Predicting CO^2 Emissions by Vehicles

Final report by:

Maurice Wendel, Oussama Fatmi, Jens Tauscher

January 2024

DataScientest

Table of Contents

Introduction

Nowadays, the automotive industry finds itself at a crucial juncture, drawing increased attention due to its substantial impact on global carbon dioxide (CO2) emissions. With the escalating concerns about climate change and environmental sustainability, there's an urgent need to explore innovative approaches that can effectively reduce the ecological footprint of vehicles. This project is dedicated to a comprehensive analysis and prediction of CO2 emissions from cars, employing advanced data science techniques such as Machine Learning and Deep Learning.

The main goals of this project are all about using data science techniques to categorize vehicles into different energy classes and employ regression models to predict and estimate CO^2 emissions. Our dataset EEA DataHub sourced from the European Environment Agency (EEA), provides a comprehensive array of information encompassing various features related to vehicle emissions. Key parameters within the dataset include country details, manufacturer specifications, vehicle characteristics, fuel types, and metrics associated with emissions. This project adopts a multifaceted approach, integrating classification techniques to categorize vehicles into different energy classes and implementing linear regression for precise predictions of CO^2 emissions. By leveraging state-of-the-art technologies in ML and DL, our aim is to construct models that contribute to a nuanced understanding of the factors influencing CO^2 emissions from cars. As we delve into the extensive dataset provided by the EEA, we intend to uncover patterns, correlations, and latent insights that can inform the development of effective strategies for reducing the carbon footprint of the automotive sector. This project aligns with the global commitment to sustainable development, addressing the pressing need for environmentally friendly transportation solutions.

The dataset, including features from vehicle specifications to emission reduction technologies, presents a unique opportunity to create models with high predictive accuracy. The outcomes of such projects have the potential to influence policy decisions, guide manufacturers towards more sustainable practices, and empower consumers to make environmentally conscious choices in the realm of transportation.

Dataset

General overview of dataset

The used dataset is a comprehensive compilation of information related to carbon dioxide (CO^2) emissions from vehicles. This extensive dataset encompasses a diverse set of features, offering valuable insights into various aspects of vehicle characteristics and emissions-related parameters. The dataset is structured with the following key features:

Column Name	Meaning
Country	Country
VFN	Vehicle family identification number
Mp	Pool
Mh	Manufacturer name (EU standard)
Man	Manufacturer name (OEM declaration)
MMS	Manufacturer name (MS registry denomination)
TAN	Type approval number
T	Type
Va	Variant
Ve	Version
Mk	Make
Cn	Commercial name
Ct	Category of the vehicle type approved
Cr	Category of the vehicle registered
r	Total new registrations
m (kg)	Mass in running order (kg)
Mt	WLTP test mass
Enedc (g/km)	Specific CO^2 emissions in g/km (NEDC)
Ewltp (g/km)	Specific CO^2 emissions in g/km (WLTP)
W (mm)	Wheel base in mm
At1 (mm)	Axle width steering axle in mm
At2 (mm)	Axle width other axle in mm
Ft	Fuel type
Fm	Fuel mode
ec (cm3)	Engine capacity in cm^3
ep (KW)	Engine power in KW

z (Wh/km)	Electric energy consumption in Wh/km
Electric range (km)	Electric range (km)
IT	Innovative technology or group of innovative technologies
Erwltp (g/km)	Emissions reduction through innovative technologies in g/km
Erwltp (g/km) (WLTP)	Emissions reduction through innovative technologies in g/km (WLTP)
De	Deviation factor
Vf	Verification factor
Date of registration	Date of registration
Fuel consumption	Fuel consumption
Status	Type of data
year	Registration year

Dataset Selection and Reduction

The European Environment Agency (EEA) offers datasets covering the years 2010 to 2022, providing a robust resource for our investigation into carbon dioxide (CO^2) emissions from vehicles. Faced with computational constraints and aiming for a manageable scope, we focused on the 2021 dataset, which is a recent and validated dataset, initially comprising 9920108 observations and 34 feature columns.

To enhance computational efficiency, we further refined our analysis by filtering the dataset based on a specific country. Opting for France, which presented a significant yet manageable number of records, totaling 1,777,878 observations, allows us to streamline computations while retaining a substantial sample size for a comprehensive analysis of CO^2 emissions from vehicles within the specific context of France.

Exploration

Upon initial examination of the dataset, we conducted a comprehensive analysis to identify the data types, ascertain the percentage of missing values, and count the number of unique values for each column. For coding reasons, some features were renamed to remove spaces and parentheses. Additionally, a detailed description has been included for each feature, aiming to facilitate a better understanding of their respective meanings. The descriptions were sourced from MS Guidelines 2019, providing a reliable reference for the interpretation of the dataset features. This preliminary exploration serves as a foundational step in comprehending the characteristics and structure of the dataset.

Name of the Column	Type	Missing Values (%)	Unique Value Count
ID	int64	0	1777878
Country	object	0	1
VFN	object	0	3248
Mp	object	1	10
Mh	object	0	65
Man	object	0	64
Tan	object	0	2173
T	object	0	357
Va	object	0	2196
Ve	object	0	9459
Mk	object	0	62
Cn	object	0	580
Ct	object	0	2
Cr	object	0	2
r	int64	0	1
m_kg	float64	0	1149
Mt	float64	0	1972
Enedc_g/km	float64	90	26
Ewltp (g/km)	float64	0	341
W_mm	float64	0	272
At1_mm	float64	0	245
At2_mm	float64	0	265
Ft	object	0	10
Fm	object	0	6
ec_cm3	float64	10	113
ep_KW	float64	0	218
z_Wh/km	float64	82	226

IT	object	36	65
Erwltp_g/km	float64	36	22
Status	object	0	1
year	int64	0	1
Date of registration	object	0	354
Fuel consumption	float64	21	155
Er_km	float64	82	470

Pre-Processing

As evident from the summary table, the dataset contains missing values, necessitating a thorough cleaning phase. This step is paramount in the field of Data Science, as it ensures the preparation of a refined dataset that is conducive to subsequent analytical steps. Cleaning the dataset is an essential process, addressing the presence of missing values and enhancing the overall quality of the data for further analysis.

Features Cleaning

Non-contributory features

The columns 'MMS', 'Ernedc_g/km', 'De' and 'Vf' exclusively consist of NaN (Not a Number) values. Given the absence of any meaningful data within these columns, they have been systematically removed from the dataset as there is no pertinent information to extract. This decision ensures the integrity and accuracy of the data analysis by eliminating irrelevant or non-contributory variables.

Features with Excessive Unique Values

Within the dataset, certain features exhibit an exceptionally high percentage of unique values, reaching up to 100%—indicating a unique value for each row. These columns, characterized by their high diversity, present a challenge for Machine Learning (ML) or Deep Learning (DL) models. The inclusion of such features may introduce noise and hinder model generalization. Consequently, to streamline the dataset and enhance the efficacy of our models, columns with an overwhelmingly high unique values percentage have been systematically removed, ensuring a more focused and relevant set of features for robust model training and evaluation.

Features with low missing values percentage

The 'Mp' column exhibits a minor 1% occurrence of missing values. Due to the difficulty in accurately estimating the Manufacturer pool for the unobserved entries, and considering the relatively low percentage of missing values, the decision has been made to eliminate the corresponding observations. This approach ensures the integrity of the dataset by mitigating the potential inaccuracies that might arise from attempting to impute Manufacturer pool data in a manner that is challenging due to the nature of the information. The removal of these observations maintains the overall quality and reliability of the dataset for subsequent analyses.

Observations Cleaning

Exclusion of non-polluting cars

Cars classified as environmentally friendly, such as *'ELECTRIC'* and *'HYDROGEN'* do not align with the specific focus of our study, which aims to predict crucial features associated with CO^2 pollution rates. Including these non-polluting vehicles could introduce confounding factors, potentially compromising the accuracy of our analysis. Therefore, it was decided to exclude these cars, alongside vehicles with an *'UNKNOWN'* fuel type, from the dataset. This intentional removal is designed to enhance the clustering effectiveness of our models, allowing them to discern patterns and relationships among vehicles directly influencing pollution metrics. By eliminating these outliers, we aim to refine the models precision and ensure a more accurate representation of factors contributing to pollution.

Rare Categories in Fuel Type Distribution

Within the dataset, certain fuel type categories show negligible representation. Specifically, the *'NG'* and *'NG-BIOMETHANE'* categories have a representation percentage below 0.1%. Given the significance of fuel type as a crucial feature in our study, it is imperative to analyze the categorical distribution within the fuel type column. Recognizing the limited impact and potential skewing effect of these sparsely represented categories, a decision has been made to exclude *'NG'* and *'NG-BIOMETHANE'* from the dataset. This intentional filtration ensures a more focused and meaningful exploration of the fuel type feature, contributing to the accuracy and reliability of subsequent analyses.

Filling Missing Values

Electric Range and Electric Energy Consumption

Noteworthy considerations arise when addressing missing values within the Electric Range 'Er' and Electric Energy Consumption 'z_Wh/km' columns. The absence of values in these columns signifies cars with no electric support, primarily applicable to non-hybrid vehicles (given the prior exclusion of fully electric cars). Consequently, for these specific instances, it is reasonable to set both the Electric Range and Electric Energy Consumption to zero. This deliberate adjustment ensures a more accurate representation of the dataset, aligning with the inherent characteristics of vehicles lacking electric support. By systematically addressing missing values in this manner, we enhance the coherence of our data, paving the way for more precise analyses and insights in our study.

Innovative Technologies and Emission Reduction through technology

Similar reasoning extends to the fields of innovative technologies and emission reduction through those technologies, specifically the 'Erwltp_g/km' (Electric Range weighted by the WLTP test cycle) and 'IT' (Innovative Technologies) columns. The presence of missing values in these columns directly implies the absence of the respective attributes. In light of this, opting to set these missing values to zero emerges as a suitable strategy, aligning with the inherent absence of these features for the corresponding vehicles. By employing this approach, we not only fill in missing data points but also establish a consistent representation that accurately reflects the non-existence of electric range and innovative technologies for these specific instances.

Tailoring Fuel Consumption Imputation by Fuel Type

The variances observed in fuel consumption ranges across different fuel types underscore a critical consideration in handling missing values within the fuel consumption columns. Employing absolute mean or median values for imputation without regard to the specific fuel type could lead to inaccuracies in estimating fuel consumption rates for certain vehicles. To address this issue, a more refined approach involves filling missing values with the median consumption rate within each group of the same fuel type. This targeted strategy ensures that the imputed values align with the

characteristic fuel consumption patterns of vehicles sharing the same fuel type, thereby enhancing the accuracy and relevance of our dataset.

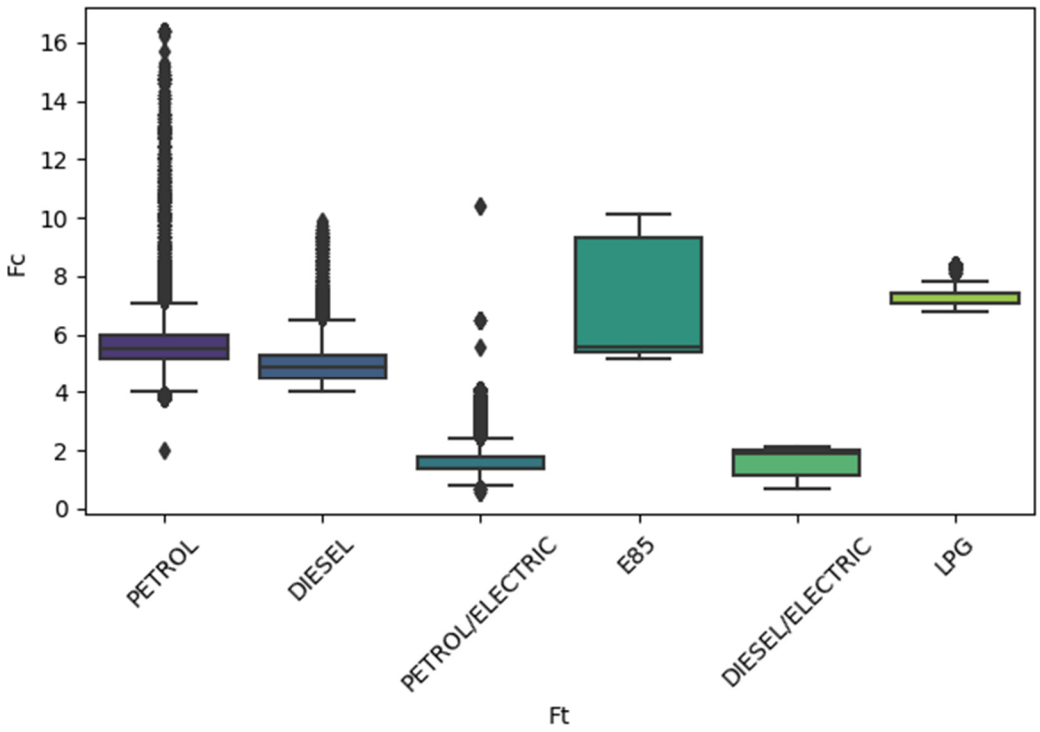


Figure 1. Fuel Consumption by Fuel Type

Regrouping 'fuel types'

An examination of the categorical distribution for fuel types has unveiled a significant imbalance among different categories. While there may not be substantial room for improvement, a thoughtful regrouping of certain categories can enhance the clarity of the analysis. Upon closer inspection, it is evident that the categories *'PETROL/ELECTRIC'* and *'DIESEL/ELECTRIC'* share similar characteristics, suggesting a potential consolidation. As a result, these two categories have been combined to form a single, more generalized group labeled *'HYBRID'*. Additionally, the *'E85'* and *'LPG'* categories, despite their uniqueness, can be regrouped for the sake of simplification and a clearer representation. Consequently, these two categories are now collectively referred to as *'OTHER'*. The graph below shows the distribution of Fuel types before and after.

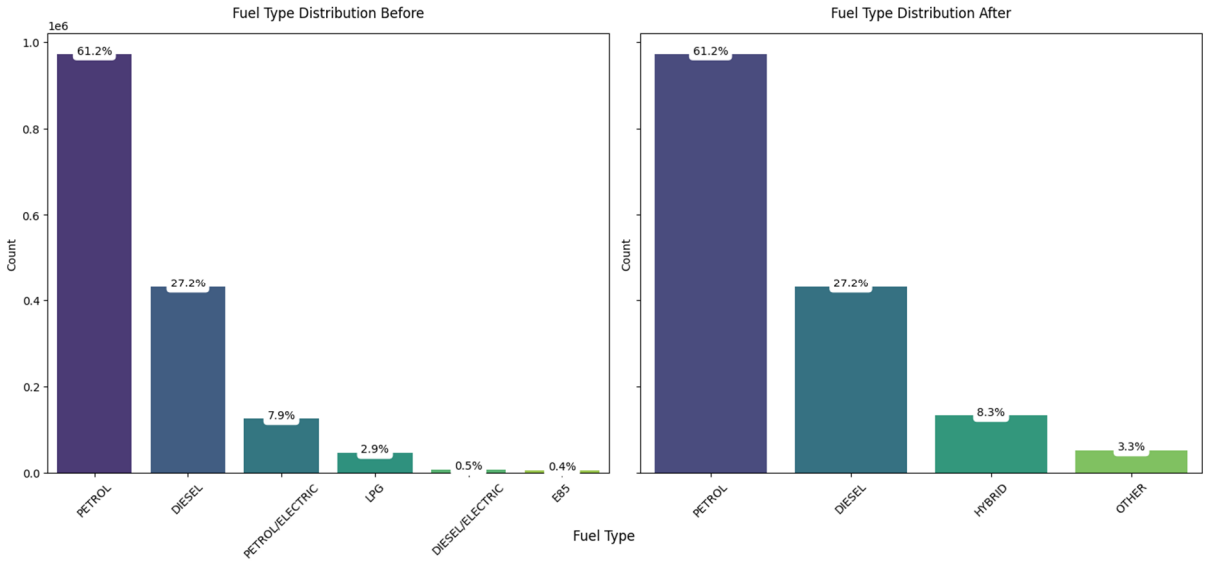


Figure 2. Fuel Type distribution before and after modification

Categorical Variables Encoding

Machine Learning models necessitate numerical data for effective processing. Therefore, it becomes imperative to employ encoding techniques to preprocess datasets for these models. One prominent solution in this regard is One-Hot Encoding. When dealing with a categorical variable featuring more than two categories that are ordinally independent, One-Hot Encoding proves to be a valuable method. This technique involves representing each category with a binary vector, where all values are set to 0 except for the position corresponding to the category of the observation, which is assigned the value of 1. This approach ensures that the model does not infer any ordinal relationships between categories, treating them as mutually independent entities.

In the actual dataset we are left with 3 categorical variables where One Hot Encoding is applied: 'Mp', 'Ft' and 'Fm'.

Exploratory Data Analysis (EDA)

Introduction

To gain a comprehensive understanding of the dataset on vehicle CO^2 emissions, our focus now shifts to the Exploratory Data Analysis (EDA) phase in this report. EDA serves as a crucial stage to delve deeper into the structure and characteristics of the data. Through the systematic application of various statistical methods and visualization techniques during the EDA phase, we aim to identify hidden patterns, significant trends, and potential factors influencing CO^2 emissions.

In this section, we will show the specific challenges and questions associated with our dataset and try to gain an understanding of the relationships between different variables. A thorough EDA is paramount to deepening our understanding of the underlying structures in the dataset, generating hypotheses, and, if necessary, planning further steps in data analysis. The insights from the EDA will serve as a foundation for the subsequent development of a Machine Learning model. Through this holistic approach, we connect exploratory data analysis with advanced modeling techniques, aiming not only to comprehend the past but also to enable forward-looking predictions in the context of vehicle CO^2 emissions.

Univariate Analysis

As the next step in our analytical journey, we are poised to conduct a univariate analysis. This fundamental approach involves examining individual variables to unravel insights into their distribution, central tendencies, and variations. Through this exploration, we aim to gain a preliminary understanding of the structure and characteristics of each variable in isolation.

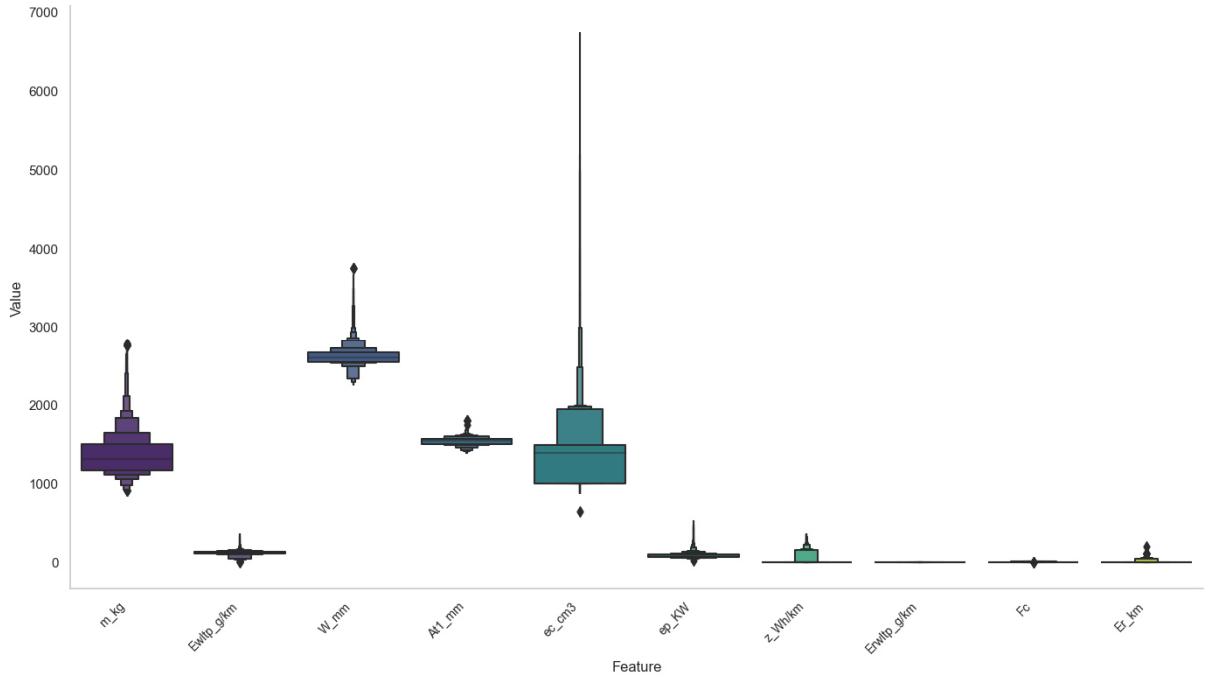


Figure 3. Boxplot Numerical Variables

The boxplot you see above, illustrates the distribution of numerical features in our dataset, providing insights into their central tendency and spread. As we can see, the variables in our dataset exhibit diverse scales and ranges, exemplified by the weight of vehicles ranging from 910 to 2785 kg and engine power varying between 28 and 537 kW. A prevalent characteristic is the positive skewness in most variables, indicated by an upward tail in their distributions. This suggests that the majority of vehicles tend to have lower values, but a subset of vehicles exhibits higher values, impacting the overall average. Additionally, the standard deviations for some variables are notably high, signaling a broad dispersion of data points around the mean.

Several variables, including 'Ewltp_g/km', 'ec _cm3', 'ep_KW', 'z_Wh/km', and 'Erwltp_g/km)', share similar distribution patterns marked by positive skewness. This commonality suggests that a significant portion of vehicles tends to have lower emissions, smaller engine displacements, lower power, reduced energy consumption, and lower emission factors. In contrast, variables such as 'm_kg', 'W_mm', 'At1_mm', and 'Fc' also exhibit positive skewness but with distinct ranges and scales.

The presence of outliers and extreme values in some variables poses potential challenges for modeling. Addressing these outliers with careful consideration is imperative to ensure model robustness. Furthermore, the positive skewness in distributions indicates a departure from normality, prompting exploration of transformations, such as log transformations, to align the data more closely with a normal distribution.

To prepare the data for modeling, it is crucial to identify and appropriately address outliers within the variables. Using transformations becomes essential, particularly if the chosen Machine Learning model assumes a normal distribution of data. Additionally, acknowledging the varying scales of the variables

is vital when selecting algorithms to guarantee their adaptability to the diverse magnitudes present in the dataset. These general insights lay the groundwork for more in-depth analyses of individual variables, providing a comprehensive understanding of the dataset for effective model development.

Manufacturer ('Mp')

The following visuals offer a detailed look into the landscape of non-electric vehicles registered in 2021, shedding light on their distribution and average CO^2 emissions. The first barplot provides a comprehensive overview of the dataset, showcasing the distribution of car counts among various manufacturers. Let's delve into the details:

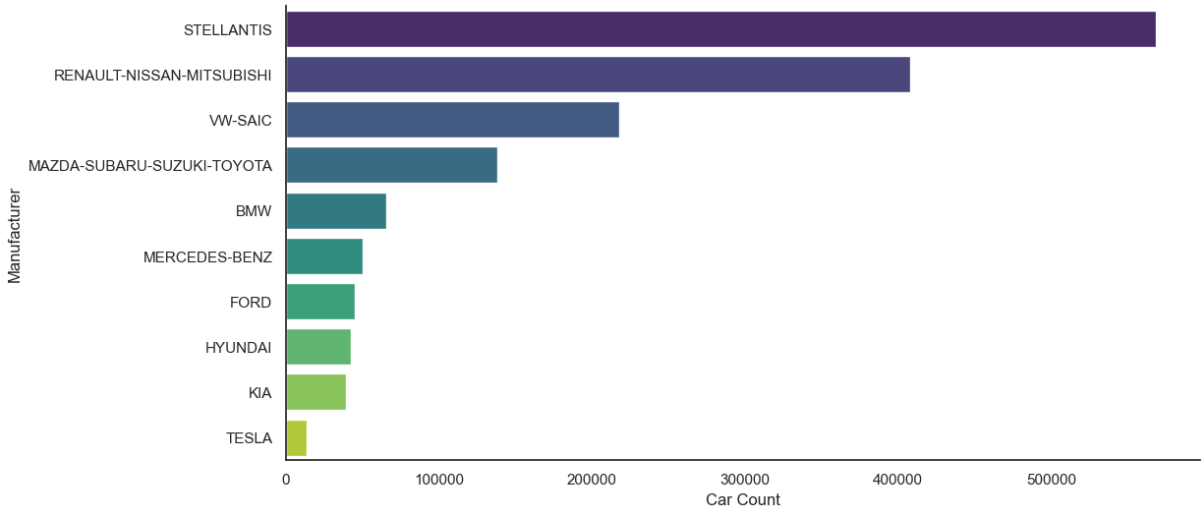


Figure 4. Barplot Manufacturer Count

- ***'Stellantis'*:** With a count of 568,603 cars, Stellantis emerges as the leading manufacturer in terms of dataset representation, holding a significant share.
- ***'Renault-Nissan-Mitsubushi'*:** Following closely, this alliance contributes 407,971 cars, solidifying its presence as a major player in the dataset.
- ***'VW-SAIC'*:** The manufacturer VW-SAIC holds a substantial but slightly smaller share, with 218,025 cars.
- '***Mazda-Subaru-Suzuki-Toyota*':** This group, comprising multiple manufacturers, presents a noteworthy presence with 138,220 cars.
- ***'BMW', 'Mercedes-Benz', 'Ford', 'Hyundai', 'Kia'*:** Each of these manufacturers adds to the dataset with varying car counts, reflecting their individual contributions and market presence.
- ***'Tesla'*:** While having the smallest count among the listed manufacturers, Tesla still represents a notable portion, contributing 13,553 cars.

The second barplot, depicting the average CO^2 emissions by manufacturer, provides valuable insights into the environmental performance of different carmakers. Let's explore the findings:

- ***'Tesla'*:** Notably, Tesla stands out with the highest average CO^2 emissions of 129.34 g/km, reflecting the unique characteristics of electric vehicles in the dataset.
- ***'VW-SAIC', 'Renault-Nissan-Mitsubishi', 'Kia', 'Ford',* 'BMW':** These manufacturers exhibit relatively close average CO^2 emissions, ranging from 120.37 g/km to 125.74 g/km. This suggests a comparable environmental impact across these brands.
- ***'Stellantis'*:** With an average CO^2 emission of 119.86 g/km, Stellantis positions itself slightly below the mean, contributing to the overall emissions landscape.

- *'**Hyundai**'*: Hyundai follows with an average of 115.94 g/km, indicating a relatively lower environmental impact per kilometer.
- *'**Mazda-Subaru-Suzuki-Toyota**', '**Mercedes-Benz**'*: These manufacturers demonstrate even lower average CO^2 emissions, with values of 113.46 g/km and 106.39 g/km, respectively. This suggests a stronger commitment to environmental sustainability in their vehicle offerings.

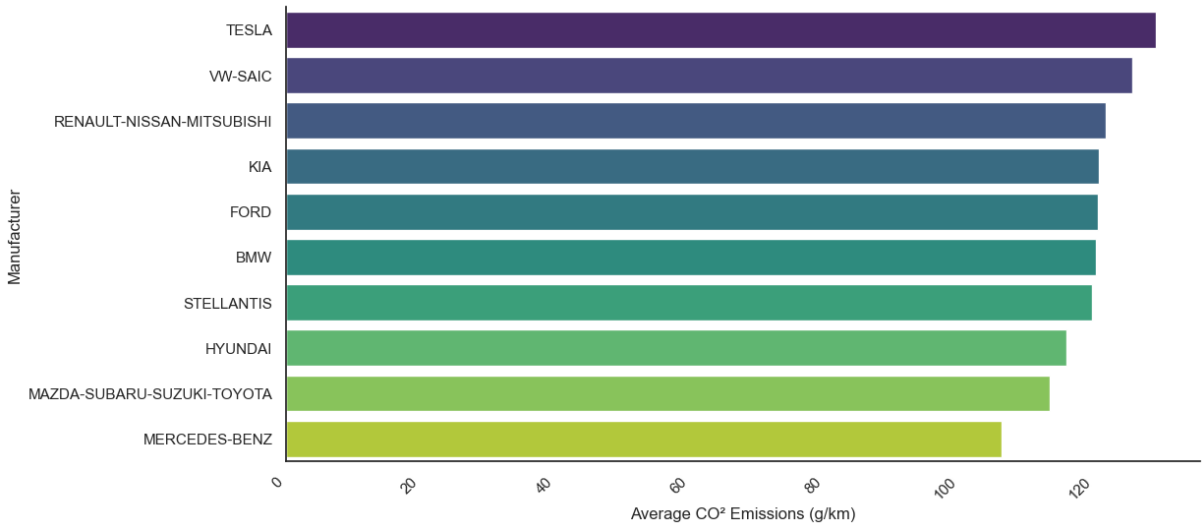


Figure 5. Barplot with average CO^2-Emissions by Manufacturer

CO^2-Emissions by Vehicle Category ('Cr')

In the dataset we can differentiate between two types of vehicle categories: *'M1'* and *'M1G'*. Both classes refer to categories designated by the United Nations Economic Commission for Europe (UNECE) for passenger cars in road traffic. This categorization is aimed at distinguishing various types of vehicles based on their design and use.

