulative % of var.  25.644  46.392  58.201  65.378  71.038  76.527  81.280
                       Dim.8   Dim.9  Dim.10  Dim.11  Dim.12  Dim.13  Dim.14
Variance               0.827   0.765   0.549   0.510   0.349   0.193   0.137
% of var.              4.594   4.250   3.047   2.834   1.941   1.071   0.759
Cumulative % of var.  85.875  90.124  93.172  96.005  97.946  99.017  99.777
                      Dim.15  Dim.16  Dim.17  Dim.18
Variance               0.027   0.008   0.003   0.002
% of var.              0.150   0.045   0.018   0.011
Cumulative % of var.  99.926  99.971  99.989 100.000

Individuals (the 10 first)
                              Dist    Dim.1    ctr   cos2    Dim.2    ctr
1                          |  3.334 | -1.087  0.001  0.106 |  0.065  0.000
2                          |  3.387 | -0.907  0.000  0.072 | -0.079  0.000
3                          |  3.522 | -1.153  0.001  0.107 | -0.031  0.000
4                          |  2.761 | -1.068  0.001  0.150 |  0.012  0.000
5                          |  2.713 | -0.991  0.001  0.133 | -0.022  0.000
6                          |  2.713 | -0.991  0.001  0.133 | -0.022  0.000
7                          |  2.847 | -1.039  0.001  0.133 | -0.066  0.000
8                          |  2.876 | -1.151  0.001  0.160 | -0.019  0.000
9                          |  4.850 | -1.810  0.002  0.139 |  0.346  0.000
10                         |  3.313 | -1.686  0.001  0.259 |  0.231  0.000
                            cos2    Dim.3    ctr   cos2
1                          0.000 |  0.018  0.000  0.000 |
2                          0.001 |  2.589  0.007  0.584 |
3                          0.000 |  2.603  0.008  0.546 |
4                          0.000 |  0.658  0.000  0.057 |
5                          0.000 |  0.609  0.000  0.050 |
6                          0.000 |  0.609  0.000  0.050 |
7                          0.001 |  0.490  0.000  0.030 |
8                          0.000 |  0.546  0.000  0.036 |
9                          0.005 | -0.005  0.000  0.000 |
10                         0.005 |  1.551  0.003  0.219 |

Variables (the 10 first)
                             Dim.1    ctr   cos2    Dim.2    ctr   cos2
make                       |  0.106  0.242  0.011 | -0.092  0.229  0.009 |
model_year                 |  0.347  2.605  0.120 |  0.032  0.028  0.001 |
vehicle_class              | -0.141  0.428  0.020 |  0.051  0.071  0.003 |
drive                      | -0.038  0.031  0.001 |  0.003  0.000  0.000 |
engine_cylinders           |  0.077  0.130  0.006 | -0.073  0.141  0.005 |
engine_displacement        |  0.125  0.338  0.016 | -0.104  0.292  0.011 |
transmission               | -0.498  5.376  0.248 |  0.069  0.128  0.005 |
fuel_type_1                | -0.419  3.802  0.175 |  0.223  1.328  0.050 |
city_mpg_fuel_type_1       |  0.837 15.173  0.700 | -0.308  2.532  0.095 |
highway_mpg_fuel_type_1    |  0.800 13.860  0.640 | -0.313  2.630  0.098 |
                            Dim.3    ctr   cos2
make                       -0.472 10.475  0.223 |
model_year                 -0.120  0.679  0.014 |
vehicle_class               0.474 10.568  0.225 |
drive                       0.158  1.172  0.025 |
engine_cylinders            0.755 26.797  0.570 |
engine_displacement         0.851 34.040  0.724 |
transmission               -0.018  0.016  0.000 |
fuel_type_1                -0.170  1.358  0.029 |
city_mpg_fuel_type_1       -0.242  2.753  0.059 |
highway_mpg_fuel_type_1    -0.351  5.790  0.123 |
```

12.6 3D Biplot for 6 clusters

After looking at the silhouette plot in the unsupervised learning part, we decided to provide a 3D biplot for six clusters, as we can also see in the elbow plot that six seem to be optimal in a way.





In this biplot, we can observe that it is possible to divide into six clusters. When comparing it to the 3D biplot in the `results_unsupervised_learning` section, we clearly notice that Cluster 2 could be divided into four smaller clusters, which indicates heterogeneity in this cluster when using only three clusters. However, with six clusters in hand, it is more difficult to interpret the four distinct clusters. In addition to that, it explains the second elbow in the elbow method: at three clusters, we obtained optimality, but we get another steep curve between Clusters 5 and 6, meaning that selecting four or five clusters would not be too much of a benefit, but adding a sixth cluster could be worth capturing. In our case, if adding a fourth of fifth cluster is not significant, it makes it harder to interpret these two specific clusters. Stopping at three clusters still is significant for us and it makes our clustering analysis more interpretable than by using six, that's why we selected only three clusters for our analysis.