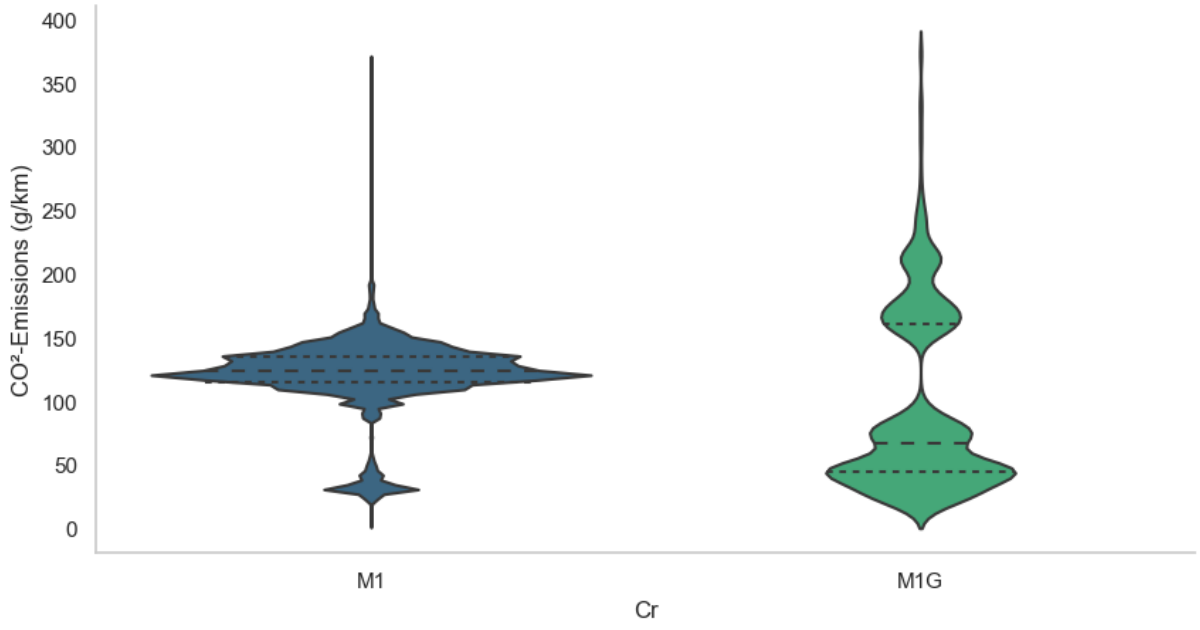


Figure 6. Violinplot of vehicle category

- *'**M1**'*: This category includes passenger cars designed and constructed for the transportation of passengers. It encompasses the majority of conventional cars intended for the individual transportation of people on public roads.
- *'**M1G**'*: This category pertains to passenger cars that, in addition to transporting passengers, can also be used for the transportation of goods. Typically, these are vehicles with a specialized design capable of carrying both passengers and cargo. In the dataset **only 1.9%** of all registered vehicles belong to this category.

Purpose of the categorization is to facilitate the differentiation of vehicle types in various traffic statistics and regulations, as they cater to different characteristics and purposes.

The violinplot for the 'Cr' variable, categorized into *'M1'* (passenger cars for individuals) and *'M1G'* (dual-purpose passenger and goods transport) categories, reveals distinct patterns in the distribution of CO^2 emissions for these vehicle.. In the case of M1 vehicles, designed primarily for individual transportation, the violinplot indicates a relatively tight and concentrated distribution around the median. The mean emissions are approximately 120.73 g/km, with a moderate spread. The distribution spans from 4 g/km to 368 g/km, reflecting a typical range for passenger cars.

On the other hand, M1G vehicles, designed for both passenger and goods transport, exhibit a broader distribution with higher variability. The mean emissions are lower at around 95.07 g/km, but the wider spread suggests a more diverse range of CO^2 emissions. Notably, there are outliers on the right side of the distribution, indicating some M1G vehicles with significantly higher emissions levels. The distribution for M1G vehicles spans from 18 g/km to 373 g/km.

In summary, the violinplot highlights the distinct emissions characteristics of M1 and M1G vehicle classes, aligning with their intended purposes.

Fuel Mode ('Fm')

The next barplot represents the distribution of different fuel modes ('Fm') within the dataset and offers insights into the diversity of fuel technologies present in the dataset.

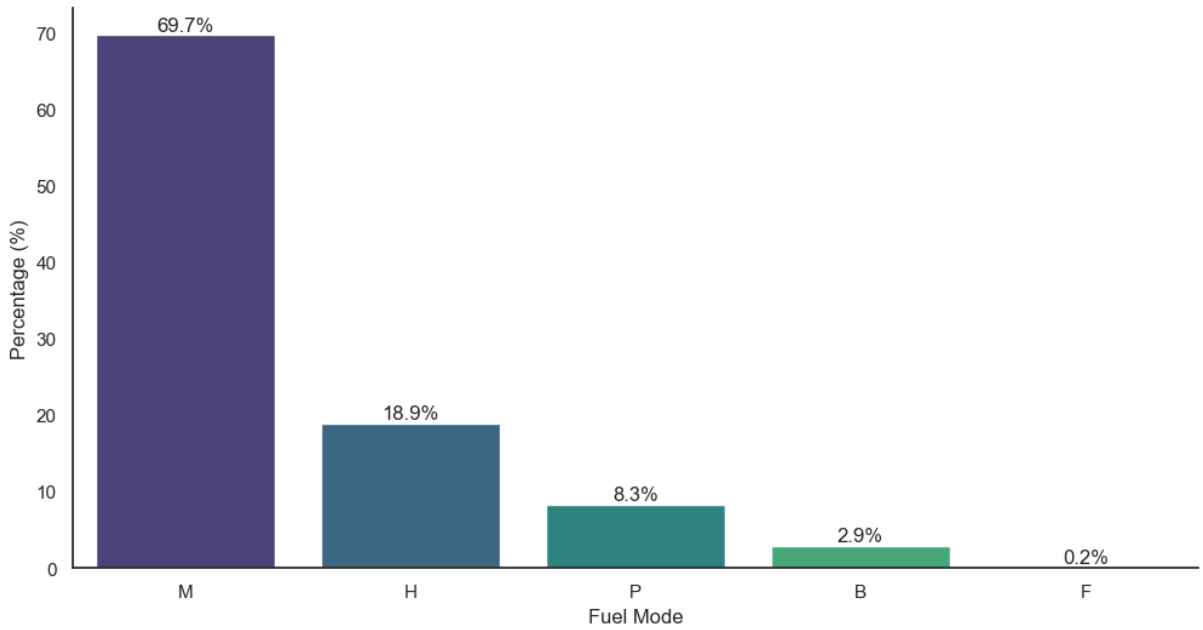


Figure 7. Fuel Modes

- **Monofuel (*'M'*):** This is the predominant fuel mode in the dataset, representing 69.7% of the vehicles. Monofuel vehicles operate solely on a single fuel type, such as gasoline, diesel, autogas, natural gas, or hydrogen.
- **Hybrid without External Charging (*'H'*):** Hybrid vehicles without the ability to charge externally account for 18.9% of the dataset. These vehicles rely on a combination of an internal combustion engine and an electric motor but cannot charge their batteries externally.
- **Plug-in Hybrid Electric Vehicles (*'P'*)**: Plug-in hybrid electric vehicles, which can charge externally, make up 8.3% of the dataset. These vehicles have both an internal combustion engine and a battery that can be charged via an external power source.
- **Bifuel (*'B'*):** Bifuel vehicles, capable of running on two separate fuel types, represent 2.9% of the dataset. These vehicles are designed to primarily operate on one fuel type but can switch to another.
- **Flex-Fuel (*'F'*):** Flex-Fuel vehicles, capable of running on various fuel mixtures, have the lowest representation at 0.2% in the dataset. These vehicles typically operate on a mixture of gasoline and ethanol.

Fuel Types (*'Ft'*)

The next barplot illustrates the distribution of various fuel types ('Ft') within the dataset and provides insights about into the prevalence of specific fuel types within the dataset.

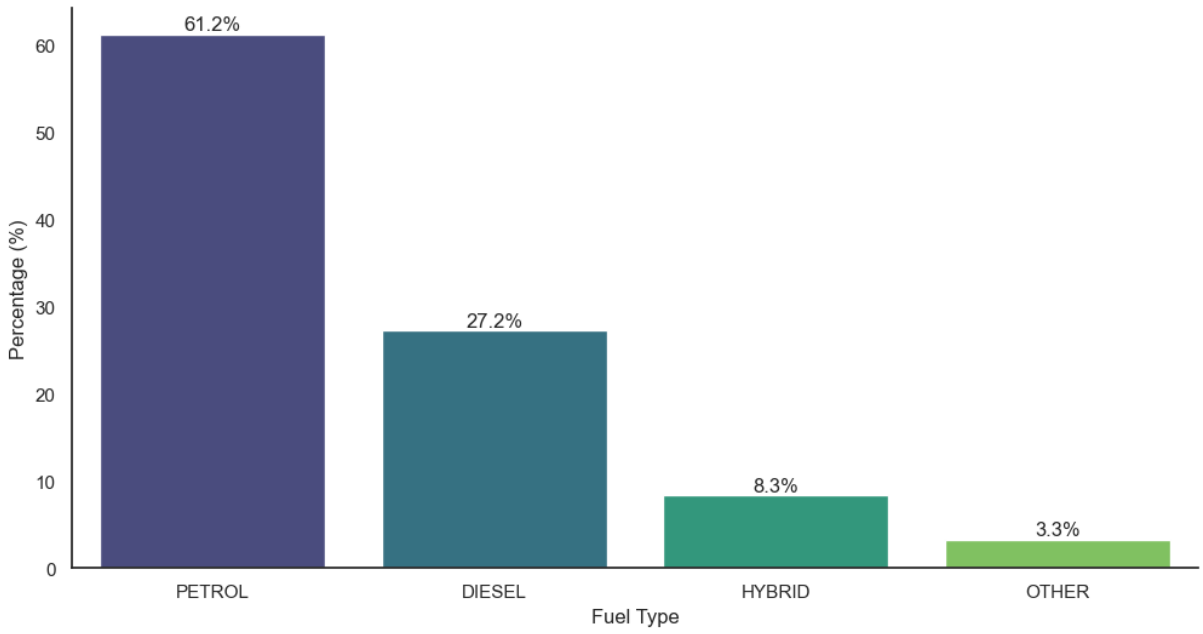


Figure 8. Fuel Types distribution

- *'**Petrol**'*: Petrol-fueled vehicles dominate the dataset, constituting 61.2% of the total. These vehicles rely on gasoline as their primary fuel source.
- *'**Diesel**'*: Diesel-fueled vehicles represent 27.2% of the dataset. Diesel engines are known for their fuel efficiency and are commonly used in various vehicle types.
- *'**Hybrid**'*: Hybrid vehicles, combining an internal combustion engine with an electric motor, make up 8.3% of the dataset. These vehicles can operate on both conventional fuel and electric power.
- *'**Other**'*: The category labeled as *'Other'* encompasses 3.3% of the dataset. This may include vehicles using alternative fuels or those with unique fuel characteristics not falling into the Petrol, Diesel, or Hybrid categories.

CO^2 emissions ('Ewltp_g/km')

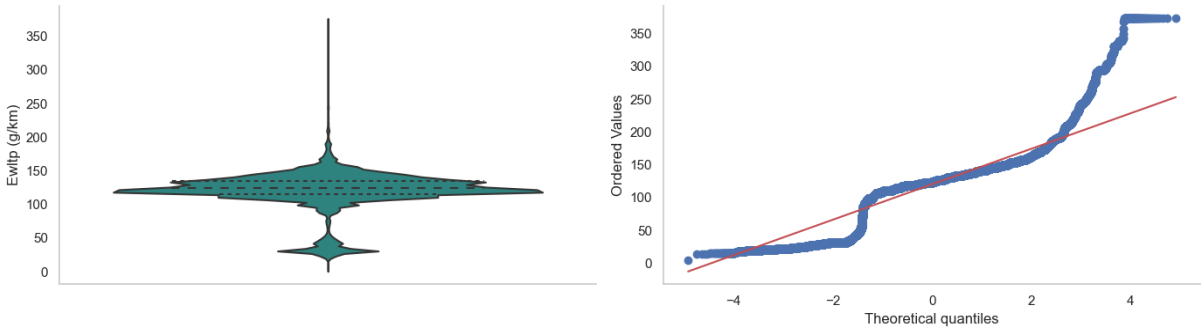


Figure 9. Violinplot and QQ plot for CO^2 emissions (g/km)

The violinplot for CO^2 emissions measured with the WLTP testing procedure ('Ewltp_g/km') depicts a positively skewed distribution, showcasing a wider upper part. This implies that higher CO^2 emissions per kilometer are less common, with the median at approximately 124. This observation is further supported by key statistical measures including a mean of around 120.23 g/km and a moderate spread of 30.03 g/km around this mean (standard deviation). An additional noteworthy observation is the presence of an apparent mode around 50 (g/km). This distinct mode indicates a concentration of vehicles with emissions centered around this value, contributing to a notable peak in the distribution.

The violinplot suggests a dual nature in the distribution, with a concentration of vehicles exhibiting moderate emissions around the median and a notable mode around 50 g/km.
The QQ-plot indicates a flattening of the empirical distribution in the lower quantiles, suggesting that lower values of CO^2 emissions per kilometer are less pronounced than expected in a perfect normal distribution. Conversely, a steeper rise in the upper quantiles suggests that higher values are more pronounced than predicted by a normal distribution. These deviations highlight that the distribution of CO^2 emissions per kilometer doesn't perfectly align with normality, indicating potential skewness, tails, or outliers influencing the distribution.

Fuel consumption ('Fc')

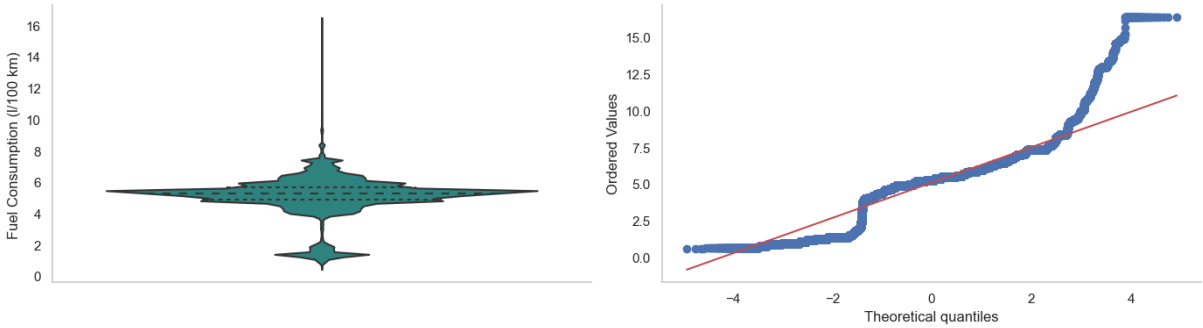


Figure 10. Violinplot and QQ plot Fuel Consumption

The violinplot for fuel consumption ('Fc') displays a positively skewed distribution similar to the pattern observed for 'Ewltp_g/km'. The wider upper part suggests that higher fuel consumption values per kilometer are less common, with the median at approximately 5.13 L/100 km. The statistical summary confirms this, indicating a mean of around 4.69 L/100 km and a moderate spread around this mean (standard deviation of approximately 15.84 L/100 km). However, a notable and distinct mode is observed a fuel consumption around 2 L/100 km. This mode indicates a concentration of vehicles with fuel consumption centered around this specific value, contributing to a prominent peak in the

distribution. The violinplot suggests a dual nature in the distribution, with a concentration of vehicles exhibiting moderate fuel consumption around the median and a significant mode around 2 L/100 km. Similar to the CO^2 emissions per kilometer QQ plot, this indicates a flattening of the empirical distribution in the lower quantiles, suggesting that lower values of fuel consumption are less pronounced than expected in a perfect normal distribution. Conversely, a steeper rise in the upper quantiles suggests that higher values of fuel consumption are more pronounced than predicted by a normal distribution.

These deviations highlight that the distribution of fuel consumption doesn't perfectly align with normality, indicating potential skewness, tails, or outliers influencing the distribution. The presence of a distinct mode around 50 L/100 km in the violinplot adds an interesting dimension to the distribution, suggesting a concentration of vehicles with specific fuel consumption values that deviate from the overall trend. Further investigation into the characteristics of vehicles contributing to this mode may provide valuable insights.

Vehicle width ('W_mm') and steering axle width ('At1_mm')

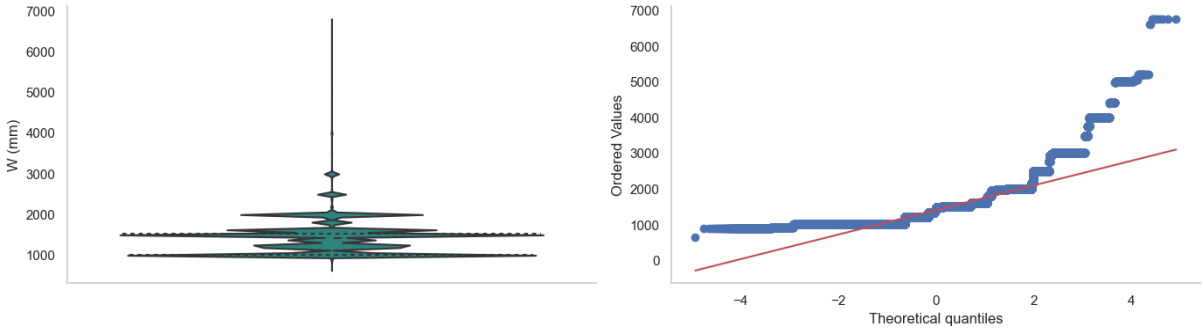


Figure 11. Violinplot and QQ plot for Vehicle Width (mm)

Transitioning from the insights gained from the analysis of 'Ewltp_g/km', we delve into the characteristics of two additional variables, 'W_mm' and 'At1_mm'.

The violinplot for 'W_mm' presents a distinctive pattern with numerous spikes, suggesting inherent variability in the data. This jagged appearance implies certain values or ranges are more prevalent, contributing to the irregular shape. Notably, the upper part of the distribution appears more pronounced, indicating a less frequent occurrence of higher values. Examining the statistical summary, we find a mean of approximately 2625.14 mm, a median around 2605 mm, and a moderate spread indicated by a standard deviation of roughly 123.44 mm.

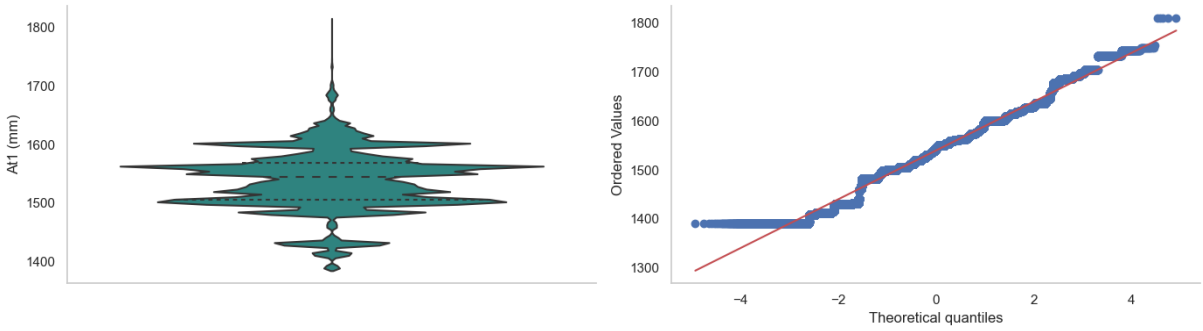


Figure 12. Violinplot and QQ plot of steering axle width ('At1_mm')

Similarly, the violinplot for 'At1_mm' exhibits a complex structure with spikes, indicative of specific data patterns The distribution highlights a more pronounced upper portion, suggesting that higher

values are less common. Key statistical measures reinforce this, revealing a mean of approximately 1539.53 mm, a median around 1544 mm, and a moderate spread with a standard deviation of about 50.41.

Both 'W_mm' and 'At1_mm' violinplots depict intricate structures, emphasizing the existence of certain data patterns or modes. The statistical summaries provide insights into the central tendency and variability of each variable. Notably, the pronounced upper portions of the violinplots suggest that higher values are less prevalent.

Examining the broader context, the descriptive statistics underscore the distributional characteristics. 'W_mm' spans from a minimum of 2250 mm to a maximum of 3750 mm, while 'At1_mm' ranges from 1390 mm to 1810 mm. These ranges align with the violinplots emphasis on specific value ranges. Both QQ plots deviate from the expected line, notably from the lower fourth quantile onward, indicating a departure from a perfectly normal distribution.

The diverse structure of the violinplots for width ('W_mm') and front axle distance ('At1_mm') reflects the fundamental logic and diversity within the vehicle data. The variations in widths and front axle distances are not uncommon and can be explained by the inclusion of vehicles from different manufacturers and models in the dataset.

Vehicles from various manufacturers and models inherently exhibit distinct geometric dimensions, especially concerning width and the distance between front axles. The multitude of spikes in the violinplots suggests that specific values or ranges in these dimensions are more prevalent..

The 8-value summary supports this context by presenting average values, standard deviations, and extreme values for 'W_mm' and 'At1_mm'. These variations are not only explainable but also expected when diverse vehicle types are represented in the data.

Identifying and interpreting these differences in geometric dimensions contribute to understanding the diversity of vehicles in the dataset. This understanding is crucial for exploring specific patterns and features that may impact CO^2 emissions or other performance factors.

Vehicle weight ('m_kg')

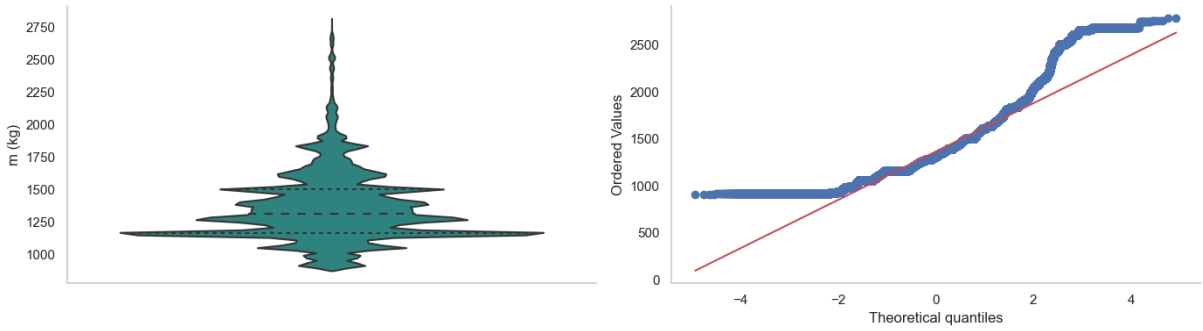


Figure 13. Violinplot and QQ plot for vehicle weight ('m_kg')

The violin plot for 'm_kg' (vehicle weight) reveals a fascinating distribution characterized by multiple spikes, indicating distinct concentrations of vehicles within specific weight ranges. The statistical summary enhances our understanding, providing key metrics such as the mean, standard deviation, and extreme values for 'm_kg'. The presence of multiple spikes suggests that certain weight categories are more prevalent, contributing to the multimodal nature of the distribution. The mean weight is approximately 1364.92 kg, with a moderate spread indicated by the standard deviation of around

268.08 kg. The complexity in the distribution is further emphasized by the minimum and maximum values of 910 kg and 2785 kg, respectively.

The QQ plot for 'm_kg' provides additional insights into the distribution, displaying deviations starting from the -lower second quantile below and the upper second quantile above the normal distribution line . These deviations suggest that the distribution of vehicle weights deviates from a perfect normal distribution. The plot indicates a steeper rise in the upper quantiles, implying that higher values of vehicle weight are more pronounced than expected in a normal distribution. The summary statistics and QQ plot align, emphasizing the non-normal characteristics of the distribution, particularly with an upward shift at the lower and upper second quantiles.

In summary, the violinplot and QQ plot unveil a complex distribution of vehicle weights, with multiple spikes indicating distinct weight categories. The key statistical measures provide a numerical backdrop, highlighting the variability and concentration of weights within the dataset. This multimodal distribution suggests that vehicles in the dataset fall into diverse weight clusters, prompting further exploration into the factors influencing these weight categories and their potential implications for other variables in the dataset.

Engine capacities ('ec_cm3') and engine power (‘ep (kW)’)

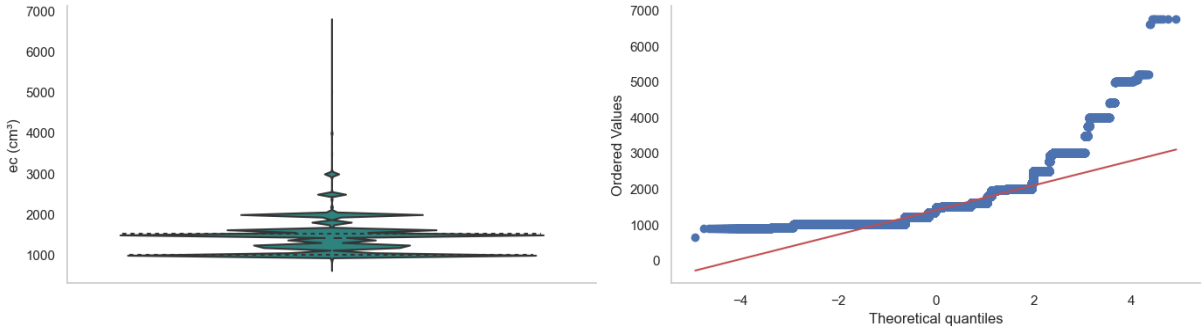


Figure 14. Violinplot and QQ plot for Engine Capacity

Moving on to the variables engine capacity in cubic centimeters ('ec_cm3') and engine power in kilowatts 'ep_KW', the violinplots reveal interesting patterns in the distribution of these engine-related features.
The violinplot for 'ec_cm3' showcases a distribution marked by multiple spikes, suggesting the presence of distinct concentrations of engine capacities. This variability is supported by the summary statistics, which provide the mean, standard deviation, and extreme values for 'ec_cm3'. The mean engine capacity is approximately 1396.17 cm^3, with a standard deviation of around 372.97, indicating moderate variability. The violinplot’s spikes indicate that certain engine capacities are more prevalent, contributing to the overall shape of the distribution.

The violinplot for 'ep_KW' reveals a distribution with pronounced variability in engine power. Similar to 'ec_cm3', the plot displays multiple spikes, indicating concentrations of specific power values. The statistical summary provides context, showing a mean power of approximately 89.01 kW, with a standard deviation of about 31.13. The presence of spikes suggests that certain power values are more prevalent within the dataset.

The QQ plots for both 'ec_cm3' and 'ep_KW' display deviations from normality, with an upward shift at the lower and upper second quantiles. These deviations indicate that the distributions of engine

capacity and power deviate from a perfect normal distribution. The descriptive statistics and QQ plots collectively highlight the non-normal characteristics of these engine-related features.

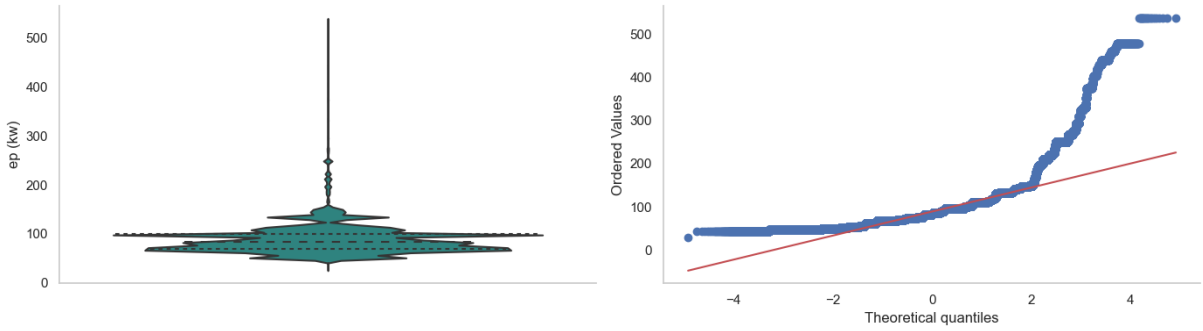


Figure 15. Violinplot and QQ plot for engine power

In summary**,** the violin plots and QQ plots for 'ec_cm3' and 'ep_KW' unveil intricate distributions with multiple spikes, suggesting concentrations of specific engine capacities and power values. The summary statistics provide numerical insights into the variability and central tendencies of these engine-related features, paving the way for further exploration into the factors influencing these concentrations within the dataset.

Electric range ('Er_km')

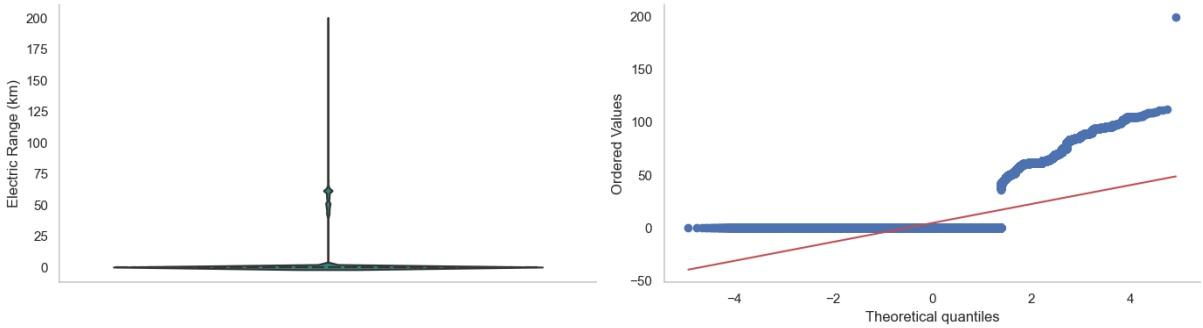


Figure 16. Violinplot and QQ plot for Electric Range ('Er_km')

The violin plot for 'Er_km' reveals a distinctive and intriguing pattern characterized by a flattened shape. This suggests that a significant portion of vehicles in the dataset appears to have no electric range or very limited electric range. The absence of a well-defined peak and the flatness of the plot indicate a concentration of vehicles with minimal or zero electric range.

The QQ-plot for 'Er_km' further supports this observation by displaying a horizontal line for observed values from the lower fourth and upper second quantile. This suggests a consistent distribution of observed values within this range, indicating that a substantial proportion of the dataset lacks electric range. Beyond the -4th quantile, the plot shows an upward shift, indicating that the observed values have a higher frequency than expected in a normal distribution. This suggests that vehicles with electric range values, although limited, are more prevalent than predicted by a normal distribution.

In summary, the flattened shape of the violin plot and the consistent horizontal line in the QQ-plot for the observed values from lower fourth and upper second quantile indicate a notable concentration of vehicles with little to no electric range. The subsequent upward shift in the QQ-plot suggests that the limited electric range values present in the dataset are more common than expected in a normal

distribution. Further investigation into the characteristics of these vehicles and potential factors influencing their electric range is warranted.

Energy Consumption ('z_Wh/km')

The variable 'z_Wh/km' represents the energy consumption per kilometer in watt-hours. It measures the amount of energy consumed by a vehicle for each kilometer traveled. This metric provides insights into the efficiency of a vehicle's energy consumption, indicating how much electrical energy is required to cover a unit distance. A lower value for 'z_Wh/km' implies higher energy efficiency, which is desirable for electric vehicles or hybrids. This variable is crucial for assessing the energy efficiency and sustainability of a vehicle's operation.

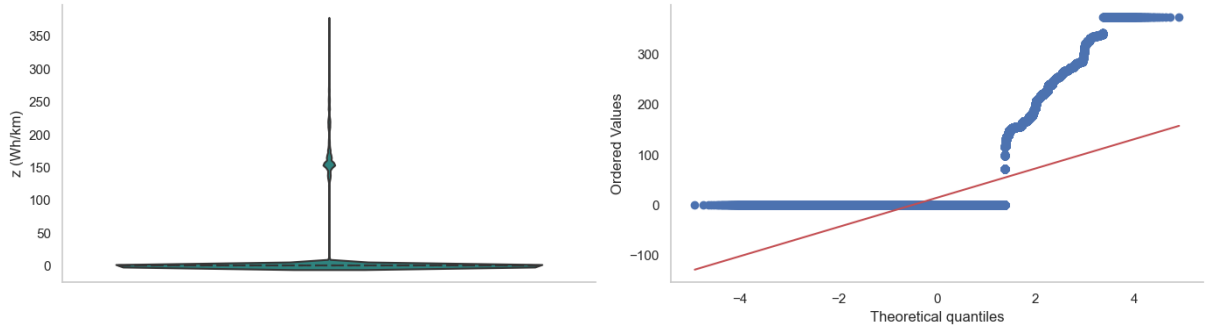


Figure 17. Violin plot and QQ plot for energy consumption ('Z_Wh/km')

The violin plot for energy consumption per kilometer ('z_Wh/km') appears similar to the pattern observed in the Electric Range variable. It exhibits a flattened shape, suggesting that a substantial number of vehicles in the dataset have minimal or zero energy consumption per kilometer. The absence of a well-defined peak and the flatness of the plot indicate a concentration of vehicles with low energy consumption or no consumption at all.

The QQ-plot for 'z_Wh/km' supports this observation, displaying a horizontal line for observed values from the -lower fourth and upper second quantile. This consistent distribution of observed values within this range suggests that a significant proportion of the dataset exhibits minimal energy consumption. Beyond the fourth quantile, the plot shows an upward shift, indicating that the observed values with energy consumption are more prevalent than expected in a normal distribution.

The similarities between the violin plot and QQ-plot for 'z_Wh/km' and the Electric Range variable suggest a commonality in the distribution patterns of these two variables. The flattened shape and the consistent horizontal line in the QQ-plot indicate a concentration of vehicles with low or no energy consumption per kilometer. The subsequent upward shift suggests that, although limited, energy consumption values are more common than expected in a normal distribution.
In summary**,** the patterns observed in the violin plot and QQ plot for 'z_Wh/km' indicate a significant presence of vehicles with minimal energy consumption or no consumption per kilometer. Further exploration into the characteristics of these vehicles and the factors influencing their energy consumption is warranted.

Emissions reduction through innovative technologies ('Erwltp_g/km')

The variable 'Erwltp_g/km' stands for "Emissions reduction through innovative technologies per kilometer" and measures a vehicle's CO^2 emission reductions achieved through innovative technologies. It indicates how effectively a vehicle reduces CO^2 emissions per kilometer through advanced technologies and environmental innovations, expressed in grams per kilometer (g/km). A

higher value signifies greater effectiveness in reducing CO^2 emissions for each kilometer traveled. As a result, this variable is relevant for assessing the environmental impact and sustainability of vehicles.

The violin plot for 'Erwltp_g/km' unveils a distribution with multiple modes, notably a significant mode around 0 g/km. This indicates a concentration of vehicles with minimal or no reported CO^2 emission reductions through environmental innovations. The width of the plot around 0 g/km signifies a substantial number of vehicles falling into this category. Additionally, other modes may suggest variations in the effectiveness of environmental technologies across different vehicles.

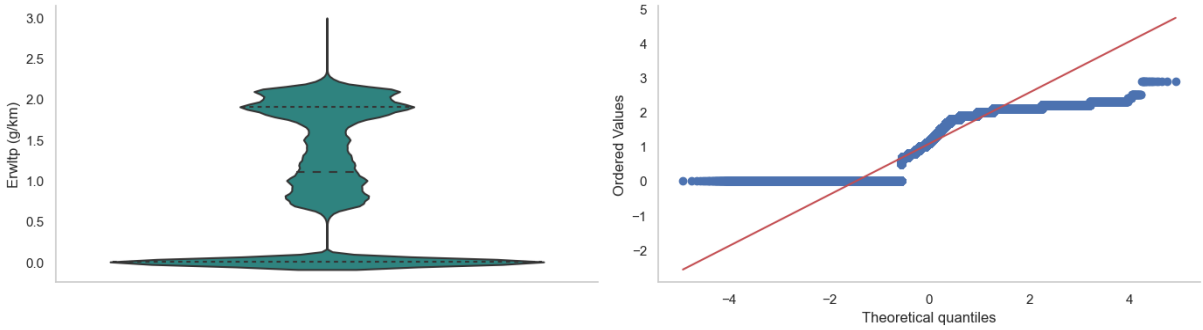


Figure 18. Violin plot and QQ plot Erwltp (g/km)

The statistical summary aligns with these observations, indicating that the mean is around 1.08 g/km, and the median (50th percentile) is 1.10 g/km. The standard deviation of approximately 0.80 implies a moderate spread around the mean. Notably, the 25th percentile is 0.00 g/km, emphasizing the prevalence of vehicles with reported emission reductions close to zero.

The QQ plot for 'Erwltp_g/km' corroborates the findings from the violin plot. A horizontal line is observed from approximately -4 to 0 on the quantile scale, indicating a consistent empirical distribution in this range. Beyond the 0 quantile, the line deviates below the expected normal distribution, suggesting that the observed values have a lower frequency than what a normal distribution would predict. This is consistent with the concentration of vehicles around 0 in the violin plot.

These combined insights from the violin plot, QQ plot, and descriptive statistics underscore that a significant proportion of vehicles in the dataset exhibit minimal or no reported CO^2 emission reductions as captured by 'Erwltp_g/km'. The multiple modes in the distribution indicate potential diversity in the effectiveness of environmental innovations across different vehicles. The statistical summary provides quantitative measures that support and complement the visual observations, offering a comprehensive understanding of the dataset's characteristics related to emissions reductions.

Multivariate Analysis

Following the univariate analysis, our trajectory leads us to the realm of multivariate analysis. This advanced analytical approach delves into the interdependencies and relationships among multiple variables within our dataset. Multivariate analysis enables us to unravel complex patterns, correlations, and interactions that may not be apparent in isolation. By exploring the collective impact of multiple variables, we aim to gain a more holistic understanding of the intricate dynamics within our data. This phase of analysis will pave the way for nuanced insights and informed decision-making in our broader statistical exploration.

Correlation of numerical features

Correlation analysis is a powerful statistical method that allows for the identification and quantification of relationships between variables. By exploring correlations, researchers and data analysts gain

valuable insights to understand patterns, associations, and trends within a dataset. This analytical technique is crucial in exploratory data analysis (EDA) and supports the derivation of hypotheses regarding potential causes and effects.

At its core, correlation analysis measures the strength and direction of linear relationships between variables. The most commonly used correlation coefficient, the Pearson correlation coefficient, ranges from -1 to 1. A value of 1 indicates a perfect positive linear relationship, -1 indicates a perfect negative relationship, and 0 indicates no linear relationship. However, it's essential to emphasize that correlation does not imply causation; a high correlation between two variables does not necessarily mean that one variable causes the other.

Correlation analysis allows for the identification of patterns of behavior, dependencies, and independencies between variables. It is applied across various disciplines such as economics, medicine, psychology, and environmental sciences to understand relationships and make informed decisions.

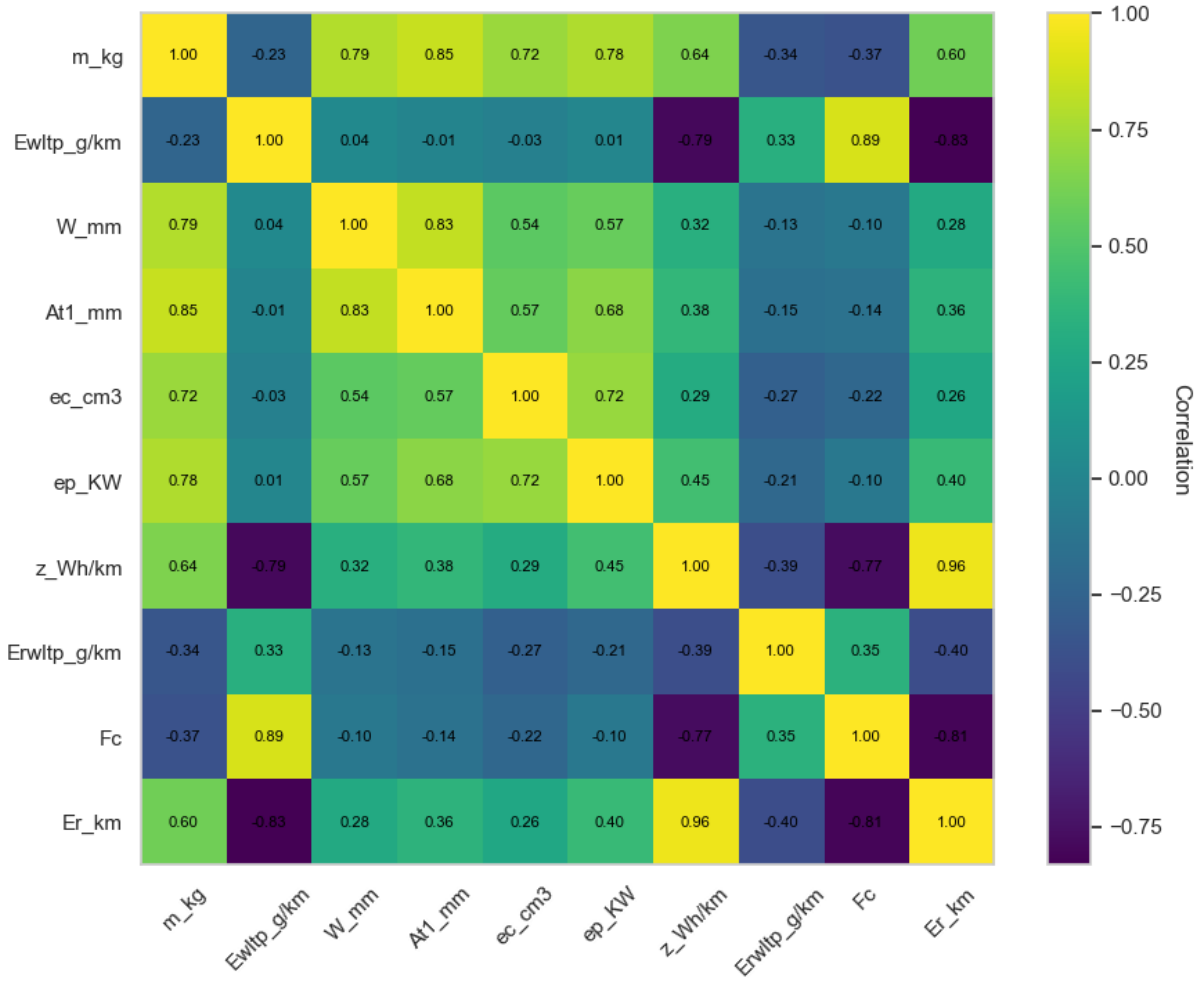


Figure 19. Correlation Matrix

In this analysis, our focus will be on interpreting correlation matrices and pair plots. These visual representations aid in recognizing relationships between variables, providing deeper insights into the structure of our data. Through the proper application of correlation analysis, we can extract patterns in our data and develop an understanding of complex relationships.

When we take a look at the correlation matrix, the dataset reveals several noteworthy insights into the relationships among various numerical variables.

Vehicle Weight ('m_kg')

- **Associations:** Positive correlation (0.79) between weight and vehicle width (W (mm)), indicating broader vehicles tend to be heavier. Positive correlation (0.72) between weight and engine capacity ('ec_cm3'), suggesting vehicles with greater weight also tend to have larger engines.
- **Environmental Impact:** Negative correlation (-0.23) with CO^2 emissions per kilometer ('Ewltp_g/km'), suggesting lighter vehicles tend to have higher CO^2 emissions per kilometer traveled.
- **Performance:** Strong positive correlation (0.78) between weight and power (ep (kW)), indicating heavier vehicles also tend to possess higher power.

CO_2 Emissions ('Ewltp_g/km'):

- **Weight Influence:** Negative correlation (-0.23) with vehicle weight, indicating lighter vehicles generally contribute more to CO_2 emissions per kilometer.
- **Interconnected Metrics:** Positive correlation (0.33) with emissions reduction through innovative technologies ('Erwltp_g/km') and fuel consumption ('Fc'), indicating intertwined relationships. Strong negative correlation (-0.79) with electric consumption ('z_Wh/km'), suggesting electric vehicles with lower CO_2 emissions tend to have higher electric consumption.
- **Electric Range Indicator:** Negative correlation (-0.83) with electric range ('Er_km'), showing vehicles with higher CO_2 emissions per kilometer tend to have a lower electric range.

Vehicle Dimensions:

- **Width and Weight Dynamics:** Positive correlation (0.79) between vehicle width and weight, confirming wider vehicles tend to be heavier. Positive correlation (0.57) between frontal area (At1 (mm)) and weight, indicating vehicles with larger frontal areas also tend to be heavier.

Engine Characteristics:

- **Capacity Dynamics:** Positive correlations (0.72, 0.54, 0.57) between engine capacity ('ec_cm3') and weight, width, and power, respectively. Indicates vehicles with larger engines tend to be heavier, wider, and more powerful.

Power (ep (kW)):

- **Comprehensive Associations:** Positive correlations (0.78, 0.72, 0.57) with weight, width, and engine capacity, respectively. Vehicles with greater weight, width, and engine capacity tend to possess higher power.
- **Impact on Emissions Reduction:** Negative correlation (-0.22) with emissions reduction through innovative technologies ('Erwltp_g/km'), suggesting vehicles with higher power may have lower emissions through innovative technologies.

Electric Consumption ('z_Wh/km'):

- **Emissions-Consumption Dynamics:** Negative correlation (-0.79) with CO^2 emissions, indicating electric vehicles with lower CO^2 emissions tend to have higher electric consumption. Positive correlation (0.96) with expected CO^2 emissions, suggesting vehicles with higher expected emissions also have higher electric consumption.

Emissions Reduction through Innovative Technologies ('Erwltp_g/km'):

- **Fuel Consumption Relationship:** Positive correlation (0.35) with fuel consumption ('Fc'), indicating a connection between expected emissions and fuel consumption.
- **Electric Range Indicator:** Negative correlation (-0.40) with electric range ('Er_km'), suggesting vehicles with higher emissions reduction through innovative technologies tend to have a lower range.

Figure 20. Pairplot

Fuel Consumption ('Fc'):

- **Inverse Relationship with Electric Consumption:** Strong negative correlation (-0.77) with electric consumption ('z_Wh/km'), indicating an inverse relationship between fuel consumption and electric consumption.
- **Efficiency Dynamics:** Strong negative correlation (-0.81) with electric range ('Er_km'), suggesting vehicles with lower fuel consumption tend to have a higher electric range.

Electric Range ('Er_km'):

- **Weight and Electric Range:** Positive correlation (0.60) with weight, implying heavier vehicles, on average, tend to have a lower electric range.
- **Emissions-Electric Range Connection:** Negative correlation (-0.83) with CO^2 emissions, indicating vehicles with higher emissions per kilometer also tend to have a lower electric range.

By underpinning the detailed insights with the corresponding numerical values, we gain a more precise understanding of the relationships within the dataset, enhancing the significance of the findings in the context of vehicle characteristics and environmental impact.

CO^2-emissions by fuel type

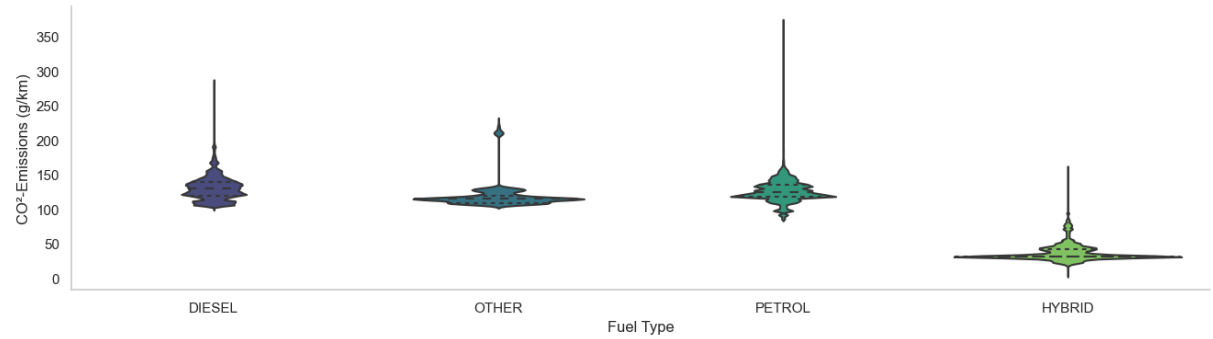


Figure 21. Histogram of CO^2 emissions by fuel type

The depicted violin plots and histograms offer a visual exploration of CO^2 emissions ('Ewltp_g/km') across various fuel types ('Ft'). In the violin plots above, the distribution of emissions for each fuel type is illustrated. Slimmer violin shapes, as observed in *'DIESEL'* and *'PETROL'*, suggest relatively lower variance, while broader shapes in *'OTHER'* and *'HYBRID'* indicate higher variability. The central markers within each violin represent the median emissions for the respective fuel types.

In the next figure, the superimposed Histogram unveils the frequency distribution of CO^2 emissions. Noteworthy patterns emerge: *'DIESEL'* showcases a concentrated distribution with a distinct peak, *'PETROL'* displays a broader distribution with a prominent peak, and *'OTHER'* maintains a more even and wider spread. *'HYBRID'* exhibits a unique narrow distribution with a characteristic peak at lower emissions.

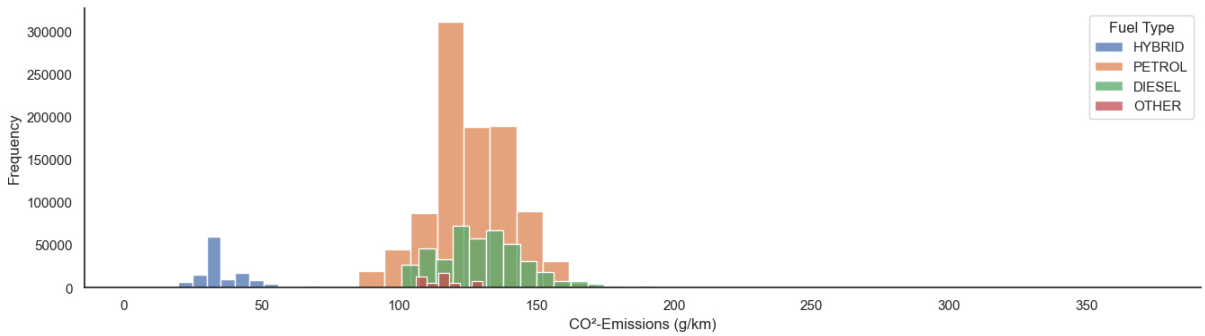


Figure 22. Violin plot CO^2-Emissions by Fuel Type

A detailed examination, considering the 8-value summary, provides additional insights:

- ***'DIESEL'*** exhibits low standard deviation and variance, signifying a relatively consistent emission pattern. The median is at 131.00, reflecting a central tendency.
- ***'PETROL'*** also displays low standard deviation and variance, along with a sharper histogram peak at a median of 126.73.
- ***'OTHER'*** and ***'HYBRID'*** present higher variances, implying broader distributions. 'HYBRID' stands out with the lowest median emissions at 37.39.
- In the Histogram, ***'OTHER'*** consistently maintains a more even and wider spread across emissions. While it does not necessarily indicate lower emissions on average, it suggests a broader and more diverse distribution of CO^2 emissions within the *'OTHER'* fuel type.

In conclusion, these visualizations enhance our understanding of the distribution and characteristics of CO^2 emissions for different fuel types. *'DIESEL'* and *'PETROL'* exhibit specific patterns, while *'OTHER'* and *'HYBRID'* demonstrate broader variance. *'OTHER'* consistently displays lower emissions, providing valuable insights into the environmental impact of distinct fuel types.

CO^2-Emissions by fuel mode

The presented violin plots and Histograms provide a visual representation of CO^2 emissions ('Ewltp_g/km') based on different fuel modes ('Fm'). In the left plot, the violin plots illustrate the distribution of emissions for each fuel mode. Narrow violin shapes in *'B'* and *'F'* suggest low variance, while broader shapes in *'H'* and *'M'* indicate higher variance. The central markings represent the median values of emissions for each mode.

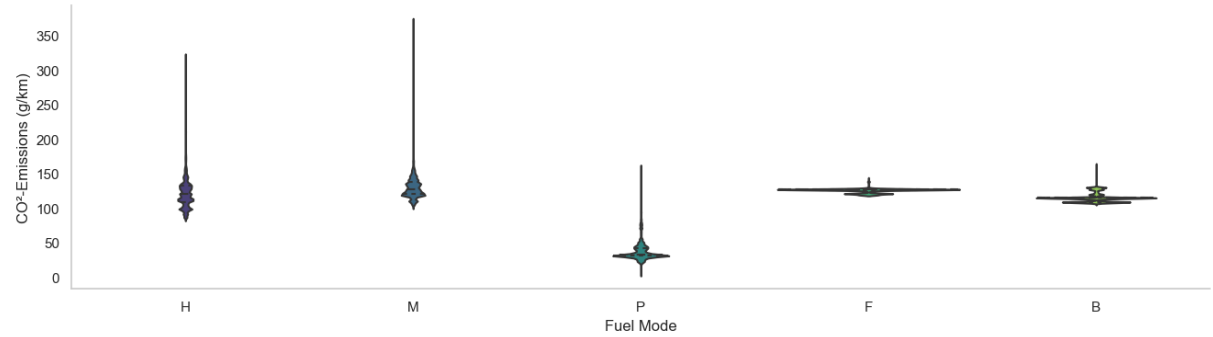


Figure 23. Violin plot CO^2-emissions by fuel mode

When we take a look at the next figure, the overlaid Histogram displays a frequency distribution of CO^2 emissions. Notably, *'B'* exhibits a narrow distribution with a distinct peak, while *'F'* presents a broader distribution with a sharp peak. *'P'* shows low frequencies and a narrow distribution. *'H'* and *'M'* display broader distributions with varying peaks.

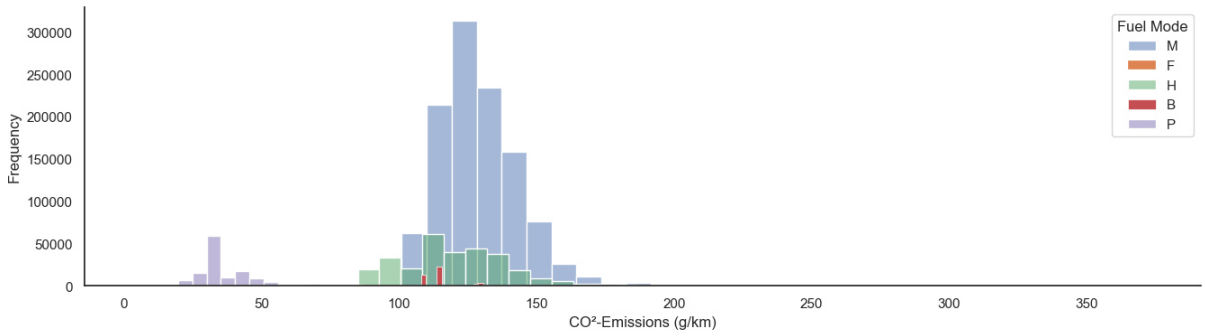


Figure 24. Histogram CO²-Emissions by Fuel Mode

A comparison with the 8-value summary further elucidates these patterns:

- ***'B'*** demonstrates low standard deviation and variance, with a median of 115.00 and a small interquartile range.
- ***'F'*** exhibits low standard deviation, low variance, and a median of 127.00, with a sharper histogram peak.
- ***'H'*** and *'M'* display higher variances and broader distributions.
- ***'P'*** maintains consistent and low CO^2 emissions.

In summary, these plots provide insights into the distribution and characteristics of CO^2 emissions for different fuel modes. *'B'* and *'F'* seem to show specific patterns, while *'H'* and *'M'* exhibit broader variance, and *'P'* consistently displays low emissions.

Comparative Analysis of CO²-emissions by manufacturer

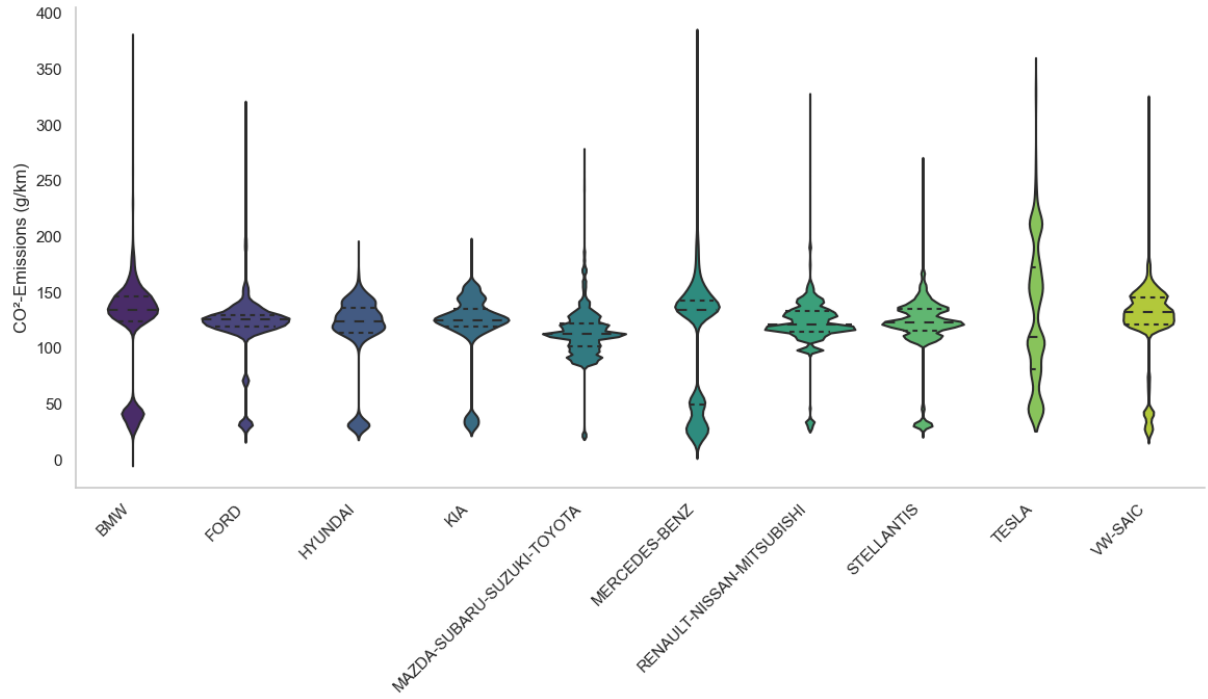


Figure 25. Violin plot CO²-emissions by manufacturer

Let us deepen our understanding of our target variable, the carbon dioxide (CO^2) emissions, measured in grams per kilometer ('Ewltp_g/km') by a detailed exploration, across various automotive manufacturers ('Mp'). Utilizing violin plot visualizations and 8-value summary statistics, the analysis aims to uncover patterns, variations, and potential outliers within the emissions data. Notably, it's

crucial to highlight that all electric vehicles have been excluded from the dataset in former steps, significantly impacting the results, especially for manufacturers prominently associated with electric vehicle production, such as TESLA. Let us summarize the main insights:

Manufacturer Averages:

- TESLA emerges with the highest average CO^2 emissions, standing at 129.34 g/km, closely followed by VW-SAIC at 125.74 g/km.
- Manufacturers such as MAZDA-SUBARU-SUZUKI-TOYOTA and STELLANTIS demonstrate relatively lower average emissions, at 106.39 g/km and 113.46 g/km, respectively.

Variability in Emissions:

- TESLA and STELLANTIS exhibit wider variability in CO^2 emissions, as denoted by the broader shapes of their violin plots.
- HYUNDAI, KIA, and MERCEDES-BENZ, conversely, showcase narrower violin shapes, indicating a more concentrated distribution of emission values.

Outlier Analysis:

- TESLA and VW-SAIC prominently display higher outlier values, evidenced by the extended "tails" in their violin plots. This infers the presence of specific vehicles within these categories characterized by exceptionally high CO^2 emissions.

Manufacturer Groups Overview:

- BMW, FORD, HYUNDAI, KIA, and MERCEDES-BENZ share analogous average CO^2 emissions.
- MAZDA-SUBARU-SUZUKI-TOYOTA exhibits a slightly lower average emission.
- TESLA and VW-SAIC stand out with higher average emissions, with TESLA presenting a wider distribution.

Quartile Comparison:

- TESLA and VW-SAIC present notably higher upper quartiles (75%), suggesting a more significant proportion of vehicles with elevated emissions in these categories.

Identifying Trends in TESLA and VW-SAIC:

- The asymmetrical violin shapes for TESLA and VW-SAIC suggest potential variations within these manufacturers, possibly stemming from different subcategories or distinct vehicle models.

Range of Values:

- Minimum and maximum CO^2 emission values demonstrate considerable variation among manufacturers, with TESLA showcasing the highest maximum value.

Conclusion: This segment of the report underscores critical insights into the distribution and variation of CO^2 emissions across diverse automotive manufacturers. The exclusion of all electric vehicles from the dataset significantly impacts the results, particularly for manufacturers like TESLA, renowned for their production of electric vehicles. Understanding these contextual nuances is imperative for interpreting the findings accurately. Further detailed analyses and exploration of potential influencing factors are recommended in subsequent sections of the report to provide a comprehensive understanding of the observed patterns.

Distribution of numerical features by CO^2-etiquette

Number of vehicles by CO^2-etiquette

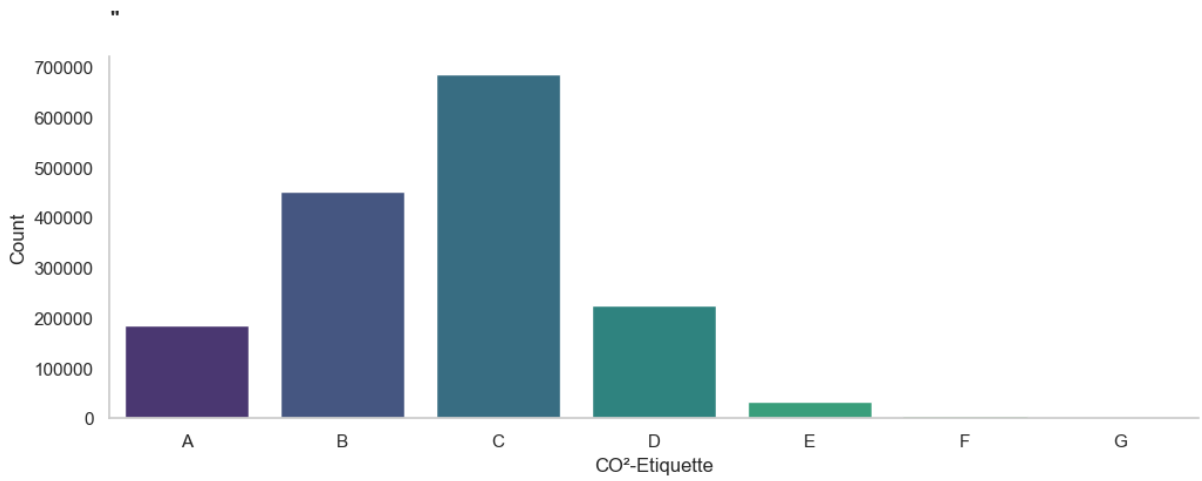


Figure 26. Count of CO^2-etiquettes

The next bar plot provides a simple and comprehensive view of the frequency distribution of vehicles based on their CO^2 labels, which correspond to specific emission classes. Let's delve into the details:

- The most prevalent CO^2 label is ***'C'***, with emissions falling within the range of 121 to 140 g/ CO^2/km. This class is dominant, encompassing 686,973 vehicles and indicating that a substantial portion of the analyzed vehicles emit CO^2 within this moderate range.
- Following closely is the ***'B'* category**, representing emissions from 101 to 120 g/ CO^2/km. With 452,121 vehicles falling into this class, it signifies a significant presence of vehicles with relatively lower CO^2 emissions.
- The ***'D'* category**, with emissions ranging from 141 to 160 g/ CO^2/km, is the next most frequent, accounting for 225,878 vehicles. This class represents vehicles with emissions falling within a specific moderate range.
- Moving to lower frequency, the ***'A'* category,** indicating emissions of 100 g/CO^2/km or less, encompasses 185,011 vehicles. This suggests a considerable number of vehicles with the lowest CO^2 emissions.
- **Categories *'E'*, *'F'*, and *'G'*** represent emissions ranging from 161 to 200 g/CO^2/km, 201 to 250 g/CO^2/km, and over 250 g/CO^2/km, respectively. These categories exhibit notably lower frequencies: *'E'* has 33,519 vehicles, *'F'* has 4,813 vehicles, and *'G'* has 1,433 vehicles. These lower frequencies signify a relative scarcity of vehicles with higher CO^2 emissions in the dataset.

Distribution of vehicle width ('W_mm') by CO^2-etiqette

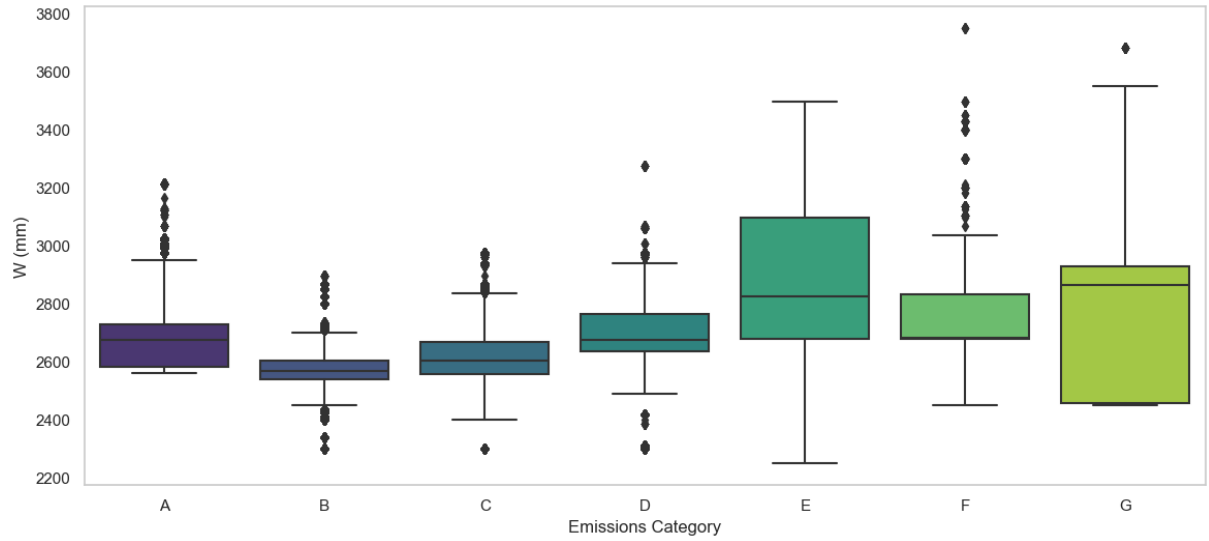


Figure 27. Boxplot vehicle width by CO^2-etiquette

The boxplot illustrates the distribution of vehicle width ('W_mm') within each emission category. Here's how we can interpret them based on the provided data:

- **Box Extent (Interquartile Range):** The height of the box in each plot represents the interquartile range (IQR), indicating the middle 50% of the data. Categories *'E'*, *'F'*, and *'G'* have taller boxes, suggesting a broader range of vehicle widths within these emission categories.
- **Line Inside the Box (Median):** The line inside each box represents the median vehicle width. While Category *'C'* has a notable median width, *'E'*, *'F'*, and *'G'* exhibit higher median values, indicating wider vehicles on average.
- **Whiskers (Minimum and Maximum):** The whiskers extend to the minimum and maximum values within a certain range. Category *'G'* has the longest whiskers, suggesting the widest overall span of vehicle widths. However, *'E'* and *'F'*, also display extended whiskers, indicating greater variability in widths within these categories.
- **Evaluation of Outliers:** Outliers, are the individual data points outside the whiskers. In our case vehicles with CO^2-Etiquette *'A'*, *'B'*, *'C'*, *'D'*, *'E'*, and *'F'* show the most outliers, signifying extreme values and further emphasizing the variability in vehicle widths within these emission categories.

  The reported maximum vehicle width of 3750 mm, while exceeding the typical dimensions of standard passenger cars limited to 2.55 meters, is considered realistic and representative.

  These outlier values are likely associated with specific vehicle types, potentially specialized or industrial vehicles, which can have larger dimensions beyond standard passenger cars. Given the size of the dataset and the limited number of such extreme values, we consider the data as realistic and representative of the diversity in vehicle dimensions within different emission categories. Despite the standard limitation for passenger cars, the presence of larger vehicles is expected, and the dataset's overall integrity is maintained.

Distribution of vehicle weight ('m_kg') by CO^2-etiquette

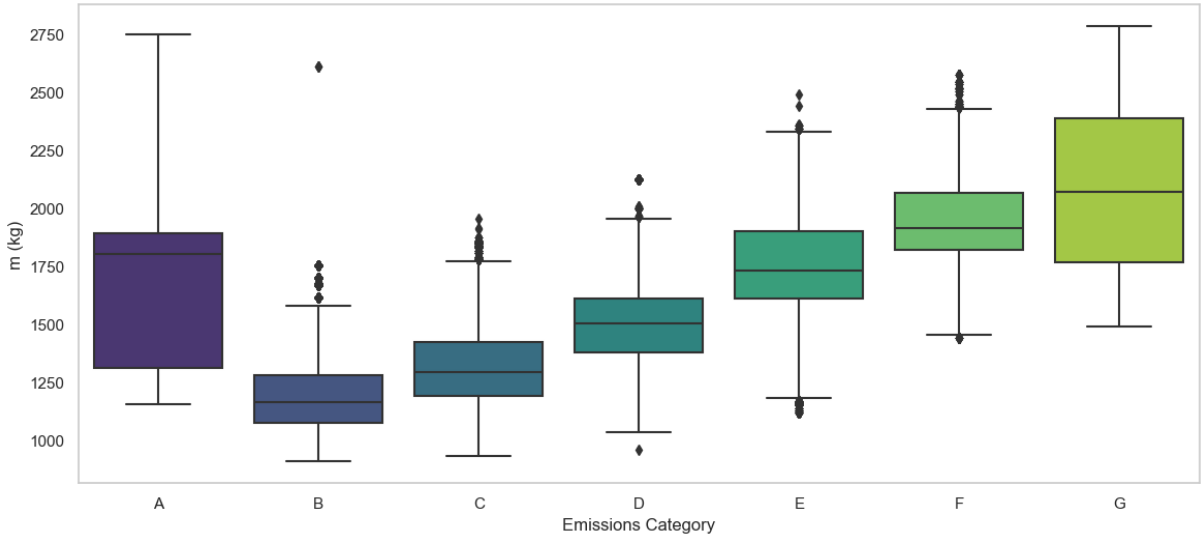


Figure 28. Boxplot of vehicle weights by CO^2-etiquette

Continuing from the analysis of vehicle widths, the examination of vehicle weights ('m_kg') within different emission categories provides further insights. The boxplots illustrate distinct characteristics among the categories.

- **Box Extent (Interquartile Range**: Building on the discussion of vehicle width, the patterns in vehicle weights echo some of the trends observed in width distributions. Categories *'C'*, *'D'*, *'E'*, *'F'*, and *'G'* exhibit broader interquartile ranges, indicating a wider variety of vehicle weights within these emission categories. The transition from vehicle width to weight seamlessly underscores the interconnected nature of these attributes.
- **Line inside the Box (Median)**: Surprisingly, Category *'A'* stands out with a high median weight, suggesting that vehicles in this category tend to be heavier on average. This unexpected finding prompts a closer examination of the data, emphasizing the importance of considering multiple vehicle attributes for a comprehensive analysis.
- **Whiskers (Minimum and Maximum):** When comparing the whiskers, Categories *'D'*, *'E'*, *'F'*, and *'G'* present longer spans, pointing to a more extensive range of vehicle weights. On the other hand, Category *'A'* exhibits extended whiskers, emphasizing the variability in weights within this particular emission category.
- **Evaluation of Outliers:** Outliers, denoted by individual data points beyond the whiskers, are observed across several categories, notably in *'A'* and *'F'*. These outliers highlight extreme values and reinforce the understanding that emission categories encompass a diverse range of vehicle weights.

  Addressing the concern of outliers, the reported maximum weight of 2751 kg in Category *'A'* is considered realistic. Such outliers likely represent specific vehicle types designed for specialized purposes, illustrating the importance of acknowledging and interpreting extreme values within a broader context. Despite these outliers, the dataset remains representative, capturing the diverse spectrum of weights within different emission categories while maintaining overall integrity.

Distribution of engine capacities ('ec_cm3') by CO^2-etiquette

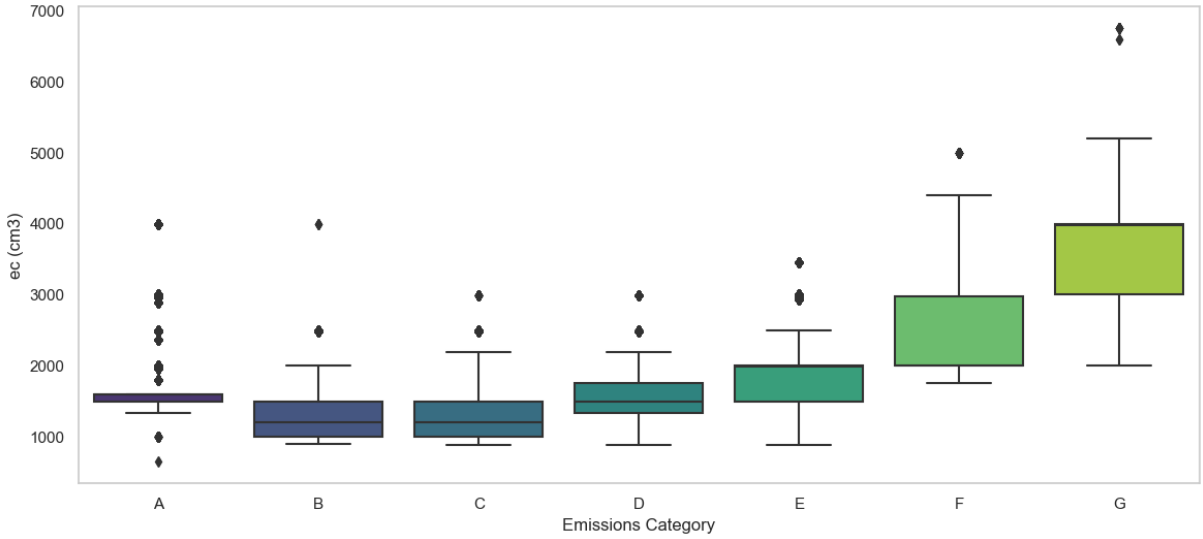


Figure 29. Boxplot of engine capacities by CO^2-etiquette

Continuing the exploration, delving into engine capacities ('ec_cm3') within distinct CO^2 emission categories unveils nuanced characteristics depicted through the boxplots.

- **Box Extent (Interquartile Range):** Echoing the analysis of vehicle width and weight, the interquartile range in engine capacities highlights distinct patterns. Categories 'C,' 'D,' 'E,' 'F,' and 'G' present broader interquartile ranges, signifying a diverse spectrum of engine capacities within these emission categories.
- **Line Inside the Box (Median):** A noteworthy revelation surfaces as Category 'A' exhibits a higher median engine capacity, signaling that, on average, vehicles in this category tend to house larger engines. This unexpected finding beckons a closer examination of the data, emphasizing the intricate interplay among various vehicle attributes.
- **Whiskers (Minimum and Maximum):** Comparing whisker lengths, Categories 'D,' 'E,' 'F,' and 'G' unveil extended spans, suggesting a wider range of engine capacities. In contrast, Category 'A' reveals elongated whiskers, underscoring the variability in engine capacities within this particular emission category.
- **Evaluation of Outliers:** Outliers, marked by individual data points beyond the whiskers, manifest across multiple categories, notably in 'A' and 'F.' These outliers underscore extreme values, accentuating the diverse engine capacities encapsulated within different emission categories.

  Addressing concerns about outliers, it's pivotal to acknowledge that the reported maximum engine capacity of 3996 cm^3 in Category 'A' aligns with realism. These outliers likely represent specialized vehicle types tailored for specific purposes, underscoring the need to interpret extreme values within a broader context. Despite the outlier presence, the dataset maintains its representativeness, capturing the expansive spectrum of engine capacities within different emission categories while upholding overall data integrity.

Distribution of engine power ('ep_kW') by CO^2-etiquette

Transitioning from our examination of vehicle weights, the focus now shifts to the power output (ep (KW)), dissected by CO^2 emission classes. The boxplot serves as our visual guide, unraveling the nuances in power distribution across different emission categories.

- **Box Extent (Interquartile Range):** As we delve into the box heights, Categories *'F'* and *'G'* emerge with towering boxes, signaling a spectrum of power outputs within these emission realms. Conversely, *'A'*, *'B'*, *'C'*, and *'D'* showcase more compact boxes, indicating a more consolidated distribution of power.
- **Line inside the Box (Median):** Plotting the line within each box unveils intriguing findings. Surprisingly, Category *'G'* claims the spotlight with the highest median power output, hinting at a fleet of vehicles with a penchant for higher average power. Meanwhile, *'A'*, *'B'*, and *'C'* assume lower median values, suggesting a relatively modest average power spectrum.
- **Whiskers (Minimum and Maximum):** The whiskers, stretching to encompass the minimum and maximum values, illuminate fascinating trends. Noteworthy is the elongated span of Category *'G'*, embodying the broadest range of power outputs. In contrast, *'A'*, *'B'*, and *'C'* unveil shorter whiskers, indicative of a narrower spectrum in power within these emission categories.
- **Evaluation of Outliers:** The outliers, stationed beyond the whiskers, become protagonists in our narrative. Categories *'A'*, *'B'*, *'F'*, and *'G'* share the stage with conspicuous outliers, spotlighting instances of extreme power values within these emission categories.

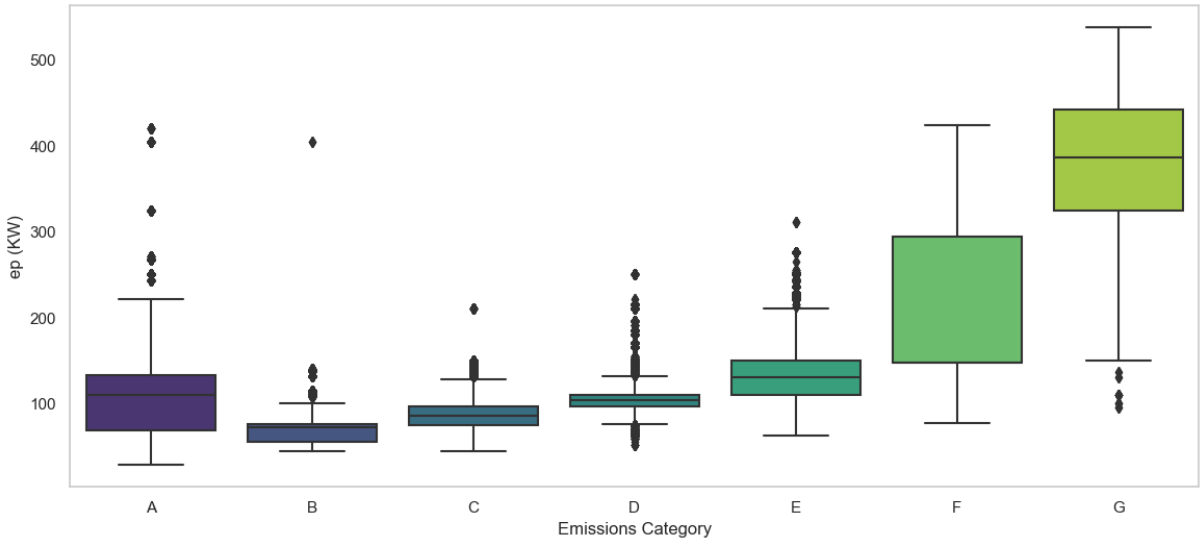


Figure 30. Boxplot of Engine Power by CO^2-etiquette

In summation, the exploration into power outputs uncovers a tapestry of patterns across diverse emission classes. *'G'* and *'F'* take center stage with elevated median values and expansive variability, while *'A'*, *'B'*, *'C'*, and *'D'* portray a more clustered power landscape. These revelations enrich our understanding of the multifaceted characteristics intertwined with varying CO^2 emission classes.

Summary

The exploratory data analysis (EDA) revealed several noteworthy insights within the dataset. Notably, the presence of significant outliers was identified, and distributions and ranges varied substantially based on factors such as fuel type, fuel mode, vehicle type, manufacturer, and CO^2 labeling. Correlation analyses provided a comprehensive understanding of relationships between variables.

Non-trivial correlations were observed, such as the negative correlation between vehicle weight and CO^2 emissions per kilometer. Additionally, intricate relationships between various metrics, including expected CO^2 emissions, fuel consumption, and efficiency, were uncovered. These findings shed light on the complex interplay of factors influencing environmental impact and vehicle characteristics.

The dataset displayed diverse patterns, and correlations were not confined to linear associations. For instance, electric consumption showed a negative correlation with CO^2 emissions, indicating that electric vehicles with lower emissions tended to have higher electric consumption. The correlation matrix and pairplots facilitated a visual understanding of these relationships.

Distinctive correlations were identified in specific domains, such as vehicle weight exhibiting positive correlations with width, engine capacity, and power. These associations contribute to a more nuanced understanding of the dataset, enhancing the significance of findings in the context of both vehicle characteristics and environmental impact.

In summary, the EDA not only highlighted the existence of outliers and variations in distributions but also uncovered intricate relationships and patterns within the dataset. These insights contribute to a more holistic understanding of the factors influencing CO^2 emissions and environmental sustainability in the automotive sector.

Pre-processing for Machine Learning

After conducting a comprehensive exploratory data analysis (EDA), we have gained a detailed understanding of the structure and characteristics of the dataset at hand. These insights serve as the foundation for transitioning into the Machine Learning (ML) and Deep Learning (DL) phases, where our focus shifts towards making more in-depth predictions regarding CO^2 emissions and categorizing vehicles into specific energy classes.

The EDA has allowed us to identify key parameters and relationships within the data. From the distribution of emission values to potential outliers and relevant correlations, the EDA has provided a clear understanding of the underlying structures. These insights not only inform the model-building process but also guide the formulation of targeted questions for the subsequent Machine Learning and Deep Learning phases.

In the upcoming section, our attention will be directed towards the application of supervised and deep neural networks. Through Machine Learning, we will classify vehicles into energy classes based on various features, enabling a fundamental structuring and segmentation of the dataset. Simultaneously, Deep Learning will be employed to make precise predictions regarding CO^2 emissions and unveil latent patterns within the complex dataset.

This seamless integration of insights from exploratory data analysis into the ensuing Machine Learning and Deep Learning phases allows us to harness the full potential of the data and contribute to a holistic solution for the challenges in the realm of CO^2 emissions analysis from vehicles.

Implementing CO^2-Emissions Categorization

In our upcoming phase, we decided to implement the categorization of CO^2-Emissions into classes (A to G) as a strategic feature engineering choice. The specific classes are described in the next figure..

Our decision is anchored in several considerations:

1. **Optimizing Model Performance:** We are gearing our Machine Learning models for classification tasks, and categorizing CO^2 emissions aligns with our objective of optimizing model performance. This step is expected to enhance the effectiveness of the training process, particularly for classification algorithms.

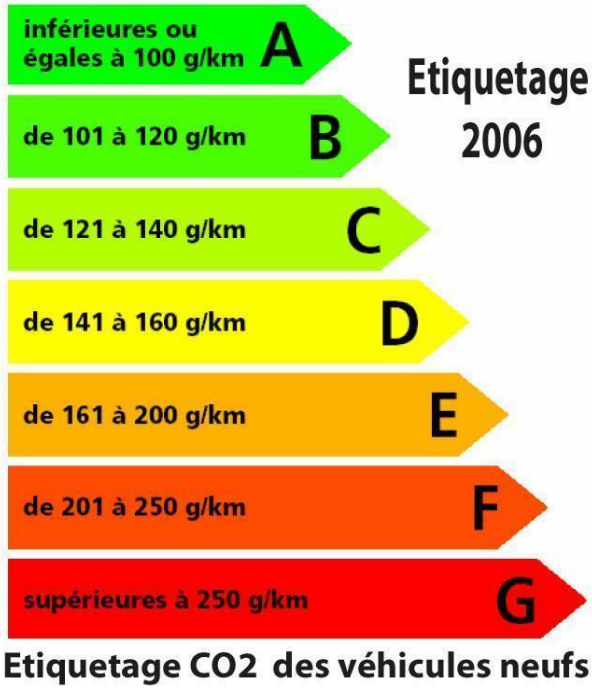
2. **User-Centric Interpretation:** From our perspective, user comprehension is at the forefront. Categorizing emissions into classes aims to deliver outputs that are intuitively understandable for stakeholders and end-users. This user-centric approach ensures clearer communication regarding the environmental impact of vehicles.

3. **Ensuring Regulatory Compliance:** Regulatory adherence is a pivotal factor guiding our decision. By categorizing emissions, we ensure that our model output is directly aligned with regulatory requirements and industry standards. This step is critical for maintaining compliance and meeting legal expectations.

4. **Building Robust Models:** Recognizing the potential for noise in continuous emissions data, our strategy includes categorization to bolster model robustness. By doing so, we mitigate the impact of minor fluctuations in emissions, resulting in a more resilient and reliable Machine Learning model.

5. **Flexibility for Diverse Stakeholder Needs:** Our approach prioritizes flexibility. The categorization of emissions allows us to tailor our model to the varied needs of stakeholders. This adaptability ensures that our model can be effectively applied in diverse contexts.

6. **Facilitating Feature Importance Analysis:** We are committed to providing a comprehensive understanding. The categorization of emissions will simplify the analysis of feature importance, enabling us to identify the key features that significantly contribute to a vehicle falling into a specific emission class.



7. **Alignment with Industry Standards:** Our decision aligns with industry best practices. Categorizing CO^2 emissions into classes ensures that our model output resonates with widely accepted metrics within the automotive sector, enhancing the credibility and applicability of our results.

To summarize by implementing CO^2 emissions categorization, we anticipate clearer insights into the environmental impact of vehicles. The user-friendly outputs, regulatory compliance, and robustness introduced through this approach will contribute to a more informed and adaptable model, ultimately serving the diverse needs of stakeholders and supporting effective decision-making in the automotive domain.

Use of One-Hot-Encoding

In preparation for applying Machine Learning and Deep Learning techniques, we utilized the 'pd.get_dummies' function for One-Hot Encoding. This process was necessary due to the presence of categorical variables in the dataset. Let's delve into a more detailed explanation of One-Hot Encoding and why it was crucial for our data.

One-Hot Encoding:

- **Definition:** One-Hot Encoding is a technique used to convert categorical variables into a binary matrix, where each category is represented by a binary value (0 or 1). This results in a new set of binary columns, each corresponding to a unique category in the original categorical variable.

Why One-Hot Encoding Was Necessary for Our Data:

1. **Algorithm Compatibility:** Many Machine Learning algorithms, including those used in classification tasks, require numerical input. Categorical variables, such as Fuel Type ('Ft'), Fuel Mode (`Fm`), Vehicle Category (`Cr`) and `Manufacturer (`Mp`) in our dataset, are non-

numeric. One-Hot Encoding transforms these categorical variables into a format that can be interpreted by algorithms, allowing them to learn from these features effectively.

2. **Preventing Misinterpretation:** Without One-Hot Encoding, a Machine Learning model might misinterpret the numerical representation of categorical variables as ordinal, assuming a meaningful order that may not exist. One-Hot Encoding eliminates this risk by providing a clear binary representation for each category.
3. **Avoiding Bias:** When dealing with categorical variables with multiple categories, assigning arbitrary numerical labels could introduce bias. For instance, assigning numbers 1, 2, and 3 to different fuel types might imply a certain order or hierarchy. One-Hot Encoding avoids this issue by creating separate binary columns, ensuring equal importance for each category.
4. **Maintaining Model Performance:** One-Hot Encoding allows the model to capture the distinctiveness of each category without imposing artificial ordinal relationships. This is particularly important when dealing with features like "fuel type" or "manufacturer specifications," where categories are typically independent and don't have inherent order.

In summary, One-Hot Encoding was necessary for our data to ensure that categorical variables were appropriately represented in a format compatible with Machine Learning algorithms. It prevents misinterpretation, bias, and ensures that the models can effectively learn from and leverage the information encoded in these categorical features during training.

Standardization of numerical features

In preparation for applying Machine Learning and Deep Learning techniques, we also employed the 'StandardScaler' for standardizing numerical features within the dataset. Let's explore in more detail what 'StandardScaler' is and why it was crucial for our data.

StandardScaler:

- **Definition:** 'StandardScaler' is a preprocessing technique that standardizes the features by transforming them to have a mean of 0 and a standard deviation of 1. It effectively scales the data, making it more suitable for certain Machine Learning algorithms.

In the exploratory data analysis phase, we identified outliers and extreme values within the dataset, as well as observed variations in the ranges of different numerical variables. These findings prompted the application of the 'StandardScaler' as a crucial preprocessing step in preparation for Machine Learning and Deep Learning techniques.

Outliers and Extreme Values:

- **Identification:** Outliers were identified as data points that deviated significantly from the overall distribution of a variable. These values could potentially introduce noise and impact the performance of Machine Learning models.
- **Reasoning:** Outliers can disproportionately influence algorithms, particularly those sensitive to extreme values, potentially leading to biased predictions. Standardizing the features helps mitigate the impact of outliers by scaling the data based on mean and standard deviation, making the algorithm more robust.

Different Variable Ranges:

- **Observation:** Numerical variables in the dataset exhibited varying scales and magnitudes. Some features had values on a larger scale compared to others.

- **Reasoning:** Machine Learning algorithms, especially those relying on distance metrics or gradient-based optimization, are sensitive to the scale of input features. Features with larger scales might dominate the learning process, leading to suboptimal performance.

'StandardScaler' in addressing these challenges:

- **Definition of 'StandardScaler':** 'StandardScaler' is a preprocessing technique that standardizes numerical features by transforming them to have a mean of 0 and a standard deviation of 1.
- **Addressing Scale Sensitivity:** Machine Learning algorithms such as support vector machines, k-nearest neighbors, and gradient descent-based optimization methods are sensitive to the scale of input features. 'StandardScaler' ensures that all features contribute equally, preventing undue influence from features with larger scales.
- **Equal Contribution of Features:** Standardization guarantees that each feature contributes proportionally to the learning process, avoiding biases that may arise from different feature magnitudes.
- **Enhancing Convergence:** Algorithms relying on distance metrics benefit from feature standardization, as it improves convergence during optimization by providing a balanced contribution from each feature.
- **Avoiding Numerical Instabilities:** 'StandardScaler' helps prevent numerical instabilities that may occur when dealing with features of vastly different scales, ensuring stable and reliable model performance.
- **Facilitating Model Interpretation:** Standardizing features simplifies the interpretation of model coefficients or weights, as they are on a standardized scale. This aids in comparing the importance of different features.

Summary: In conclusion, the application of 'StandardScaler' was essential for our data to address challenges related to outliers, extreme values, and differing variable scales. It promotes a more stable and robust learning process for Machine Learning algorithms, ensuring fair contributions from all features and facilitating model interpretability.

Regression

The objective of this project is to examine the impact of various car properties on CO^2emissions. In this chapter, our goal is to develop a regression model that demonstrates a strong goodness of fit. This will enable us to effectively explain the influence of the available information about the cars on the target variable, CO^2emissions.

To achieve this, we will explore and compare different regression algorithms, such as Decision Tree Regression, Random Forest Regression, XGBoost Regression, and a Deep Neural Network. Each algorithm will be introduced, explained, and the reasons for including it in the regression task will be provided. We will also explain the implementation of the algorithms and present the results.

Following that, we will compare the models to select the best performing one and enhance its performance by fine-tuning the hyperparameters. This final model will be utilized for model interpretation in the subsequent chapter.

Decision Tree

The Decision Tree Regressor is a supervised learning algorithm which allows us to model the relationship between input features and a numerical target variable using a tree structure.

We chose Decision Tree Regression for our project because the algorithm offers several advantages that align with our dataset and goals. Firstly, it presents a white-box model, providing transparent and logical explanations for decisions. Therefore, Decision Trees are intuitive and easily interpretable, enabling us to visualize and understand how the properties of cars affect CO^2 emissions, our primary goal. Decision trees perform implicit feature selection by identifying the most informative features for predicting the target variable. This capability makes it straightforward to determine which features have the most significant impact on CO^2 emissions. Additionally, Decision Trees can handle both numerical and categorical data. As a result, there's no need to drop categorical features or perform additional preprocessing on our data. Decision Trees capture non-linear relationships between features and the target variable, making them suitable for modeling complex, non-linear patterns in the data. This capability is relevant to our project since we have a relatively high number of parameters, many of which do not have linear relationships with our target. Another notable advantage is the efficient computational cost of Decision Trees; predicting outcomes using a Decision Tree scales logarithmically with dataset size. This efficiency is crucial since our dataset comprises 1.8 million observations.

However, Decision Trees come with limitations that we must acknowledge. First, they are prone to overfitting, especially when allowed to grow deep and complex, capturing noise from the training data. Second, the algorithm is unstable; thus, small variations in the training data can result in significant changes to the tree's structure, which makes the interpretation of the model less generalizable. Lastly, while Decision Trees are simple to interpret, they might not capture complex relationships as effectively as more advanced models, potentially leading to less accurate predictions compared to other methods.

Decision Tree Regression Algorithm

The Decision Tree algorithm for regression tasks works by recursively partitioning the feature space and fitting a basic model within each partition. The algorithm starts with the entire dataset and looks for the best feature and split point that minimizes the variance of the target variable within the resulting partitions. This split is determined based on a chosen criterion, in our case minimizing the Mean Squared Error. Once the initial split is made, the algorithm repeats the splitting process on each resulting partition. This creates a tree structure where each internal node represents a split based on a feature, and each leaf node represents a predicted value. The algorithm stops partitioning when certain conditions are met, such as reaching a set tree depth, having too few samples in a node, or not achieving a sufficient variance reduction. For new data, the algorithm traverses the tree based on the feature values to determine the associated leaf node. The prediction within a leaf is the mean of target values for samples in that node. This approach results in a tree that partitions the feature space, fitting simple models within each section for regression predictions.

Implementation of Decision Tree Regression

Initially, we utilized the preprocessed data, incorporating all the available features, for our regression models. However, upon analysis, we observed that both the Decision Tree and subsequent algorithms primarily relied on Fuel Consumption ('Fc') and Electric Range ('Er_km') to make predictions. These variables show a strong correlation with our target variable, CO^2 Emission. The models that incorporated these variables demonstrated high accuracy, as evidenced by the test scores. When incorporating Fuel Consumption and Electric Range, the Decision Tree model achieves test scores of RMSE = 0.895 and R^2 = 0.991. In contrast, without including these variables, the model yields test scores of RMSE = 1.667 and R^2 = 0.997. However, these results did not provide satisfactory insights into specific car properties that significantly influence CO^2 emissions. This limitation is clearly illustrated in the accompanying graph.

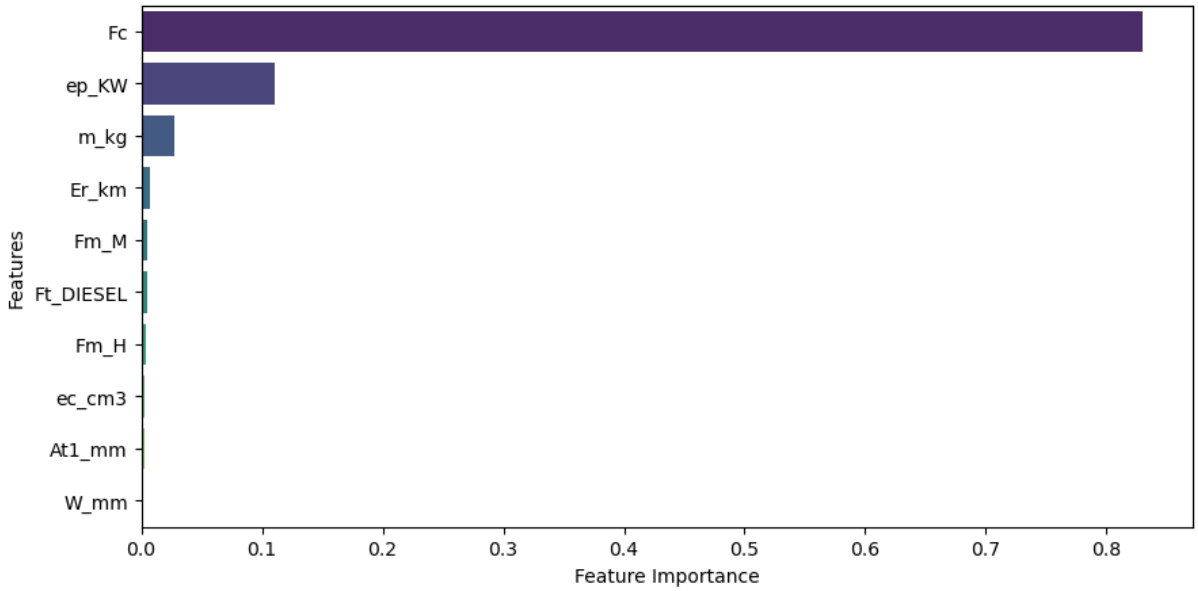


Figure 32. Feature Importance with fuel consumption and electric range

The graph shows the feature importances of the Decision Tree Regression model, which includes Fuel Consumption and Electric Range in kilometers. Feature importance quantifies the extent of improvement achieved by splitting the data based on a specific feature. Each feature's importance is calculated during the construction of the model and subsequently normalized to ensure a sum of 1 at the end of the training process. In the context of Decision Tree Regression, feature importance is calculated as the normalized total reduction in the Mean Squared Error brought about by a particular feature across all nodes. The measures are accessible through the 'feature_importances_' attribute in the sklearn's models. The graph reveals that Fuel Consumption accounts for 83% of the reduction in Mean Squared Error, signifying its dominant influence on the prediction of CO^2 emission. The three most significant features collectively contribute to 96.7% of the overall feature importance. Consequently, this model fails to provide valuable insights for our project, as the remaining 27 features accumulate to a feature importance of 3.3%, rendering them insignificant compared to the three prominent features. Similar results are observed across the algorithms used in subsequent chapters. Therefore, we have decided to exclude the Fuel Consumption and Electric Range variables from all classification and regression models.

We continued by removing the Fuel Consumption and Electric Range variables from our preprocessed dataset. The remaining 28 features, which describe various properties of the cars, were stored in a variable called "data". Our target variable is the WLTP score, which measures a vehicle's CO^2 emission in grams per kilometer. This score is measured under real-world driving conditions and follows the standardized Worldwide Harmonized Light Vehicles Test Procedure (WLTP). Next, we split the data into a training set and a testing set using the 'train_test_split' function from the 'sklearn.model_selection' library. We allocated 20% of the data for the test set and set the 'random_state' parameter to 123 to ensure reproducibility of the code. Since Decision Trees are non-parametric algorithms, they don't rely on assumptions about the distribution, independence, or variance of the training data and residuals. Hence, we can apply the Decision Tree algorithm to our data without the need to test any assumptions first. Additionally, Decision Trees are not affected by features of different scales, so there is no requirement for additional preprocessing steps such as standardization or normalization of the features.

We utilized the 'DecisionTreeRegressor' class from the 'tree' package in 'sklearn' to implement our Decision Tree model. As part of our general approach, we initially created a baseline model as a point

of comparison for the four chosen algorithms. This allows us to evaluate the performance of each algorithm before proceeding with hyperparameter tuning for the best-performing one. Therefore we used the following default hyperparameters for our instance of 'DecisionTreeRegressor': The criterion parameter was set to *'squared_error'*, which minimizes the squared differences between predicted and actual values. The splitter parameter was set to *'best'*, indicating that the best split at each node is chosen based on the selected criterion. The 'max_depth' parameter was set to *'None'*, allowing nodes to expand until all leaves are pure or contain fewer samples than min_samples_split. Other parameters such as min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, random_state, max_leaf_nodes, min_impurity_decrease, and ccp_alpha were also kept at their default values.

The concrete implementation of our Decision Tree is shown in the following. We used the same logic for the implementation of the upcoming regression algorithms.

```python
# Train a Decision Tree Regression
if train_models:
  # Train model
  dt_model_wo_fc = DecisionTreeRegressor(random_state=123)
  dt_model_wo_fc.fit(X_train_wo_fc, y_train_wo_fc)
  # Save model
  dump(dt_model_wo_fc, 'models/dt_model_wo_fc.joblib')

else:
  print("Only load model. Change variable train_models to True in chapter 1.0 if you want to retrain model.")

# Load model
dt_model_wo_fc = load('models/dt_model_wo_fc.joblib')
```

Python

First, we initialized a 'DecisionTreeRegressor' and stored it in the variable 'dt_model_wo_fc'. The '_wo_fc' suffix indicates that the data used doesn't include Fuel Consumption and Electric Range. We only needed to specify the 'random_state' parameter during the initialization, as the other parameters were left at their default values. Next, we trained the model using the training data 'X_train_wo_fc' and 'y_train_wo_fc'. To make our code more efficient, we added an if-else condition based on the Boolean variable 'train_models'. This variable is initialized at the beginning of the script. If we set 'train_models' to True, the Decision Tree model is initialized (line 4) and trained (line 5) as described above. To save time and computing resources, we added the 'dump' function from the 'joblib' library in line 7 to save the trained model. If we only want to load a pretrained model, we set 'train_models' to False. In this case, lines 3 to 7 are skipped, and the pretrained model is loaded in line 13.

Results

To simplify the process of calculating evaluation metrics for our regression models, we created a function called 'get_evaluation_table'. This function abstracts away the repetitive task of computing metrics such as RMSE, MSE, MAE, and R2 score.

```python
def get_evaluation_table(model,feat_train, feat_test,target_train, target_test):

  # Evalutaion on Training Set
  y_train_pred = model.predict(feat_train)
  train_rmse = mean_squared_error(target_train, y_train_pred, squared=False)
  train_mse = mean_squared_error(target_train, y_train_pred)
  train_mae = mean_absolute_error(target_train, y_train_pred)
  train_r2 = r2_score(target_train, y_train_pred)

  # Evalutaion on Test Set
  y_test_pred = model.predict(feat_test)
  test_rmse = mean_squared_error(target_test, y_test_pred, squared=False)
  test_mse = mean_squared_error(target_test, y_test_pred)
  test_mae = mean_absolute_error(target_test, y_test_pred)
  test_r2 = r2_score(target_test, y_test_pred)

  # Create table
  results = pd.DataFrame(
    index = ['MSE', 'RMSE', 'MAE', 'R2 Score'],
    data={'Training Set': [train_mse, train_rmse, train_mae, train_r2],
          'Test Set': [test_mse,test_rmse, test_mae, test_r2],
          'Delta': [test_mse-train_mse, test_rmse-train_rmse, test_mae-train_mae, test_r2-train_r2]})

  return results
```

Python

The 'get_evaluation_table' function is designed to evaluate the performance of a given model on both training and test set. It takes the trained model, as well as the train and test split as an argument and returns a dataframe that contains their evaluation metrics. First, the function uses the 'predict' method of the model to get the predicted outcomes based on the features provided in the X_train (line 4) and X_test (line 11) array. Next, it calculates four key metrics for each dataset using the respective functions from the 'sklearn.metrics' library: Root Mean Squared Error (RMSE), Mean Squared Error (MSE), Mean Absolute Error (MAE), and R^2 score. To calculate the Root Mean Squared Error (RMSE), we needed to set the squared parameter of the 'mean_squared_error' function to False. These metrics provide insights into the model's prediction errors and the proportion of variance explained by the model. After computing the metrics, the function organizes the results into a structured table using the Pandas 'DataFrame' function. Each metric corresponds to a row, and the columns represent the evaluations on the training and test sets. Additionally, we have included a row named "Delta," which displays the difference between the test and train scores. This allows for easier analysis of potential overfitting in the model. Finally, the function returns this table, which can be further analyzed or used for reporting purposes. While each metric provides valuable insights into the model's efficacy, it's essential to consider their respective strengths and limitations when assessing overall model performance. In the following, we will explain the used measures and report the results of the Decision Tree Regression.

The table below shows the results of the performance metrics of a Decision Tree Regression model on both the training and test datasets:

	MSE	**RMSE**	**MAE**	**R^2**
Training Set	2.627	1.621	0.930	0.997
Test Set	2.779	1.667	0.938	0.997
Delta	0.152	0.046	0.008	-0.000

The Mean Squared Error (MSE) provides a measure of the average squared deviation between the predicted ($\hat{y}_i$) and actual (y_i) outcomes:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

The MSE gives a high weight to large errors. This ensures, that significant errors aren't ignored, but it also makes the measure sensitive to outliers. For the training set, the model achieves a MSE of 2.627, while for the test set, the MSE is slightly higher at 2.779. However, the difference is very small, indicating that overfitting is not a concern.

The Root Mean Squared Error (RMSE) builds upon the MSE by taking the square root of the average squared deviations.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2}$$

It serves as a more comprehensive metric for quantifying the average prediction error, as the results are in the same scale as the target variable. Just like the MSE, the RMSE is sensitive to outliers. The model achieves an RMSE of 1.621 for the training set, slightly outperforming its performance on the test set, where the RMSE is 1.667. Considering that the range of our target variable is 369, with a minimum value of 4 and a maximum value of 373, the RMSE can be considered low. This indicates that the model's predictions are generally close to the actual values and that it is performing well in terms of minimizing the overall prediction error.

Moving to the Mean Absolute Error (MAE), this metric evaluates the average absolute discrepancies between predictions and observed values, with less emphasis on extreme outliers compared to RMSE and MSE. This makes the MAE more robust against outliers.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}|$$

The model has a MAE of 0.930 on the training set, which incrementally increases to 0.938 on the test set. The difference in MAE between the training and test sets is 0.008. This is another indication that overfitting is not a concern for this model. The MAE is bigger than the RMSE, which indicates that the model made a few relatively large errors.

Lastly, the R^2 score measures the extent to which the model's independent variables explain the variance observed in the dependent variable. The R^2 formula calculates the proportion of the variance in the dependent variable explained by the independent variables:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Here, SS_{res} is the sum of squared differences between observed (y_i) and predicted values ($\hat{y_i}$), representing the unexplained variance. Conversely, $SS_{tot}$is the sum of squared differences between observed values (y_i) and their mean ($\bar{y}$), capturing the total variance. A higher R^2 value (closer to 1) indicates that a greater proportion of the variance in the dependent variable is accounted for by the model's independent variables. Therefor it offers an intuitive measure of goodness-of-fit. The model shows a robust R^2 score of 0.997 on the training and on the test set.

In summary, the Decision Tree Regression model demonstrates a good performance, as evidenced by the evaluation metrics. The marginal differences between training and test set metrics, coupled with

high R^2 values, validate the model's reliability and effectiveness in predicting CO^2 emissions based on car properties.

Random Forest Regression

Random Forest Regression is an Ensemble Learning method that combines multiple Decision Tree Regressors to improve the accuracy and robustness of the predictions.

We chose Random Forest Regression because it offers several advantages compared to simpler regression methods like Decision Trees. Random Forests generally provide high prediction scores as they can capture complex relationships in the data by combining multiple Decision Trees. Unlike individual Decision Trees, Random Forests are less prone to overfitting, thanks to their ensemble approach and the utilization of random feature subsets. Consequently, we expect the Random Forest Regression to make better predictions than our Decision Tree Regression Model by further minimizing overfitting. Moreover, as a non-parametric algorithm, Random Forests are suitable for our dataset. They are also efficient in handling large datasets with high dimensionality and a substantial number of training examples. This capability is important, as our dataset has a high dimensionality with a shape of (1589748,28)[1]. Additionally, Random Forests offer a feature importance measure, enabling us to easily identify the car properties that significantly influence CO^2 emissions which is the goal of our project.

However, the Random Forest algorithm also has some drawbacks that should be considered. Firstly, the computational complexity of training many Decision Trees can be quite high, especially when dealing with very large datasets. The process of constructing multiple trees and combining their predictions can be time-consuming and resource intensive. Additionally, Random Forests can require a significant amount of memory, particularly when dealing with many trees and features. Each tree in the ensemble needs to be stored in memory, along with the associated feature subsets and split points. This memory requirement can become a limiting factor, especially on systems with limited resources. Furthermore, Random Forests have several hyperparameters that need to be tuned for optimal performance. Finding the optimal combination of hyperparameters in Random Forests can be a challenging task that demands extensive experimentation and tuning. This process can be time-consuming and resource-intensive, especially when dealing with a large dataset. Additionally, despite providing feature importance for model interpretability, the ensemble nature of Random Forests can make them less interpretable compared to simpler models like Decision Trees. This is because the Random Forest algorithm functions as a black box model, where the individual Decision Trees are combined to make predictions, making it more difficult to understand the underlying decision-making process of the model.

In summary, Random Forest Regression offers high accuracy and robustness for large and complex datasets. However, it requires significant computational resources, and we can’t visualize its decision-making process due to its black box nature.

[1] Excluding the Target, Electric Range and Fuel Consumption

Algorithm

Random Forest Regression is an Ensemble Learning method that combines multiple Decision Tree Regressors to improve the accuracy and robustness of the predictions. The algorithm of Random Forest, as described by Hastie, Tibshirani, Friedman & Friedman (2009)[2], is outlined below:

Algorithm for training the Random Forest

For $b = 1, \dots, B$ with B as number of Trees (n_estimators)

1 Draw a bootstrap sample Z^* of size N from the training data

2 Grow a Random Forest Tree T_b using the bootstrapped data by recursively performing the following steps for each terminal node of the tree until the minimum node size n_{min} is reached

2.1 Randomly select m variables from p available variables.

2.2 Determine the best variable/split-point among the selected m variables.

2.3 Split the node into two daughter nodes.

The algorithm starts by Bagging (Bootstrap Aggregating): For each of the desired number of trees B, it first creates a bootstrap sample Z^* of the same size N as the original training data by randomly sampling with replacement. The Bootstrapping ensures that each Decision Tree is trained on a slightly different dataset. This diversity increases the overall robustness of the algorithm and reduces the likelihood of overfitting. Then, a Decision Tree T_b is grown using this bootstrapped sample. During the construction of each tree, at every terminal node, a subset of m variables is randomly chosen from the total p variables available. The random features subsets reduce the correlation between the trees, which increases the diversity of trees which improves the overall performance of the model. Among these selected variables, the algorithm determines the best variable and its corresponding split point to divide the node into two child nodes. In our case we use the Mean Squared Error as the criterion for choosing the best split. This process continues recursively until a minimum node size n_{min} is attained, ensuring that the tree doesn't become overly complex or prone to overfitting. The algorithm returns an ensemble of trees: $\{T_b\}_{b=1}^{B}$. For regression tasks, the predictions for new data x is obtained from the Random Forest by averaging the predictions of all the individual trees.

$$\widehat{f_{rf}}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$

Implementation

The Random Forest was implemented using the 'RandomForestRegressor' class from the 'sklearn.ensemble' library. For the baseline model, we used the default parameters. Therefore the 'n_estimators' parameter is 100, determining the number of trees in the forest. The criterion for splitting decision nodes is *'squared_error'*, which means the splits within the trees were based on minimizing the Mean Squared Error. The 'max_depth' parameter is *'None'*, allowing the trees to expand until they captured the full depth of the data. The 'min_samples_split' and 'min_samples_leaf' parameters are 2 and 1, respectively, ensuring that nodes were split and leaves formed without imposing further restrictions. The 'max_features' parameter is 1.0, considering all features during the

[2] Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2, pp. 1-758). New York: springer.

split for comprehensive feature evaluation. The 'oob_score' parameter was not utilized, meaning that the model did not compute an out-of-bag estimate of prediction error. Other parameters such as 'max_leaf_nodes', 'min_impurity_decrease', 'ccp_alpha', 'n_jobs', 'verbose', 'warm_start', and 'max_samples' also retained their default settings. However, the 'random_state' parameter is specified to ensure reproducibility of the results.

We used the same train test split for the Random Forest as previously used for the Decision Tree Regression. This means that our target is the WLTP Scores, while the features did not include Electric Range and Fuel Consumption and the test set consisted of 20% of the data. Since the Random Forest algorithm does not rely on distances or make assumptions about feature distributions, there was no need for additional preprocessing steps like normalization or standardization. This is because the algorithm is capable of effectively handling features with different ranges and scales. We trained the Random Forest model on the training data using the 'fit' method of the 'RandomForestRegressor' class. As explained in detail in the previous chapter, we saved the trained model using the 'joblib' library to optimize computational resources. This allowed us to load the model later without retraining it, saving time and computing power.

Results

For the evaluation of the model, we used our 'get_evaluation_table' function, which is explained in detail in the previous chapter. The results are shown in the table below.

	MSE	RMSE	MAE	R^2
Training Set	2.639	1.625	0.933	0.997
Test Set	2.767	1.664	0.938	0.997
Delta	0.128	0.039	0.006	-0.000

The Random Forest model demonstrates good performance based on the provided evaluation metrics. Both the training and test sets exhibit low values for Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE), considering that the target has a range of 369. The higher values for RMSE indicate, that the model's predictions are consistently close to the actual values, with fewer extreme outliers affecting the RMSE metric. Additionally, the R^2 Score, which measures the goodness of fit, is remarkably high for both datasets, reaching 0.997. Notably, the small differences observed between the training and test sets across these metrics indicate that the model does not suffer from overfitting. This suggests that the model's performance is reliable and generalizes well on unseen data.

XGBoost Regression

The gradient boosting algorithm is an Ensemble Learning method that creates a robust predictive model by combining the predictions of multiple individual models, with each new model focusing on the errors made by the previous ones. XGBoost, an advanced and efficient implementation of this algorithm, has become widely popular for solving structured and tabular data problems.

We have chosen to use XGBoost for our project, due to its numerous advantages. Firstly, XGBoost is renowned for its superior performance, which often outperforms other algorithms in Machine Learning competitions. Therefore, we want to compare its performance against other models implemented in our project. Additionally, XGBoost is optimized for both memory efficiency and computational speed, making it well-suited for our relatively big dataset. Moreover, XGBoost

incorporates built-in L1 (Lasso Regression) and L2 (Ridge Regression) regularization to prevent overfitting, distinguishing it as a better choice compared to other Boosting Algorithms.

However, XGBoost does come with its drawbacks. Improper regularization or excessive training rounds can lead to overfitting of the training data. Like other boosting algorithms, XGBoost can be sensitive to noisy data and outliers, which is a concern as our dataset contains outliers. Although XGBoost is efficient compared to other boosting algorithms, training it on large datasets like ours can still be time-consuming and computationally demanding. Lastly, XGBoost offers a multitude of hyperparameters to fine-tune, and the interaction between them can pose challenges in finding the optimal settings, particularly given the time-consuming nature of model training.

Algorithm

The XGBoost algorithm begins by initializing predictions with a constant value for all instances in the dataset, typically the mean of the target variable. This initial prediction provides a baseline for subsequent model enhancements.

Subsequently, the algorithm enters an iterative process where it sequentially constructs new models, in our case Decision Trees. During each iteration, XGBoost computes the gradients and Hessians of a specified loss function concerning the predictions made by the existing ensemble of trees. The gradient of the loss function is a vector with its first-order partial derivatives that points in the direction, where the loss function increases most rapidly. The Hessian is a square matrix of second-order partial derivatives which helps identifying whether a critical point is a local maximum, local minimum, or saddle point. For our purposes, the Squared Error serves as the loss function. Using the gradients and Hessians as guidance, XGBoost trains a new tree to predict the negative gradients of the loss function. By predicting the negative gradients of the loss function, the new tree is moving into the direction, where the loss function decreases the most. In other words, it’s learning from the mistakes of the previous model. The tree's construction uses a Taylor expansion approximation aimed at minimizing the loss function.

XGBoost incorporates regularization directly into its objective function, distinguishing it from traditional gradient boosting algorithms. This regularization term penalizes overly complex models, reduces overfitting and therefore improves the algorithms’ ability to generalize to unseen data. After training each new tree, its predictions are scaled by a learning rate and integrated into the ensemble's predictions. This iterative procedure continues until reaching a predetermined number of trees or until further iterations fail to yield significant reductions in the loss.

Ultimately, the final prediction from XGBoost is derived by aggregating the predictions of all individual trees within the ensemble.

Implementation

For the XGBoost Regression, we utilized the 'XGBRegressor' from the 'xgboost' library. We used the default parameters of the 'XGBRegressor' class, because we want to compare baseline models, before continuing to hyperparameter tuning for the best model. Therefore the booster parameter is '*gbtree*', which signifies the use of Decision Trees as base learners. The 'n_estimators' parameter is 100, determining the number of boosting rounds or trees during training. With a 'max_depth' of 3, each Decision Tree in the ensemble was limited to a maximum depth of three levels, reducing overfitting and promoting better generalization to unseen data. The default learning rate is 0.1, which controls the contribution of each tree to the overall ensemble. The numerous other hyperparameters were also kept at their default values, expect to the random state, which we initialized to make the code reproducible.

We conducted the training process using the 'fit' method from the 'XGBRegressor' class on the training set, using the same train-test split as the previous models. Since XGBoost does not rely on distances, no additional preprocessing steps were necessary. To save the trained model, we utilized the 'dump' function from the 'joblib' library, as explained in detail in the previous chapters.

Results

We evaluated the model using our 'get_evaluation_table' function. The following table shows the evaluation metrics of the XGBoost

	MSE	RMSE	MAE	R^2
Training Set	4.197	2.049	1.289	0.995
Test Set	4.305	2.075	1.290	0.995
Delta	0.108	0.026	0.001	-0.000

The model exhibits good performance, with high R^2 scores of 0.995 on both training and test set, indicating a robust explanatory power. The RMSE and MAE are very low, considering the range of the target variable, which is 369. The RMSE is bigger than the MAE, which indicates that some of the predictions were significantly different from the actual values. While the MSE and RMSE show slightly increased values in the test set compared to the training set, the differences are marginal, suggesting that the model generalizes well to unseen data. Overall, the results suggest that the XGBoost regression model performs effectively and reliably across training and test datasets. However, it should be noted that despite the good performance, the results look worse compared to the previous models.

Deep Learning

A Deep Neural Network (DNN) is a type of artificial neural network that consists of multiple layers of interconnected nodes, enabling it to learn and represent complex patterns in data.

Deep Neural Networks (DNNs) have numerous advantages and disadvantages for regression tasks. Firstly, they excel at modeling complex patterns and capturing intricate non-linear relationships that simpler models struggle to represent. Therefore, we have chosen to utilize DNNs for our project to assess their performance against other algorithms. Secondly, DNNs can handle large tabular dataset, making them well-suited for our project. Finally, DNNs can automatically learn and extract relevant features from raw data. This is beneficial to us, as we want to understand how the car properties effect the target variable.

Like the other algorithms, DNNs have several drawbacks that can impact their practicality and effectiveness. First and foremost, training DNNs demands substantial computational resources, often requiring powerful hardware and extensive training times. Additionally, DNNs possess numerous hyperparameters that should be tuned, such as layer count, neuron count, and learning rate. This tuning process can be time-consuming, especially because of the lengthy training times. Consequently, our baseline model might not outperform other algorithms if the hyperparameters are not appropriately configured. Another limitation lies in the black box nature of DNNs, making it challenging to interpret the inner workings of the model. However, we can utilize libraries like 'skater' and 'shap' to interpret the results and gain insights into how the car properties affect CO^2 emissions. Lastly, overfitting is a common issue with DNNs, particularly when the model is overly complex or when the training data is limited. Although we have sufficient training data, there is a possibility of overfitting if the hyperparameters are not tuned correctly.

Algorithm

Deep Neural Networks (DNNs) work by processing input data through multiple layers of interconnected neurons to learn and extract complex representations of the input. The input data is fed into the input layer of the neural network. Each feature of the input data is represented by a node in the input layer. DNNs consist of one or more hidden layers, each containing multiple neurons. These hidden layers are responsible for learning and extracting features from the input data. Each neuron in the hidden layers performs a weighted sum of the inputs from the previous layer, followed by the application of an activation function. The weighted sum z in a neuron, incorporating the weights w_i, inputsx_i, and bias b ,is calculated as:

$$z = \sum_{i=1}^{n} (w_i \times x_i) + b$$

In our case we use the ReLU as an activation function, which returns 0 if the input is negative and otherwise the input value:

$$f(z) = (0, z)$$

The ReLu activation function introduces nonlinearity into the network, allowing it to learn complex patterns while being more computational efficient compared to other activation functions. The connections between neurons are characterized by the weights and biases, which are adjusted during the training process to minimize the difference between the predicted and actual outputs. These adjustments are made using an optimization algorithm, in our case we use the Adam optimizer. Adam is an adaptive learning rate optimization algorithm that combines ideas from RMSProp (Root Mean Square Propagation) and Momentum, making it efficient and widely used in practice. The final hidden layer feeds into the output layer, which produces the network's predictions or outputs. The number of neurons in the output layer depends on the specific task, in our case we have one neuron, as we want to predict the CO^2 emission. We use the linear activation function for our output layer, which takes the weightes sum of the inputs z (see formular above) and passes it directly as an output:

$$f(z) = z$$

The linear activation function is commonly used for regression tasks, because it does not restrict the range of output layers. This enables the prediction of a complete spectrum of continuous numbers, which aligns with the typical requirements in regression tasks.

Once the DNN is trained, it can be used to make predictions on new, unseen data. The input data is fed into the trained network, and the output layer produces the predicted results based on the learned representations.

Implementation

We divided the preprocessed data, excluding Fuel Consumption and Electric Range, into training and testing sets. As mentioned earlier, 20% of the data was allocated for the test set, ensuring code reproducibility by setting the 'random_state' parameter to 123. Subsequently, we applied preprocessing to the data using the MinMaxScaler from the 'sklearn.preprocessing' library. By instantiating an instance of the MinMaxScaler class, we were able to normalize the data using the 'fit_transform' method for the training data and the 'fit' method for the test data. This ensured that all variables were scaled between 0 and 1, which is necessary for DNNs as the algorithm is sensitive to data scaling.

We created the DNN using the 'keras' library from tensorflow. The model was created sequentially, with layers instantiated and applied using the functional construction approach. Initially, we created

an input layer using the Input function from the Keras library, shaped according to the number of features in the training data. Next, three Dense Layers were initialized using the 'Dense' function from Keras. The first two layers, dense1 and dense2, contained 64 units each and employed the ReLU activation function. This structure was chosen as it is common for regression tasks and was introduced during our bootcamp. The final layer, responsible for prediction, consisted of 1 unit with a linear activation function suitable for regression tasks.

Afterwards, the input layer is passed through the dense layers sequentially using the functional construction approach. The output of dense1 is used as the input for dense2, and the output of dense2 is used as the input for predict. This creates a flow of information through the layers.

To finalize the model, the input and output layers were defined using the Model function from Keras, specifying the inputs and outputs as arguments. The structure of the model was obtained using the 'summary' method:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 28)]              0

 Dense_1 (Dense)             (None, 64)                1856

 Dense_2 (Dense)             (None, 64)                4160

 predict (Dense)             (None, 1)                 65

=================================================================
Total params: 6081 (23.75 KB)
Trainable params: 6081 (23.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 33. Model Summary for DNN Regression

Following model construction, we compiled the model using the 'compile' method from Keras. We opted for the Adam optimizer with a learning rate of 0.001, while the Mean Absolute Error (MAE) served as the chosen loss function. The model is trained using the 'fit' method on the training data with the number of epochs set to 40, representing the number of iterations over the entire training dataset. The batch size was defined as 32, determining the number of samples processed before updating the model's internal parameters. Additionally, a validation split of 0.2 was utilized, allocating 20% of the training data for validation during the training process. To visualize the decrease in MAE across epochs, we generated the following plot.

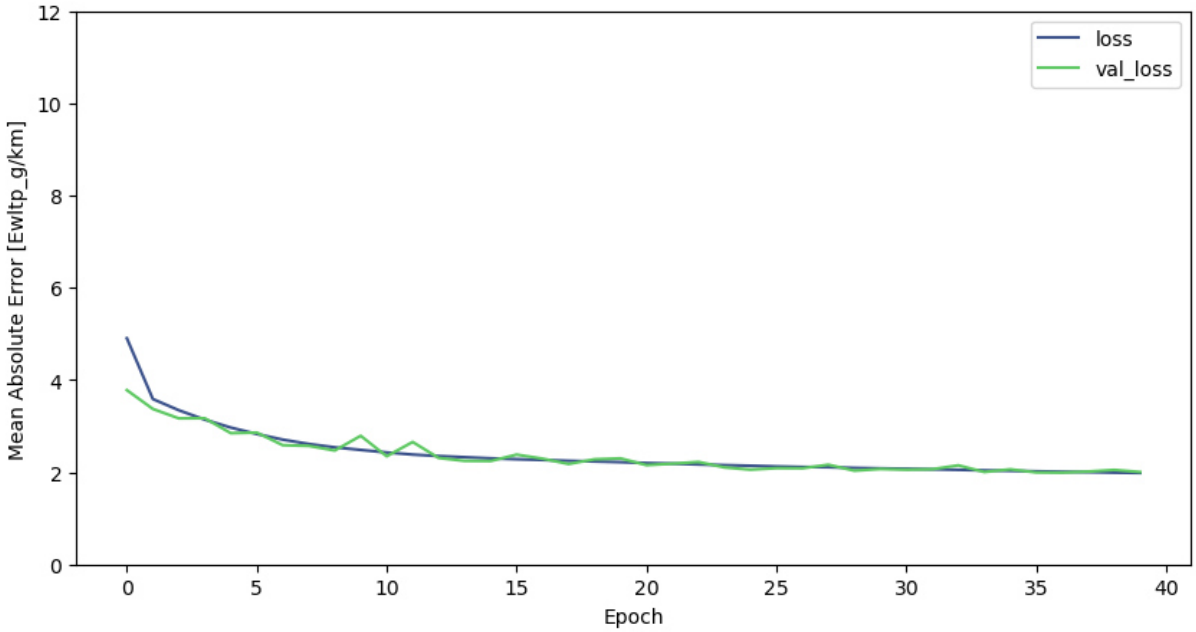


Figure 34. Learning Curve for DNN Regression

The plot demonstrates that during the initial 15 epochs, the Mean Absolute Error (MAE) experiences a significant decrease for both the testing and validation sets. Subsequently, it stabilizes at an approximate MAE of 2. Notably, the MAE values for the testing and validation sets are very similar, suggesting that Overfitting is not a concern. The plot shows the model's learning progress and performance enhancement, affirming that extending the number of epochs would be unnecessary, given that the model was maintained for the final 25 epochs.

Results

For the evaluation of the DNN model, we once again created an evaluation table using the 'get_evaluation_table' function.

	MSE	RMSE	MAE	R2 Score
Training Set	10.355	3.218	1.993	0.989
Test Set	10.447	3.232	1.997	0.988
Delta	0.093	0.014	0.004	-0.000

The model shows generally good results with a R^2 score of 0.988 for the testing set. Furthermore, the Test Set has low RMSE and MAE values, measuring 3.232 and 1.997, respectively, which shows the accuracy of the predictions. Overfitting does not seem to be an issue, as the differences between the Testing and Training Scores are very low. It is worth noting that, like previous models, the RMSE is bigger than the MAE, indicating the presence of outliers in terms of prediction accuracy.

Model Selection

In this section, we will compare the performances of the four baseline models to select the best algorithm for further improvement through hyperparameter tuning. The table below summarizes the evaluation metrics for the models, including the results of a Naïve model used as a benchmark. The Naïve model utilizes the mean of CO^2 emissions as a prediction, implemented through the DummyRegressor class from the 'sklearn.dummy' library.

Model	RMSE (Test)	Delta RMSE	MAE (Test)	Deta MAE	R^2 (Test)	Delta R^2
Decision Tree Regressor	1.667	0.046	0.938	0.008	0.997	0.000
Random Forest Regressor	1.664	0.039	0.938	0.006	0.997	0.000
XGBoost Regressor	2.075	0.026	1.290	0.001	0.995	0.000
DNN	3.232	0.014	1.997	0.004	0.988	0.000
Naive Regressor	29.969	-0.080	18.830	-0.040	-0.000	0.000

All models performed exceptionally well, with R^2 values ranging from 98.8% to 99.7%. The RMSE values range from 1.667 to 3.232, significantly lower than the benchmark of 29.969. Similarly, the MAE values for all models are low, ranging from 0.938 to 1.997 compared to the DummyRegressor's score of 18.83. Overfitting does not appear to be an issue as the deltas between the test and training scores are minimal. It is worth noting that all models exhibited outliers in terms of prediction accuracy, with higher RMSE values compared to MAE values. However, since our objective is not to predict real values but to interpret the relationship between features and the target, overall model performance is of greater importance than outliers.

Although all models perform well, the Decision Tree Regressor and Random Forest Regressor outperform the XGBoost Regressor and DNN, particularly in terms of RMSE and MAE values. One possible reason for this discrepancy is the XGBoost Regressor and DNN's heavy reliance on hyperparameters, which need to be fine-tuned for optimal performance. As the baseline models were not subjected to hyperparameter tuning, they may perform worse compared to the other models. The Decision Tree Regressor and Random Forest Regressor exhibit similar performance, making it challenging to select one algorithm as superior. Both models have a R^2 of 99.7% and an MAE of 0.938. However, the Random Forest Regressor demonstrates slightly less overfitting with a delta of 0.006 compared to the Decision Tree Regressor's delta of 0.008 for MAE. Additionally, the Random Forest Regressor marginally outperforms the Decision Tree Regressor in terms of RMSE, with a testing score of 1.664 compared to 1.667, exhibiting slightly less overfitting. Despite these slight advantages, the performance of the Decision Tree Regressor and Random Forest Regressor is so similar that it is not justifiable to use them as selection criteria. While the Decision Tree Regressor is a white box model, allowing us to comprehend how features affect the target, it has one primary disadvantage compared to the Random Forest Regressor. The Decision Tree model lacks stability, as even slight changes in the data can significantly impact the tree's structure. This is a concern since we aim to understand the relationship between car properties and CO^2 emissions. To ensure reliable statements about this relationship, the Random Forest model is a more stable option than the Decision Tree Regressor. Therefore we select the Random Forest Regressor as the best model for our task.

Hyperparameter Tuning

Random Search

We began the process of hyperparameter tuning with Random Search, an optimization algorithm that involves defining a hyperparameter space and randomly sampling points within it. Each sampled point is then evaluated to determine its performance, with the algorithm keeping track of the best solution found so far. This process repeats for a specified number of iterations or until a stopping criterion is met. Random Search was chosen initially over Grid Search due to its efficiency, as it does not use all possible combinations of hyperparameters. This efficiency was particularly relevant in our case, as training a single Random Forest model on our data took several minutes.

To implement Random Search, we utilized the 'RandomSearchCV' class from the 'sklearn.model_selection' library. We defined the hyperparameter space as follows:

For the 'Number of Estimators' parameter, which determines the number of trees in the Random Forest, we tested values from 10 to 200 to cover a range of forest sizes. For the 'Maximum Features' parameter, we considered the options 'auto', 'sqrt', and 'log2'. 'Auto' considers all features available in the dataset when determining the best split, while 'sqrt' and 'log2' focus on the square root and logarithm base 2 of the number of features, respectively. We also included the 'Maximum Depth' parameter, ranging from 1 (shallow tree) to 20 (complex tree structure), in the hyperparameter space. The 'Minimum Samples Split' parameter, which determines the optimal criteria for splitting internal nodes, was experimented with values from 2 to 20. Similarly, the 'Minimum Samples Leaf' parameter, which helps determine the optimal granularity for the tree leaves, was set between 1 and 20. Finally, we included the option to bootstrap (True or False) in the hyperparameter space.

Once we defined the parameter space, we used it to initialize the 'RandomizedSearchCV' with a new Random Forest Regressor. In the configuration, we set the Random Search to stop after 10 iterations and adopted a 5-fold cross-validation approach, utilizing Negative Mean Squared Error as the evaluation metric. To conduct the search on the training data, we employed the 'fit' method.

Unfortunately, the results of the Random Search were not satisfactory. The best model chosen performed significantly worse than the baseline Random Forest model, with a decrease in R-Square from 99.7% to 99.4% and an increase in RMSE from 1.663 to 2.328. This highlights that the default hyperparameters were already a good fit for our model, as experimenting with other values decreased the model's performance. Consequently, we decided to continue the hyperparameter tuning process with Grid Search, as it uses an exhaustive search and has the potential to yield better results than Random Search.

Grid Search

The Grid Search algorithm functions in a similar manner to Random Search. However, Grid Search involves defining a discrete grid of hyperparameters rather than a continuous hyperparameter space. The algorithm then proceeds to systematically test every possible combination of hyperparameter values defined in the grid. Like Random Search, Grid Search utilizes Cross-Validation to evaluate the models and determine the best model.

For the Grid Search, we defined a simple parameter grid, only varying the number of estimators (50,100,150,200) and the maximum depth of the Decision Trees ('None', 10 and 20). We chose a simplified grid since we were aware that the default parameters already performed well, and the training time for each new model was lengthy. To implement the Grid Search, we utilized the 'GridSearchCV' class from the 'sklearn.model_selection' module. We initialized the 'GridSearchCV' object with a new 'RandomForestRegressor' and the parameter grid we defined. In order to improve

computational efficiency, we reduced the number of cross-validation folds to 3. Finally, we executed the Grid Search by employing the 'fit' method on the training data.

We conducted an evaluation of the Random Search and Grid Search estimators on both the Training and Testing sets. The results are presented in the table below:

Model	RMSE (Test)	Delta RMSE	MAE (Test)	Delta MAE	R^2 (Test)	Delta R^2
Random Forest Regressor	16.636	0.0390	0.9383	0.0058	0.9969	0.0002
Random Forest Regressor (Random Search)	23.275	0.0187	15.038	0.0016	0.9940	0.0001
Random Forest Regressor (Grid Search)	16.633	0.0389	0.9382	0.0057	0.9969	0.0002
Naive Regressor	299.689	-0.0800	188.305	-0.0397	-0.0000	0.0000

From the results, it is evident that the Random Search did not effectively optimize the parameters, leading to a decrease in model performance. On the other hand, the best estimator from the Grid Search, which utilized 200 estimators and had an unlimited depth of Decision Trees, slightly outperformed the baseline model. The RMSE decreased from 16.636 to 16.633, and the MAE decreased from 0.9383 to 0.9382. The improvement is marginal, which reinforces the notion that the default parameters of the Random Forest Regressor were well-suited for our model. Our final model, which we will further analyze in the chapter 'Interpretation' is therefore the optimized Random Forest Regressor, with 200 estimators.

Classification

Selecting a Classification Model

Choosing an appropriate classification model is crucial for the success of a Machine Learning project. In our case, after careful evaluation of requirements and data context, we opted for the Random Forest, Decision Tree, and XGBoost models. The selection is based on various aspects:

1. Random Forest:

- **Ensemble Technique:** Random Forest is an ensemble technique based on multiple Decision Trees. Combining multiple models enhances robustness and reduces overfitting.
- **High Precision:** Aggregating predictions from many Decision Trees often results in higher precision and robustness against outliers compared to individual Decision Trees.
- **Handling Feature Diversity:** Random Forest can effectively handle datasets with a variety of features and provides automatic feature importance evaluations.

2. Decision Tree:

- **Simple Interpretation:** Decision trees are easily understandable and interpretable. This is crucial for gaining insights into the model's decision-making and explaining results to stakeholders.

- **Natural Decision Structure:** Decision Trees model decisions in an intuitive tree structure, often resembling human-like reasoning.

3. XGBoost (Extreme Gradient Boosting):

- **Gradient Boosting Technique:** XGBoost is a powerful implementation of the gradient boosting technique, allowing gradual improvement of model performance by combining multiple weak models.
- **High Precision and Speed:** XGBoost offers high precision with fast processing, making it a popular choice in competitions and demanding projects.
- **Overfitting Regulation:** Implementation of regularization techniques in XGBoost helps prevent overfitting and stabilizes model performance.

4. Overall Selection Justification:

- **Ensemble Advantages:** By integrating Random Forest and XGBoost, we harness the advantages of ensemble methods to achieve robust, precise, and generalizable classification.
- **Interpretability and Simplicity:** The inclusion of a Decision Tree allows for straightforward interpretation and explanation of model decisions, particularly crucial when the model influences political or business decisions.
- **Consideration of Data Complexity:** The model selection considers the potential complexity of the data, ranging from simpler Decision Trees to powerful ensemble techniques.

Overall, this selection provides a balanced trade-off between precision, interpretability, and robustness. The models were evaluated and fine-tuned on a validation dataset to ensure good generalization and avoid overfitting before assessing their final performance on the test dataset.

Oversampling

To tackle the issue of imbalanced class distribution within the dataset, oversampling techniques were employed. Specifically, the 'RandomOverSampler' from the 'imbalanced-learn' library was utilized. This process involves generating synthetic samples for the minority classes, thereby balancing the class distribution and preventing the model from being biased towards the majority class.

Before the application of oversampling, the dataset exhibited a certain distribution among its classes. Post-oversampling, the dataset underwent a transformation to ensure a more equitable representation of classes, contributing to a more balanced and fair training process. This strategic preprocessing step is essential for improving the model's capacity to recognize patterns and make accurate predictions across all classes, ultimately enhancing the Random Forest Classifier's robustness and reliability.

Machine Learning

Random Forest Classification

Random Forest Classification is a versatile and robust machine learning technique primarily employed for classification tasks. It belongs to the ensemble learning family, combining the strengths of multiple decision trees to enhance predictive accuracy and generalization to new data.

Algorithm Random Forest

The underlying concept of the Random Forest algorithm involves creating a collection of Decision Trees that collectively contribute to a robust and accurate prediction. This is achieved by introducing randomness into the learning process.

The first key aspect is the so-called Bagging (Bootstrap Aggregating). For each tree in the ensemble, a random sample (bootstrap sample) is drawn from the training dataset. This sample allows for some data points to be considered multiple times, while others may not be included at all. This results in diverse training datasets for each tree.

Another crucial mechanism is feature randomization. At each step of Decision Tree construction, only a random subset of the available features is considered. This randomness ensures that each tree focuses on different aspects of the dataset and reduces the correlation between the trees in the ensemble.

The actual Decision Trees are built according to the usual principles of a Decision Tree. By combining the decisions of all trees, the prediction is made through majority voting. This means that each tree prediction contributes to the final prediction, and the class with the most votes is selected.

The Random Forest algorithm offers various advantages, including high robustness against overfitting, high accuracy through the combination of different trees, and automatic feature evaluation. The diversity of the trees and the aggregated decision often result in precise predictions, making Random Forests a versatile and powerful method for classification tasks.

Implementation

In the implementation of the Machine Learning model, we employed the 'scikit-learn' library, a versatile and widely-used toolkit for various data analysis and modeling tasks. The primary focus was on utilizing the Random Forest Classifier, a powerful Ensemble Learning method. The 'RandomForestClassifier' class within scikit-learn provided the foundation for constructing the ensemble model, which is essentially a collection of Decision Trees.

The dataset was divided into training and testing sets with a testing size of 0.3, represented by X_train, y_train, X_test, and y_test. To better understand the features used in the analysis, we referenced data_ros.columns to obtain the column names. This step is crucial for maintaining clarity and coherence throughout the analysis.

During the training phase, the 'RandomForestClassifier' was configured with specific parameters to enhance model performance and efficiency. Cross-validation, a robust technique for assessing model generalization, was employed with n_folds set to 5. This parameter determines the number of folds in the cross-validation process, contributing to a thorough evaluation of the model's predictive capabilities.

Additionally, the ‘n_jobs’ parameter was set to -1, indicating parallelization across all available processor cores. This optimization significantly accelerated the training process, making it more efficient and scalable.

The resulting Random Forest model was then evaluated using the testing set, providing valuable insights into its performance metrics, such as accuracy, precision, recall, and F1-score. Moreover, the feature importance analysis was conducted to identify the significance of each feature in contributing to the model's predictions.

Results

The Random Forest Classifier showcased exceptional performance across various metrics, affirming its suitability for the vehicle classification task. With an impressive overall accuracy of 97%, the model demonstrated a high level of correctness in predicting vehicle types within the test dataset.

Metric	Score
Accuracy	0.97
Mean roc_auc_ovr	0.9993
Mean f1_macro	0.9712

A notable strength of the Random Forest model is evident in the 'mean roc_auc_ovr value' of 0.9993, emphasizing its outstanding ability to discriminate between different vehicle classes. This metric is particularly crucial in multi-class classification scenarios, and the model's near-perfect score reflects its robust discriminatory power.

Class	Precision	Recall	F1-Score
1	1.00	1.00	1.00
2	0.96	0.97	0.97
3	0.95	0.92	0.94
4	0.93	0.96	0.94
5	0.98	0.96	0.97
6	0.99	0.99	0.99
7	0.99	0.99	0.99

The 'mean f1_macro score', standing at 0.9712, further underscores the model's proficiency. This metric provides a balanced assessment of precision and recall across all classes, indicating that the Random Forest Classifier maintains a harmonious trade-off between accurately identifying positive instances and avoiding false positives.

Precision, a key metric, measures the model's ability to correctly classify positive instances among the predicted positive labels. In the context of the Random Forest Classifier, high precision values across different vehicle classes highlight the model's accuracy in avoiding false positives and making precise predictions for each category.

The confusion matrix provides a detailed breakdown of the model's predictions, illustrating correct and erroneous classifications for each class. Notably, the diagonal elements, representing correct predictions, dominate the matrix, confirming the model's overall accuracy.

Analyzing class-specific metrics, including precision, recall, and F1-Score, offers insights into the model's performance for individual classes. High precision values indicate the model's proficiency in minimizing false positives and making precise predictions for each vehicle type.

Predicted / True	1	2	3	4	5	6	7
1	206,082	10	0	0	0	0	0
2	145	200,824	5,123	1	0	0	0
3	0	8,677	190,291	7,124	0	0	0
4	4	0	4,506	197,029	4,553	0	0
5	0	0	0	7,217	198,112	763	0
6	0	0	0	0	145	203,904	2,043
7	0	0	0	0	0	1,163	204,929

In conclusion, the Random Forest Classifier has demonstrated exceptional accuracy, robust discrimination between classes, and precise identification of various vehicle types. These results highlight the model's efficacy and reliability for real-world applications in vehicle classification.

Feature Importance

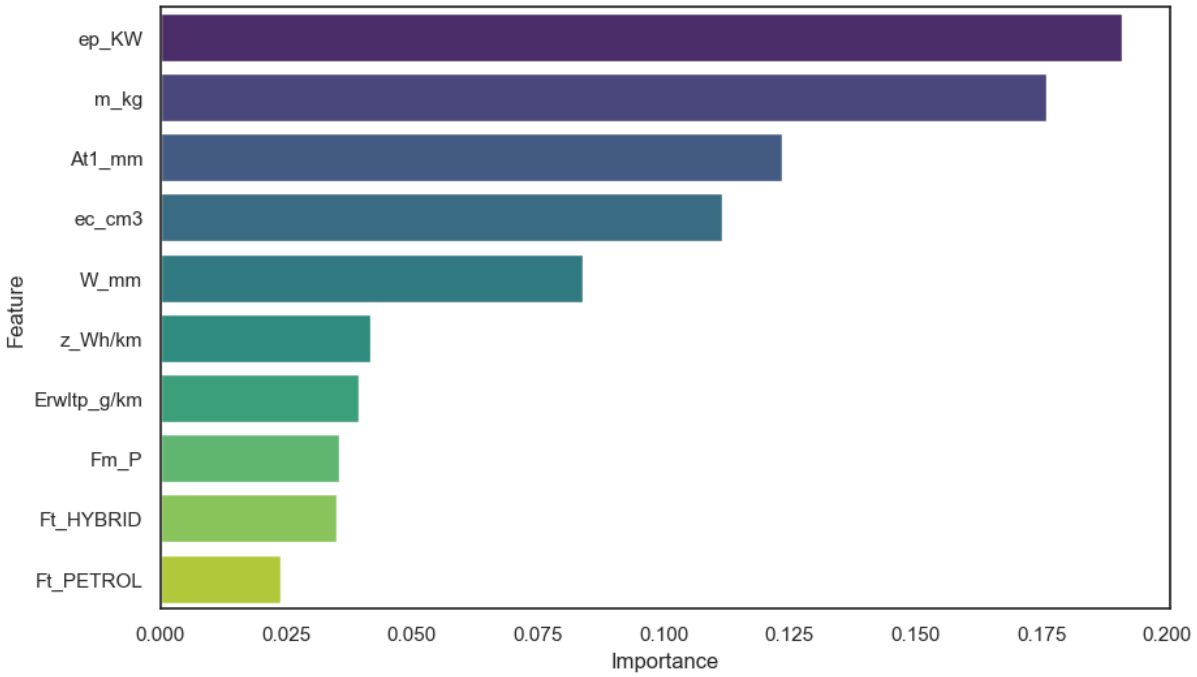


Figure 35. Top10 Feature Importance for Random Forest Classification

Feature importance in the context of a Random Forest Classifier is a measure of the contribution of each feature to the overall predictive performance of the model. Random Forest calculates feature importance based on the Gini impurity or information gain associated with each feature.

In a Random Forest model, feature importance is computed by evaluating how much the Gini impurity decreases when a particular feature is used to split the data. The higher the decrease in Gini impurity, the more important the feature is considered. The importance scores for all features are normalized to sum up to 1.

For the Random Forest Classifier you used, feature importance is obtained through the 'feature_importances_' attribute of the trained model. This attribute provides a ranked list of features along with their corresponding importance scores.

Interpreting feature importance is valuable for understanding the impact of each feature on the model's predictions. Features with higher importance scores are more influential in making accurate predictions. This information is useful for feature selection, identifying key factors driving the model's decisions, and gaining insights into the underlying patterns in the data.

For reporting purposes, presenting the top features along with their importance scores in a bar plot or table format can effectively communicate the relative significance of different features to stakeholders and decision-makers.

The bar plot above illustrates the top 10 feature importances obtained from our Random Forest classification model. These importances highlight the key factors influencing the model's predictions for CO^2 emissions from vehicles.

Engine power (‘ep_KW’) and vehicle weight (‘m_kg’) emerge as the most influential features, contributing 19.09% and 17.60%, respectively, to the decision-making process of the Random Forest model. The dimensional attribute At1_mm and engine capacity (‘ec_cm3’) also play crucial roles, representing 12.34% and 11.16% importance, respectively.

Furthermore, features such as ‘W_mm’, ‘z_Wh/km’, ‘Erwltp_g/km’, and fuel-related attributes like ‘Fm_P’, ‘Ft_HYBRID’, and ‘Ft_PETROL’, contribute significantly to the model's accuracy. The feature importance values range from 8.38% to 2.39%, indicating the varying degrees of influence each attribute has on the model's predictions.

The bar plot visually demonstrates the relative importance of these features in the Random Forest model, providing insights into the factors that significantly contribute to the accurate classification of CO^2 emissions from vehicles.

Decision Tree Classification

Decision Tree Algorithm

The Decision Tree algorithm is a powerful tool for classification tasks, employing a recursive, tree-like structure to make predictions based on input features. It begins at the root node, where the algorithm evaluates different features to identify the most informative one for splitting the dataset into distinct classes. The criteria for evaluation often involve metrics like Gini impurity or information gain.

As the algorithm progresses, it recursively partitions the data into subsets at each decision node, creating a branching structure. This process continues until a stopping criterion is met, such as reaching a maximum tree depth, having a minimum number of samples in a node, or achieving pure nodes with a homogeneous class distribution.

The leaf nodes of the tree represent the final predictions, with each leaf node corresponding to a specific class determined by the majority class of instances in that node. While Decision Trees are interpretable and provide transparency in decision-making, they are susceptible to overfitting, capturing noise from the training data. Pruning techniques can be applied to alleviate overfitting by removing nodes that do not significantly contribute to improving predictive accuracy on a validation set.

In summary, Decision Trees are intuitive, interpretable, and capable of capturing complex relationships in data. Their ability to visually represent decision-making processes makes them valuable for understanding feature importance. However, practitioners should be cautious about overfitting, and

techniques like pruning can be employed to strike a balance between model complexity and predictive accuracy.

Implementation

In our analysis, we implemented the Decision Tree Classifier using the 'scikit-learn' library in Python. The Decision Tree model stands out as a powerful tool for classification tasks due to its interpretability and versatility. The following outlines how we configured, trained, and evaluated the Decision Tree model:

We initiated the Decision Tree classifier using the DecisionTreeClassifier class from scikit-learn, a widely used Machine Learning library. The dataset was divided into training and testing sets using the 'train_test_split' function to ensure the model's generalization to unseen data.

To robustly assess the model's performance, we employed K-Fold Cross-Validation with 5 folds. This approach provides a more reliable estimate of the model's capabilities by training and testing on different subsets of the data.

For model evaluation, we chose essential scoring metrics, including 'roc_auc_ovr' (area under the ROC curve) and 'f1_macro' (F1-Score). These metrics offer insights into the model's ability to discriminate between classes and its overall performance.

To better understand the features contributing to the model's decisions, we visualized the feature importance using a bar plot. This step aids in identifying the most influential variables in the classification process.

Finally, we made predictions on the test set and presented the results through a confusion matrix and a classification report. These provide a detailed breakdown of predictions for each class, including precision, recall, and F1-Score.

Results

The Decision Tree model has demonstrated exceptional efficacy in predicting vehicle categories within the test set, achieving an impressive accuracy of 97%. This accuracy reflects the model's ability to make correct predictions across the seven distinct classes, showcasing its proficiency in handling complex classification tasks.

Metric	Score
Accuracy	0.97
Mean roc_auc_ovr	0.9993
Mean f1_macro	0.9711

- **Robust Discrimination Capability:** The model's discriminative power is highlighted by its exceptional mean roc_auc_ovr score of 0.9993. This score signifies the model's robustness in distinguishing between the different vehicle categories, showcasing its capability to effectively capture the underlying patterns in the data.
- **Precision-Recall Balance:** The mean f1_macro score, standing at 0.9711, indicates a harmonious balance between precision and recall across all classes. This equilibrium is crucial in classification tasks, ensuring that the model maintains accuracy while considering both false positives and false negatives. The model's ability to strike this balance is a testament to its adaptability in capturing the complexities of the dataset.

- **Confusion Matrix Insights:** The confusion matrix provides a detailed breakdown of the model's predictions for each class, offering valuable insights into various performance metrics.
- **Precision (0.93 to 1):** Precision scores ranging from 0.93 to 1 demonstrate the model's precision in correctly identifying instances for each class. A high precision score implies a low rate of false positives, indicating the model's accuracy in labeling instances.
- **Recall (0.92 to 1):** Recall values between 0.92 and 1 reflect the model's effectiveness in capturing a significant portion of positive instances for each class. A high recall emphasizes the model's ability to detect relevant patterns within the data.
- **F1-Score (0.94 to 1):** F1-Scores, harmonizing precision and recall, range from 0.94 to 1, underscoring the model's overall accuracy. The F1-Score considers both false positives and false negatives, providing a comprehensive assessment of the model's performance.

Class	Precision	Recall	F1-Score
1	1.00	1.00	1.00
2	0.96	0.97	0.97
3	0.95	0.92	0.94
4	0.93	0.96	0.94
5	0.98	0.96	0.97
6	0.99	0.99	0.99
7	0.99	0.99	0.99

Predicted \ True	1	2	3	4	5	6	7
1	206,08	10	0	0	0	0	0
2	145	200,830	5,117	0	0	0	0
3	0	8,678	190,290	7,124	0	0	0
4	4	0	4,515	197,016	4,556	0	0
5	0	0	0	7,217	198,112	763	0
6	0	0	0	0	145	203,904	2,043
7	0	0	0	0	0	1,163	204,929

In conclusion, the Decision Tree model exhibits a robust classification performance, excelling in accuracy, discrimination capability, and the delicate balance between precision and recall across diverse vehicle classes. These findings establish the Decision Tree model as a compelling choice for this classification task, laying the groundwork for further model comparisons and in-depth analysis.

Feature Importance

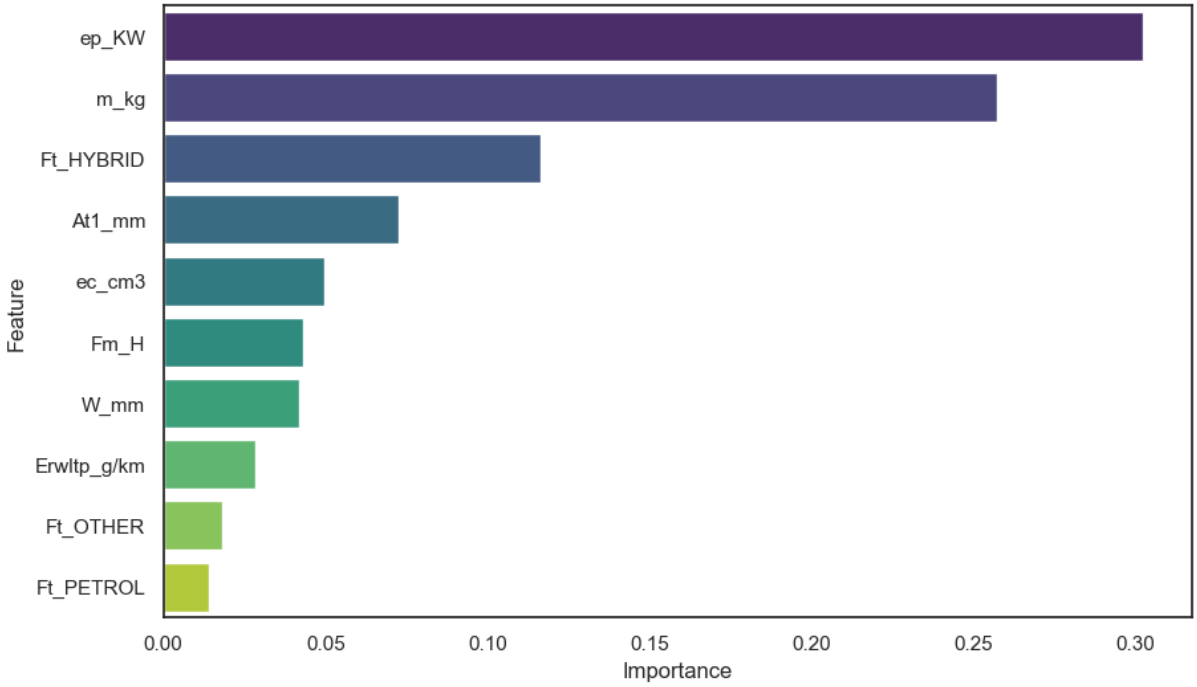


Figure 36. Top10 Feature importance for Decision Tree Classification

In the context of a Decision Tree Classifier, feature importance represents the significance of each feature in making decisions within the tree. Unlike Random Forest, a single Decision Tree's feature importance is typically computed using metrics such as Gini impurity or information gain.

The feature importance for a Decision Tree is calculated based on how much each feature contributes to the overall reduction in impurity or entropy when creating decision nodes. Features with higher importance have a more substantial impact on the tree's ability to separate and classify instances.

The Decision Tree Classifier in 'scikit-learn' provides feature importance through the 'feature_importances_' attribute, similar to Random Forest. The values in this attribute represent the relative importance of each feature.

Interpreting feature importance in a Decision Tree is crucial for understanding which features play a crucial role in the model's decision-making process. It helps identify the most discriminative features, providing insights into the underlying patterns in the data.

For reporting purposes, presenting the top features along with their importance scores in a bar plot or table format can help communicate the relevance of different features in influencing the Decision Tree's decisions. This information aids in feature selection, model interpretation, and improving the overall understanding of the predictive behavior of the model.

The bar plot illustrates the top 10 feature importances derived from our Decision Tree classification model. The analysis underscores the pivotal role of certain features in predicting CO^2 emissions from vehicles. Notably, engine power (‘ep_KW’) and vehicle weight (‘m_kg’) emerge as the most influential factors, contributing 30.22% and 25.73%, respectively, to the model's decision-making process. The presence of hybrid fuel type (‘Ft_HYBRID’) also plays a significant role, representing 11.66% importance in the model's predictions.

In addition, the dimensional features, ‘At1’_mm and ‘W_mm’, as well as engine-related attributes like fuel metering system (‘Fm_H’) and engine capacity (‘ec_cm3’), are identified as essential contributors

to the model's robust performance. The feature importance values, ranging from 4.41% to 0.37%, emphasize the varying degrees of influence each attribute exerts on the model's predictions.

XGBoost Classification

XGBoost algorithm

XGBoost builds a boosted ensemble of Decision Trees in a sequential manner. Each tree corrects the errors made by the previous ones, and their predictions are combined to form the final prediction. The algorithm minimizes a loss function, which quantifies the difference between predicted and actual values.

In each iteration, a new Decision Tree is added to the ensemble. The trees are trained to fit the negative gradient of the loss function, making subsequent trees focus on the examples that the current ensemble finds difficult. Regularization terms, such as L1 and L2 regularization, are incorporated into the objective function to control the complexity of individual trees and the overall model.

XGBoost employs a process called pruning to control the size of the trees, preventing them from becoming too deep and overfitting the training data. It also handles missing values by incorporating them into the tree construction process, allowing the algorithm to make decisions even when certain features are unavailable.

Parallel processing is a notable feature of XGBoost, enabling efficient computation and faster training times. The algorithm is highly customizable, allowing fine-tuning of hyperparameters to achieve optimal performance for specific datasets.

In summary, Decision Trees make decisions based on local information, while XGBoost combines the strengths of multiple trees to create a powerful ensemble. XGBoost's regularization, handling of missing data, and parallel processing contribute to its efficiency and effectiveness in producing accurate and robust predictions.

Implementation

In our classification task, we utilized the powerful XGBoost algorithm, implemented through the 'scikit-learn' library. The dataset was meticulously prepared by splitting it into training, validation, and test sets, maintaining a stratified distribution of classes to ensure representative sampling.

To enhance model performance, feature scaling was applied using the 'StandardScaler' from scikit-learn. This step standardized the dataset, ensuring that all features contribute equally to the model.

The XGBoost model was initialized with specific parameters tailored for multi-class classification. Notably, we set the objective parameter to 'multi:softmax' and defined the number of classes using 'num_class=7'. Additionally, the model was configured to run on a GPU for faster training, employing the 'cuda' device.

The K-fold cross-validation technique was employed for robust evaluation, leveraging the KFold class from scikit-learn. Key scoring metrics, such as 'roc_auc_ovr' and 'f1_macro', were utilized to assess the model's performance under various conditions.

Feature importance was visualized using a custom function, offering insights into which features played a crucial role in the model's decision-making process.

Our trained XGBoost model was then applied to the test set for predictions. The results were evaluated using a confusion matrix, providing a detailed breakdown of true positive, true negative, false positive, and false negative predictions. Additionally, a classification report was generated, presenting metrics such as precision, recall, and F1-score for each class.

Throughout this process, scikit-learn provided essential utilities for data preparation, model evaluation, and metric computation. The 'xgboost' library, integrated seamlessly with scikit-learn, facilitated the implementation of a high-performing classification model with efficient GPU utilization.

Results

Our XGBoost classification model yielded impressive results across various metrics, showcasing its robust performance. The mean roc_auc_ovr (Receiver Operating Characteristic Area Under the Curve - One vs. Rest) score of approximately 0.9991 indicates high discriminative power across multiple classes.

Metric	Score
Accuracy	0.97
Mean ROC AUC (One-vs-Rest)	0.9991
Mean F1-Score (Macro)	0.9691

The mean f1_macro score, measuring the model's ability to balance precision and recall, was observed to be around 0.9691. This score signifies strong overall predictive performance across all classes.

Class	Precision	Recall	F1-Score
1	1.00	1.00	1.00
2	0.96	0.97	0.97
3	0.95	0.92	0.93
4	0.93	0.95	0.94
5	0.97	0.96	0.97
6	0.99	0.99	0.99
7	0.99	0.99	0.99

The confusion matrix reveals a robust performance of the model across various classes. The model excels in accurately predicting instances from Classes 1, 2, 5, 6, and 7, showcasing high precision, recall, and F1-scores. Notably, Classes 1, 6, and 7 exhibit flawless recognition, with perfect scores across all metrics.

Class 2 also demonstrates strong performance with precision and recall around 0.95, indicating accurate predictions. Classes 3 and 4, while displaying slightly lower scores, still exhibit good recognition by the model.

These findings underscore the model's effectiveness in handling diverse classes, with distinct strengths observed in recognizing different emission categories.

Predicted / True	1	2	3	4	5	6	7
1	144,260	4	0	0	0	0	0
2	102	140,335	3,827	0	0	0	0
3	0	6,185	133,017	5,058	4	0	0
4	2	8	3,623	137,193	138,088	0	0
5	0	0	5	5,439	148	733	0
6	0	0	0	0	725	142,740	1,377
7	0	0	0	0	0	948	143,317

The weighted average metrics, such as weighted precision, recall, and F1-score, provide an overall assessment of the model's performance, considering the class imbalance in the dataset. With an accuracy of 0.97, the model demonstrates its ability to make correct predictions on the majority of instances.

In summary, our XGBoost Classification model, equipped with carefully tuned parameters and trained on well-prepared data, achieved exceptional performance, making it a reliable choice for multi-class classification tasks. The model's proficiency is evident in its high accuracy and balanced performance across all classes.

Feature Importance

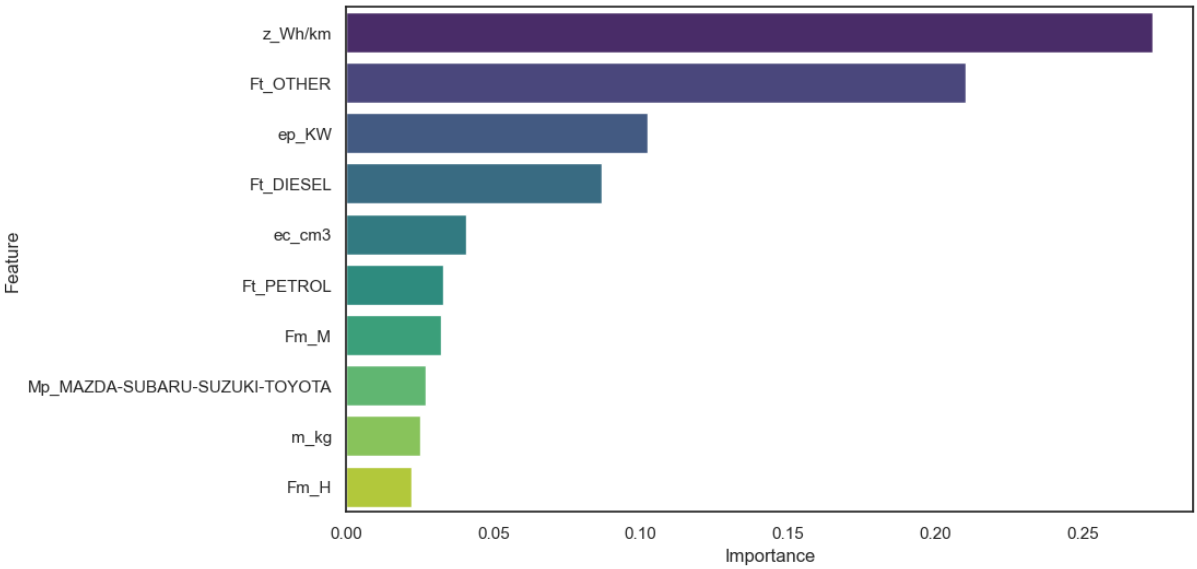


Figure 37. Top10 Feature Importance XGBoost-Classification

Analyzing the feature importance of our XGBoost model provides valuable insights into the key factors influencing CO^2 emission predictions. The top 10 crucial features and their respective weights shed light on the relative contributions to the model's performance.

The pivotal feature is 'z_Wh/km' (charging energy per kilometer) with a substantial importance of 27.4%. This suggests that the energy consumption during charging significantly impacts CO^2 emissions.

'Ft_OTHER' (fuel type: OTHER) is also significant (21.04%), indicating that vehicles with fuel types other than diesel or petrol substantially influence predictions.

Engine power ('ep_KW') with 10.22% and the fuel type 'Ft_DIESEL' with 8.68% are additional crucial factors. 'ec_cm3' (engine displacement), 'Ft_PETROL' (fuel type: petrol), and the model type 'Fm_M' also contribute significantly to the model's performance.

Manufacturer-specific features like 'Mp_MAZDA-SUBARU-SUZUKI-TOYOTA' also have an influence of 2.69%. Vehicle weight ('m_kg') and model type 'Fm_H' complete the top 10.

Hyperparameter Tuning

The continuous refinement of Machine Learning models is a crucial step to maximize their performance and precision. In this process, hyperparameter tuning plays a central role by allowing the targeted adjustment of various algorithm settings to extract the best possible results from a model.

Hyperparameters are external configurations that need to be predetermined, as they are not optimized by the learning process itself. Tuning these hyperparameters is essential to enhance the effectiveness of Machine Learning models and tailor them to the specific requirements of a task.

In our exploration of hyperparameter tuning techniques, we experimented with both 'RandomSearch' and 'GridSearch'. While both methods aim to find optimal sets of hyperparameters, our findings revealed that Grid Search yielded the most significant success in our context and was better suitable for our large dataset.

In the next steps, the focus was on hyperparameter tuning in the context of optimizing classification models for categorizing CO^2 emission classes. Our aim was not only to improve the predictive accuracy of the models but also to ensure that they respond efficiently and effectively to the challenges presented by our datasets. Through the deliberate adjustment of hyperparameters, we strived to find a balanced compromise between model complexity, computational efficiency, and accuracy. This introduction sets the stage for a detailed analysis of our efforts in hyperparameter tuning and the improvements achieved in model performance.

In the pursuit of refining our modeling strategies to address the challenges posed by our extensive dataset, we strategically extended our hyperparameter tuning efforts to optimize both the Decision Tree and XGBoost models. While the Random Forest model had previously exhibited commendable performance, our decision to emphasize the Decision Tree and XGBoost models was motivated by the necessity to overcome specific hurdles encountered during our analyses.

Given the intricacies inherent in our dataset, characterized by its substantial volume, we acknowledged the critical need to enhance the computational efficiency and predictive accuracy of our models. Consequently, our hyperparameter adjustments were tailored not only to the nuances of the Decision Tree model but also to the sophisticated algorithmic architecture of XGBoost. For the Decision Tree, parameters such as the splitting criterion (‘criterion’) and maximum tree depth (‘max_depth’) were meticulously fine-tuned. Simultaneously, for XGBoost, a broader range of key parameters, including the number of boosting rounds ('n_estimators'), learning rate ('learning_rate'), maximum tree depth ('max_depth'), minimum loss reduction for further partition ('gamma'), and L1 and L2 regularization terms ('alpha and lambda'), were subjected to optimization.

Navigating through the complexities presented by large datasets and performance constraints, our hyperparameter tuning approach involved a nuanced consideration of the unique characteristics of both the Decision Tree and XGBoost models. Employing techniques like Grid Search, we diligently

sought optimal sets of hyperparameters for each model, aiming to strike a delicate balance between computational efficiency and model effectiveness.

This dual focus on both the Decision Tree and XGBoost models reflected our commitment to overcoming challenges associated with data size, thereby empowering us to extract meaningful insights and classifications from our extensive dataset. By tailoring our hyperparameter tuning strategies to the specific nuances of each algorithm, we aimed to harness the full potential of these models for robust and accurate predictions in the context of our complex data landscape.

Hyperparameter Tuning XGBoost

The process of hyperparameter tuning was initiated by defining a comprehensive hyperparameter space for the XGBoost model. This space included critical parameters such as 'n_estimators', 'gamma', 'alpha', 'lambda', 'learning_rate', and 'max_depth'. Each of these parameters plays a unique role in formulating the model structure and significantly influences the quality of predictions.

1. '**n_estimators**':
   - The number of trees in the model is crucial for ensemble performance.
   - Here, the value was set to 50, ensuring a sufficient number of trees to leverage diversity and improve model performance.
2. '**booster**':
   - The choice of booster is fundamental to the functionality of the XGBoost model.
   - By setting it to 'gbtree,' a tree-based boosting method was chosen, indicating that each tree in the ensemble serves as a decision unit contributing to the overall prediction.
3. '**gamma**':
   - Gamma is a parameter controlling the complexity of trees by specifying the minimum reduction in the loss function for a leaf split.
   - The tested values (0.01, 0.1) indicate that the search for the optimal gamma setting aims to calibrate model complexity and prevent overfitting.
4. '**alpha**' **and** '**lambda**':
   - Alpha and Lambda are regularization terms that affect leaf weighting, contributing to controlling overfitting.
   - The tested values (0, 0.5) for 'alpha' and (0.1, 0.5) for 'lambda' show a careful balance between L1 and L2 regularization to find the optimal trade-off between model complexity and robustness.
5. '**learning_rate**':
   - The learning rate influences the size of steps taken during the gradient descent algorithm.
   - Here, the learning rate was set to 0.7, indicating a preference for moderate adjustments in weights to ensure both convergence and stability in the training process.
6. '**max_depth**':
   - The maximum depth of trees influences model complexity and the risk of overfitting.
   - Setting it to 9 chose sufficient depth to capture complex patterns without increasing the risk of overfitting.

The decision to use a Grid Search approach allowed for a systematic and comprehensive exploration of the hyperparameter space. Iterating over the defined values of each parameter aims to identify the optimal configuration that maximizes model performance. This detailed consideration and adjustment of hyperparameters underscore our commitment to precision and efficiency in model development, especially concerning the specific requirements of our complex dataset.

For the training processes, KFold cross-validation techniques with 3 folds and random shuffling were employed. These ensured a reliable evaluation of model performance across different data splits and helped identify potential overfitting. Additionally, the 'early stopping' concept was integrated, where the model was evaluated on a validation dataset, and training was halted if no improvement in model performance was observed in 15 consecutive rounds. This approach is crucial to ensuring precise model configuration while preventing overfitting.

The final adjustment of the XGBoost model was made using the best-identified hyperparameters. This specific focus on the XGBoost model allowed us to address the unique challenges of our dataset and develop a precise, powerful prediction tool. This iterative process of hyperparameter tuning highlights our commitment to continuous model optimization and adaptation to the specific demands of our complex dataset.

Results Optimized XGBoost

The evaluation of our optimized XGBoost classification model indicates a commendable performance, with a mean AUC-ROC of 0.9463 and a mean F1-macro of 0.8852. These metrics suggest the model's robust ability to distinguish between positive and negative examples while achieving a balanced trade-off between precision and recall.

Metric	**Score**
Accuracy	0.97
Mean ROC AUC (One-vs-Rest)	0.9463
Mean F1-Score (Macro)	0.8852

Examining the confusion matrix, we observe excellent performance in accurately predicting instances across various classes. Notably, Classes 1, 2, 5, 6, and 7 exhibit near-perfect precision, recall, and F1-scores, indicating the model's proficiency in handling diverse emission categories.

Predicted / True	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	144,256	7	0	0	0	0	0
2	109	140,459	3,696	0	0	0	0
3	0	6,123	133,045	5,097	0	0	0
4	2	0	3,236	135,785	5,242	0	0
5	0	0	0	3,218	140,400	647	0
6	0	0	0	0	132	142,848	1,284
7	0	0	0	0	0	1,027	143,237

While global metrics such as precision, recall, and F1-score are moderate, the overall accuracy stands at 0.97, indicating the model's correct predictions in the majority of cases. Further insights from class-specific metrics reveal strong performance in certain classes and highlight opportunities for enhancement in others.

Class	Precision	Recall	F1-Score
1	1.00	1.00	1.00
2	0.96	0.97	0.97
3	0.95	0.92	0.94
4	0.94	0.94	0.94
5	0.96	0.97	0.97
6	0.99	0.99	0.99
7	0.99	0.99	0.99

In summary, the optimized XGBoost model showcases strong predictive capabilities, with room for targeted improvements in specific classes. These results guide future refinements to maximize the overall accuracy and effectiveness of the model in predicting vehicle CO^2 emissions.

Feature Importance

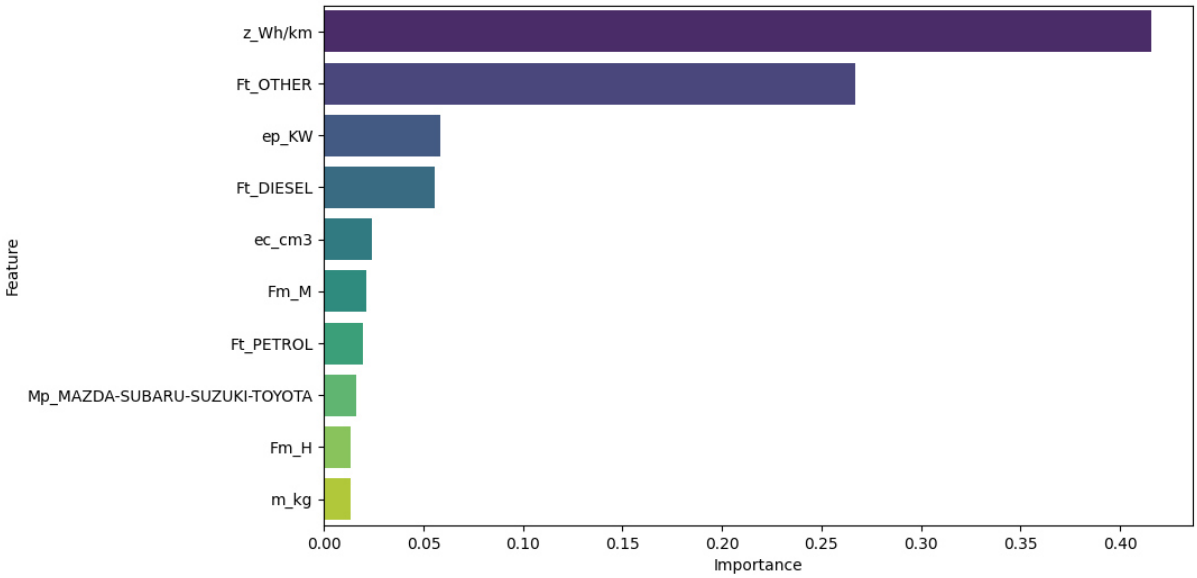


Figure 38. Top10 Feature Importance Optimized XGBoost

The optimized XGBoost classification model reveals crucial insights into the factors driving its predictions. At the forefront is the energy consumption per kilometer (‘z_Wh/km’) with a substantial importance of 41.60%. This indicates that vehicles exhibiting lower energy consumption per kilometer significantly shape the model's forecasts. The diversity of fuel types also plays a pivotal role, particularly with "Other" fuel types (‘Ft_OTHER’), contributing significantly at 26.68%.

Engine power (‘ep_KW’) remains a vital determinant, though slightly less influential than energy consumption, boasting an importance score of 5.82%. The specific fuel type of Diesel (‘Ft_DIESEL’) holds importance as well, accounting for 5.54% of the model's decision-making process. Engine capacity (‘ec_cm3’) is relevant but to a lesser extent, contributing 2.43%.

Additional considerations include the transmission type, where vehicles with manual transmission (‘Fm_M’) contribute modestly at 2.12%. Fuel type, specifically petrol (‘Ft_PETROL’), plays a role with a

modest importance of 1.95%. Certain manufacturers, including Mazda, Subaru, Suzuki, and Toyota (‘Mp_MAZDA-SUBARU-SUZUKI-TOYOTA’), contribute 1.63% to the model's predictions.

Hyperparameter Tuning Decision Tree

The hyperparameter tuning process for the Decision Tree model involved a meticulous exploration of key parameters to enhance the model's predictive performance. Here's a detailed breakdown of the hyperparameters and their respective settings:

1. **‘criterion’:**
   - The criterion parameter defines the quality measurement function for the split.
   - Options considered: 'gini', 'entropy', and 'log_loss'.
1. **‘max_depth’:**
   - Specifies the maximum depth of the Decision Tree.
   - Tested values: 5, 10, 25, 50, 100.

The tuning process also followed a systematic Grid Search approach, where various combinations of these hyperparameters were evaluated using KFold cross-validation with 3 folds and random shuffling. This ensured a robust assessment of model performance across different data splits, helping identify the optimal configuration.

The chosen values for criterion aimed to assess different split criteria, such as Gini impurity, information gain (‘entropy’), and ‘log loss’. For ‘max_depth’, the exploration focused on varying tree depths to strike a balance between capturing complex patterns and preventing overfitting.

The Grid Search methodology allowed for an exhaustive search through the hyperparameter space, aiming to maximize the model's effectiveness. This methodical approach aligns with our commitment to precision and efficiency in model development, tailored to the specific requirements of our dataset.

Results Optimized Decision Tree

The optimized Decision Tree model showcases exceptional performance across various evaluation metrics, affirming its proficiency in making accurate predictions across multiple classes. The ‘mean roc_auc_ovr score’, measuring the model's ability to discriminate between classes, impressively stands at approximately 0.9992. This indicates a high level of precision in distinguishing different classes, which is crucial for reliable predictions.

Metric	Score
Accuracy	0.97
Mean ROC AUC (One-vs-Rest)	0.9992
Mean F1-Score (Macro)	0.9711

Furthermore, the macro F1-Score, with a value of approximately 0.9711, underlines the model's robust precision and recall across all classes. This metric provides a comprehensive assessment of the model's performance, especially in scenarios with imbalanced class distribution.

Class	Precision	Recall	F1-Score
1	1.00	1.00	1.00
2	0.96	0.97	0.97
3	0.95	0.92	0.94
4	0.93	0.96	0.94
5	0.98	0.96	0.97
6	0.99	0.99	0.99
7	0.99	0.99	0.99

Examining the confusion matrix, we observe that the model excels in predicting Classes 1, 6, and 7, achieving near-perfect precision, recall, and F1-Score for these categories. Similarly, Classes 2, 4, and 5 exhibit high precision, recall, and F1-Score, indicating the model's effectiveness in making accurate predictions for these classes. Although Class 3 displays slightly lower precision, recall, and F1-Score compared to others, the model still demonstrates strong predictive ability across the majority of classes.

Predicted / True	1	2	3	4	5	6	7
1	206,082	10	0	0	0	0	0
2	145	200,828	5119	0	0	0	0
3	0	8,678	190,285	7,129		0	0
4	6	0	4,514	197,017	4,557	0	0
5	0	0	0	7,217	198,112	763	0
6	0	0	0	0	145	203,904	2,043
7	0	0	0	0	0	1,163	204,929

The classification report provides a detailed breakdown of precision, recall, and F1-Score for each class, offering insights into the model's performance on an individual basis. The weighted average F1-Score, precision, and recall, all hovering around 0.97, underscore the overall high accuracy of the model.

With an overall accuracy of 0.97, the optimized Decision Tree model is well-tailored to handle the complexities of the dataset. These results affirm the success of hyperparameter tuning, resulting in a finely calibrated model capable of making accurate and reliable predictions across diverse classes.

Feature Importance

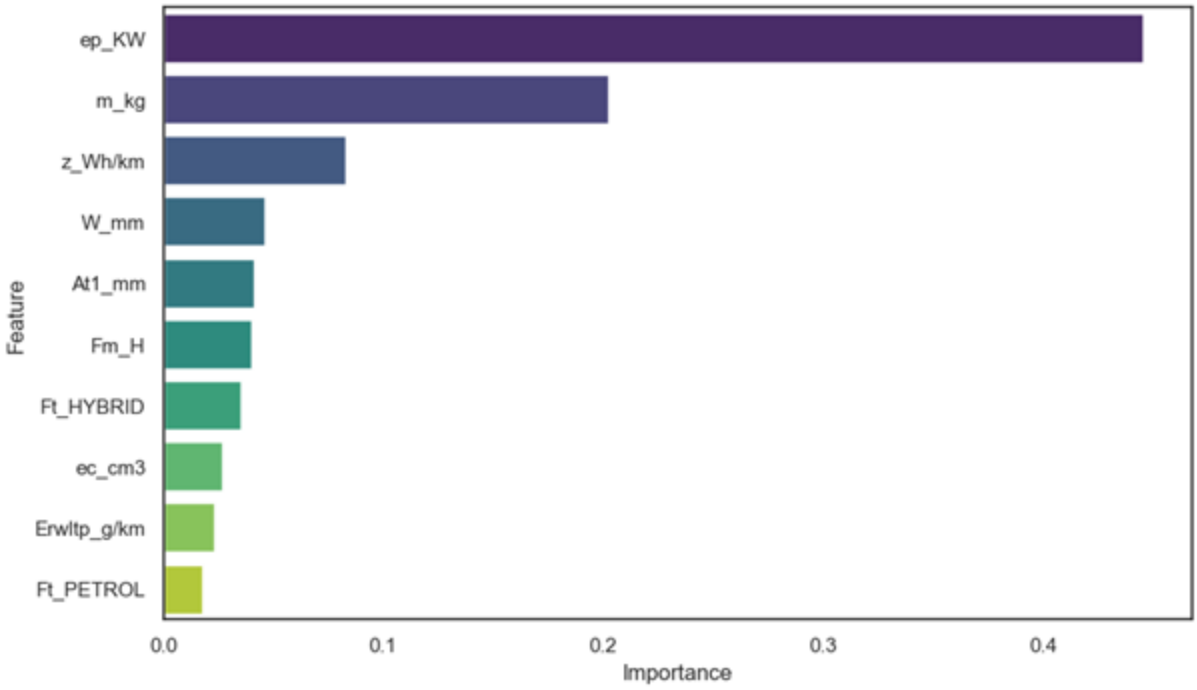


Figure 39. Top10 Feature importance optimized Decision Tree

The feature importance analysis for the optimized decision tree model provides crucial insights into the features that significantly contribute to predicting the target variable. At the forefront is the engine power ('ep_KW'), playing a substantial role with a contribution of 44.51%, followed by vehicle mass ('m_kg') as the second most important feature with a share of 20.20%. Energy consumption per kilometer ('z_Wh/km') also holds considerable importance, contributing 8.28%.

Wheelbase ('W_mm') and front track width ('At1_mm') are pivotal factors influencing the model's decisions, with contributions of 4.59% and 4.12%, respectively. The presence of hybrid as a fuel mode ('Fm_H') and hybrid vehicles ('Ft_HYBRID') each contribute significantly, with 4.03% and 3.51%, respectively.

Less decisive but still relevant features include engine capacity ('ec_cm3') with a contribution of 2.70%, emissions reduction through innovative technologies ('Erwltp_g/km') with 2.31%, and vehicles using petrol as a fuel type ('Ft_PETROL') with 1.76%.

Deep Learning

In the previous phase, we implemented Machine Learning models to predict the CO^2 emission class for the cars within our dataset, and the obtained results were satisfactory. However, a natural curiosity arises: can we achieve comparable or even superior performances using Deep Learning? The transition from traditional Machine Learning to Deep Learning introduces a pivotal question—can the intricate patterns and relationships within the data be discerned more effectively through the complexity of deep neural networks? As we explore Deep Learning, the main challenge is clear: not just trying to match existing performance but also striving to exceed it.

Implementation

In alignment with the principles outlined in the regression chapter, the Fuel Consumption and Electric Range features are dropped from the dataset. During the search for the best model architecture and the best hyperparameters, the data is splitted to keep 20% for validation and 80% for training (this will be changed once the final model is selected). A random state (123) has been employed for reproducibility. Additionally, the data underwent preprocessing using MinMaxScaler to ensure uniform scaling across features.

The model architecture consists of three layers. The initial layer, comprising 32 units, is designed to closely match the number of features, promoting an intuitive correlation between input dimensions and layer units. This layer employs the Rectified Linear Unit (ReLU) activation function, known for its effectiveness in handling non-linearities.

Subsequently, a second layer with 128 units, also utilizing the ReLU activation function, has been incorporated. This layer's increased complexity allows the model to capture more intricate patterns in the data.

The final layer, encompassing 7 units, aligns with the number of classes to be predicted. In this multi-class classification scenario, a softmax activation function is applied. Softmax transforms the raw output scores into a probability distribution across all classes. This probabilistic interpretation facilitates a clearer understanding of the model's confidence in each class prediction.

By adopting the described model architecture and preprocessing steps, the aim is to construct a neural network capable of effectively classifying instances into the designated classes. The choice of activation functions and layer configurations is guided by both empirical success in similar tasks and theoretical considerations, aiming to strike a balance between model complexity and interpretability.

The first model is described as follows:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense1 (Dense)              (None, 32)                544

 dense2 (Dense)              (None, 128)               4224

 dense3 (Dense)              (None, 7)                 903

=================================================================
Total params: 5671 (22.15 KB)
Trainable params: 5671 (22.15 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 40. Model Summary

The Adam optimizer was employed with a default learning rate of 0.001 in the second phase of model configuration. The choice of the Adam optimizer, known for its efficiency in handling large datasets and adaptive learning rates, aligns with the aim of enhancing the model's convergence speed.

To assess the model's performance, accuracy was selected as the evaluation metric. Accuracy provides a comprehensible measure of the model's overall correctness in predicting the correct class labels. Additionally, it is particularly relevant in the context of multi-class classification tasks, offering insights into the model's ability to correctly classify instances across all classes.

For the loss function, 'sparse_categorical_crossentropy' was chosen. This choice is motivated by the categorical nature of the target variable, where each instance belongs to a specific class. The sparse version of the loss function is well-suited for scenarios where the target labels are integers, simplifying the task of model training.

To start with the model's learning process, we opted for 20 epochs and a batch size of 1024. This configuration provides a balance between allowing the model to undergo multiple iterations through the entire dataset (epochs) and efficiently updating the model parameters using batches of data. The specific choice of these hyperparameters is often the result of empirical experimentation to achieve a balance between model performance and computational efficiency.

The implementation of these configurations is illustrated in the code snippet below:

```python
dnn = Sequential()
dnn.add(Dense(units=32, activation='relu', name='dense1' ))
dnn.add(Dense(units=128, activation='relu', name='dense2'))
dnn.add(Dense(units=7, activation='softmax', name='dense3'))

optimizer = Adam(learning_rate=0.001)
dnn.compile(loss = 'sparse_categorical_crossentropy',
            optimizer = optimizer,
            metrics = ['accuracy'])

training_history = dnn.fit(X_train_dnn, X_train_dnn,
                           epochs = 20,
                           batch_size = 1024,
                           validation_data=(X_test_dnn, y_test_dnn))

```

Python

In the graph below, the training history data is plotted to show the evolution of both accuracy and loss over epochs for both Training and validation. These plots allow us to evaluate the model's progression during training. From the plot, it is more likely that we didn't reach convergence, this is a reason to further test with a higher number of epochs. The obtained maximum accuracy score of 0.94, leaves room for improvement, suggesting that additional training iterations may reveal further refinements in the model's performance.

On the other hand, the absence of a discernible overfitting issue is a positive outcome. The training and validation curves seem to be tracking closely, indicating that the model has not merely memorized the training data but is demonstrating a capacity to generalize well to unseen instances.

To enhance performance, an extended training with a higher number of epochs could be considered. This iterative approach aligns with the model development process, allowing for a more comprehensive exploration of the model's learning dynamics and convergence behavior.

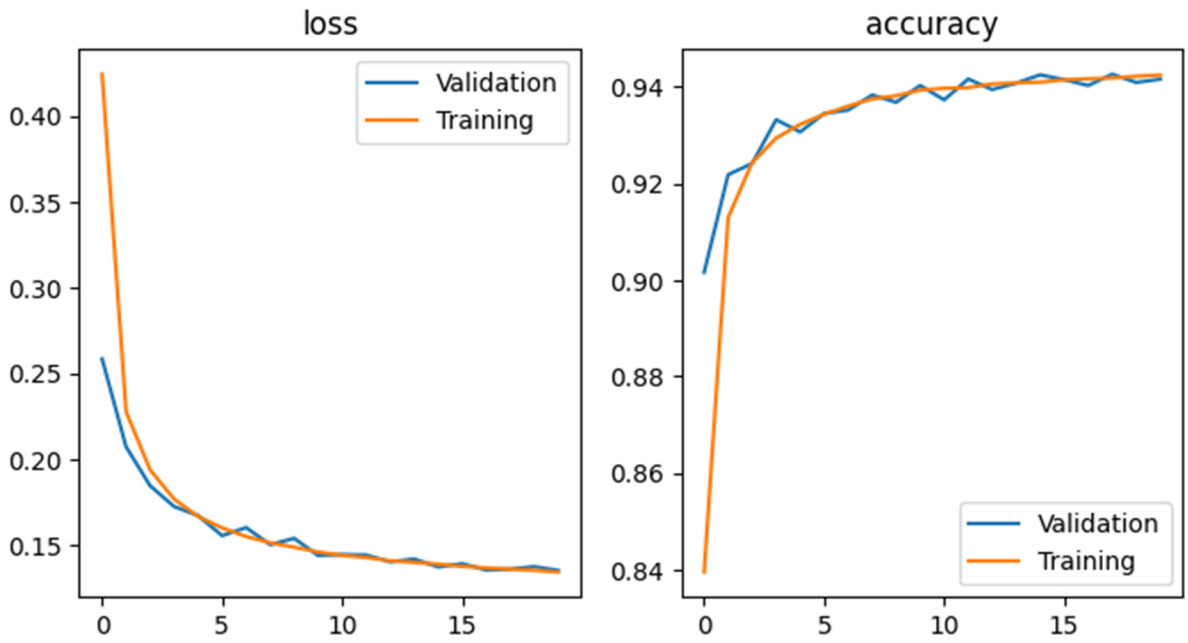


Figure 41. Training History

Hyperparameter tuning

Hyperparameter tuning indeed stands out as one of the most intricate and time-consuming aspects in the construction of a robust Deep Neural Network (DNN) model. Striking the right balance between achieving convergence, minimizing training time, optimizing performance, and efficiently utilizing computational resources poses a formidable challenge. This complexity is further compounded by the absence of one-size-fits-all rules or guidelines applicable across all cases and scenarios.

The quest for an optimal set of hyperparameters requires a delicate compromise, where adjustments made to enhance one aspect may potentially impact others. This involves navigating the vast hyperparameter space to uncover configurations that lead to satisfactory convergence, while keeping an eye on training efficiency and overall model performance.

Adding to the complexity is the opacity of DNNs. These models are often regarded as "black boxes," making it challenging to intuitively interpret how changes in hyperparameters influence their behavior. Unlike traditional Machine Learning models, understanding the inner workings of DNNs demands a deeper level of exploration and analysis.

In order to optimize the performance of the deep neural network (DNN) model, there are several hyperparameter tuning options available. The current architecture consists of three dense layers with 32, 128, and 7 units, respectively, using rectified linear unit (ReLU) activation for the first two layers and softmax activation for the output layer.

A lot of hyperparameter tuning options are available such as

- The learning rate can significantly impact the model's training and convergence. For instance, the current learning rate is set to 0.001, but exploring different learning rates, including higher and lower values, could improve convergence and overall performance.

- The **number of epochs**, **batch size**, and the **architecture** itself, such as the **number of units** in each layer, can be further tuned.
- Techniques like **Grid Search** or **Random Search** can be employed to systematically explore various hyperparameter combinations.
- **Advanced optimization algorithms** or employing **learning rate schedules** may also contribute to enhancing the model's training efficiency and accuracy.

Hyperparameter tuning is a crucial step in the model development process, and experimenting with different configurations is essential to achieve optimal results for the specific task at hand.

Throughout our investigation, we tried multiple methods of hyperparameter tuning in order to optimize the performance of our deep neural network (DNN) model. Regrettably, the pursuit of achieving superior performance compared to Machine Learning, was constrained by limitations in both time and resources during the course of this project. However, the identified challenges serve as valuable insights, paving the way for future enhancements and a deeper exploration of hyperparameter tuning strategies. In the upcoming subchapters, we will systematically address and elaborate on two specific methods that hold promise for refining our model's configuration and performance, thereby providing a foundation for continued advancements in our pursuit of model optimization.

Random Search

```python
# Model to optimize
def create_model(input_dim, learning_rate=0.01, units_layer1=32,     units_layer2=128, units_layer3=128):
    model = Sequential()
    model.add(Dense(units=units_layer1, activation='relu', input_dim=input_dim))
    model.add(Dense(units=units_layer2, activation='relu'))
    model.add(Dense(units=units_layer3, activation='relu'))
    model.add(Dense(units=7, activation='softmax'))

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

    return model
# Specify the hyperparameter search space
param_dist = {
    'units_layer1': randint(16, 64),
    'units_layer2': randint(64, 256),
    'units_layer3': randint(64, 256),
    'learning_rate': uniform(0.0001, 0.2),
    'batch_size': [512, 1024, 10240, 51200],
    'epochs': randint(35, 50, 100)
}
# Set up RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=keras_model,
    param_distributions=param_dist,
    n_iter=10,
    cv=3,
    scoring='accuracy',
    n_jobs=-1
)
```

Python

In our quest to enhance the performance of our neural network model, we conducted a Random Search using the 'scikeras' library in conjunction with ‘Keras’ for Deep Learning. The primary goal was to optimize the accuracy of a Keras Classifier model designed for a multi-class classification task. The hyperparameters under investigation included the number of units in the first three dense layers, as well as the learning rate for the Adam optimizer. Our randomized search consisted of 10 iterations and employed a 3-fold cross-validation strategy to robustly assess the model's generalizability. The randomized search aimed to identify the most effective combination of hyperparameters that

maximizes the accuracy of the model on the training data. The results, encompassing the best hyperparameters and their corresponding accuracy, are crucial for guiding subsequent refinements to the neural network architecture, thereby enhancing its predictive capabilities.

The implementation of a Random Search significantly enhanced the model's performance, yielding an improved accuracy of 0.952. Compared to the initial model with an accuracy of 0.94, this is a step forward but we still didn't match Machine Learning models' performances for example. The fine-tuned hyperparameters discovered through the search process have evidently contributed to the model's enhanced predictive capabilities, underscoring the efficacy of the optimization approach in refining the neural network's performance.

Best Accuracy: 0.952					
batch_size	**epochs**	**learning_rate**	**units_layer1**	**units_layer2**	**units_layer3**
10240	148	0.0075	59	111	112

In contrast to our initial model, we used an additional layer to be used by the Random Search algorithm. The model selected by Random Search incorporates more units in each layer (excluding the last one). This enhancement empowers the model to grasp finer details and explain complex information from the features. The Random Search opted for a higher Learning Rate (0.0075) compared to the default value of 0.001 and extended the training process significantly, spanning 148 Epochs as opposed to the initial 20 Epochs in our first model.

The model obtained from Random Search is described as follows :

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense1 (Dense)              (None, 59)                1121

 dense2 (Dense)              (None, 111)               6660

 dense3 (Dense)              (None, 112)               12544

 dense4 (Dense)              (None, 7)                 791

=================================================================
Total params: 21116 (82.48 KB)
Trainable params: 21116 (82.48 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 42. Model Summary

Learning Rate scheduler

In the pursuit of further optimizing the model's performance, a meticulous exploration of hyperparameters led us to incorporate a dynamic learning rate strategy using the 'ReduceLROnPlateau' callback. Employing the model obtained through the preceding Random Search, our goal was to fine-tune the learning rate during training, adapting it to the evolving landscape of the optimization process. This adaptive approach, coupled with the 'EarlyStopping' callback, aimed to mitigate overfitting and enhance the model's generalization ability. The intricate interplay between these hyperparameter tuning techniques reflects our commitment to refining and tailoring the model to achieve the highest possible performance on the given task.

The following code snippet, illustrate the process described in this section:

```python
dnn = Sequential()
dnn.add(Dense(units=59, activation='relu', name='dense1' ))
dnn.add(Dense(units=111, activation='relu', name='dense2' ))
dnn.add(Dense(units=112, activation='relu', name='dense3'))
dnn.add(Dense(units=7, activation='softmax', name='dense4'))

optimizer = Adam(learning_rate=0.01)
dnn.compile(loss = 'sparse_categorical_crossentropy',
              optimizer = optimizer,
              metrics = ['accuracy'])
# Set up ReduceLROnPlateau callback
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.7, patience=10, min_lr=1e-5, min_delta=0.05)
# Set up Early Stop callback
eraly_stop = EarlyStopping(monitor='val_loss', patience=15)

training_history = dnn.fit(X_train_dnn, y_train_dnn,
                           epochs = 148,
                           batch_size = 10240,
                           validation_data=(X_test_dnn, y_test_dnn),
                           callbacks=[lr_scheduler, eraly_stop])
```

Python

The Learning Rate Scheduler didn't succeed in improving the prediction accuracy as it reached the same accuracy (0.95) obtained in the previous section. Trying different other approaches such as using different optimizers or running Random Search again with different values didn't help improving the model's performance.

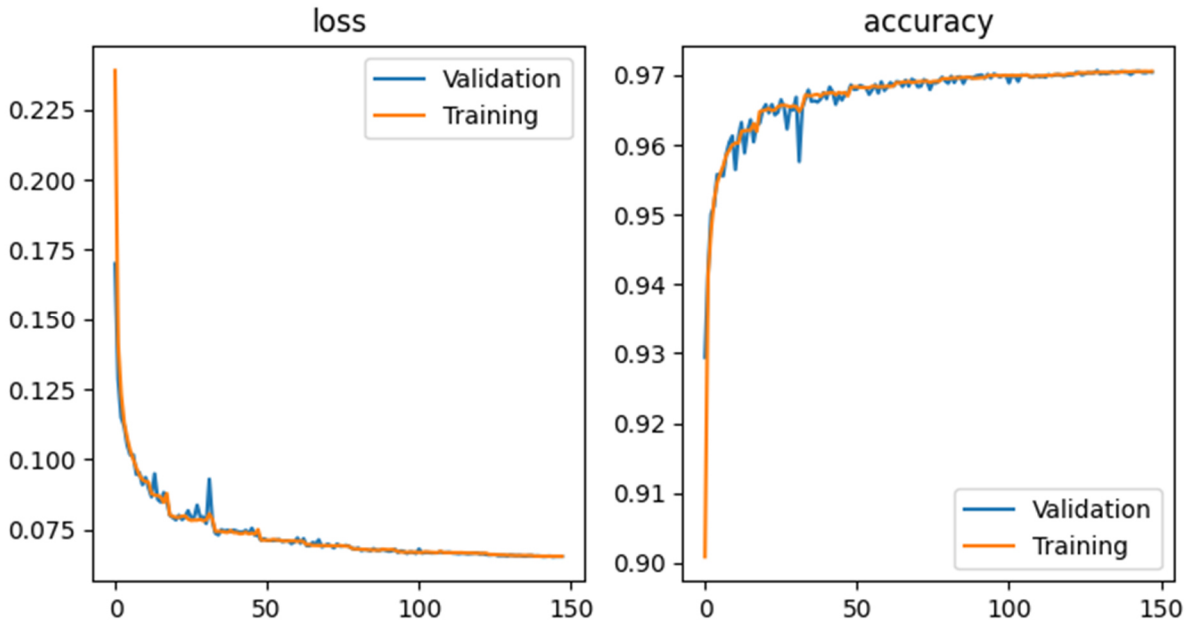


Figure 43. Training History

Oversampling

The findings from the initial dataset highlight the significant challenges posed by imbalanced datasets for both Machine Learning and Deep Learning models. In response to this, we employed oversampling techniques in this section to address class imbalances and enhance our model's accuracy. By utilizing the 'RandomOverSampler,' we expanded our dataset from 1,589,748 to 4,808,811 observations, achieving a balanced representation across all classes. The oversampling, combined with the previously established model, resulted in an improved accuracy of 0.97.

The training history plots show that both accuracy and loss converged. It is also to highlight that training and accuracy curves are following the same trend and keeping track with each other.

Results

At this stage, we used the model selected from the previous section. The model is trained again by splitting the data between validation (3.5%), training (66.5%) and testing (30%).

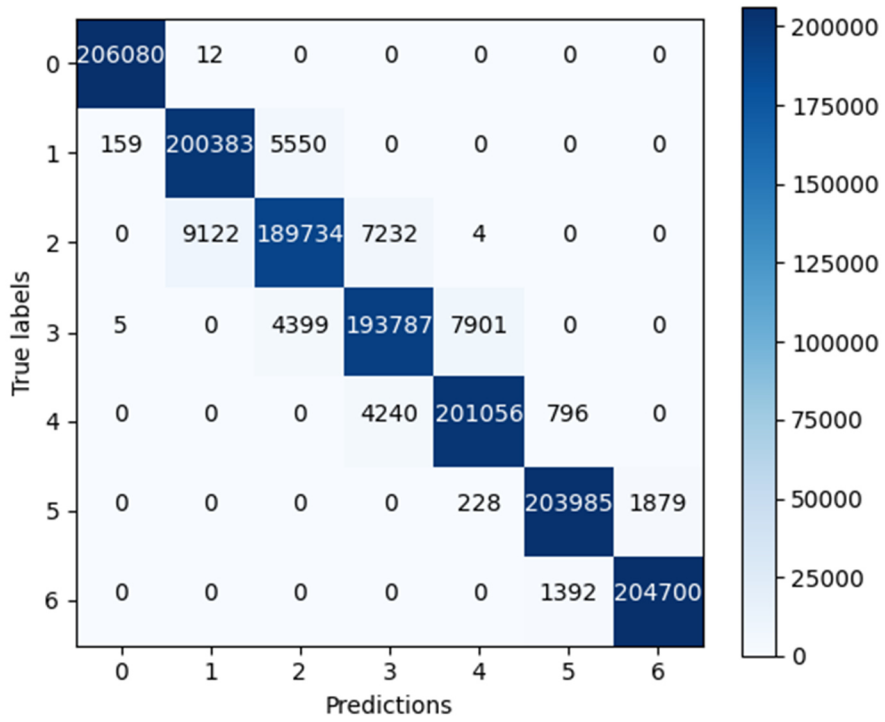


Figure 44. Confusion Matrix

The Confusion Matrix allows us to understand where the model succeeded or struggled the most. It is easy to see that the classes 0, 5 and 6 are the most well predicted. Predictions were not as accurate for the intermediate classes. We could observe that when our model makes errors in predicting a class labeled as N, it consistently tends to mistake it for either N-1 or N+1. This pattern suggests a tendency for the model to be off by one class in either direction, revealing a specific challenge in distinguishing between neighboring classes. It is no big surprise as the classes were obtained from a continuous numerical variable. Distinguishing between vehicles situated at the borders of adjacent classes proves to be a complex task for the model.

Looking at the Classification Report, we are able to confirm some of our previous conclusions.

Class	Precision	Recall	F1-Score
0.0	1.00	1.00	1.00
1.0	0.96	0.97	0.96
2.0	0.95	0.92	0.94
3.0	0.94	0.94	0.94
4.0	0.96	0.98	0.97
5.0	0.99	0.99	0.99
6.0	0.99	0.99	0.99
Accuracy average	0.97	0.97	0.97

Figure 45. Classification Report

Comparison

Having individually examined each model, our next step is to compare them comprehensively. This comparative analysis aims to provide a more holistic perspective, allowing us to understand the strengths and weaknesses of each model.

Model	Accuracy	Mean ROC AUC (OvR)	Mean F1-Score (Macro)
RandomForest	0.97	0.9992	0.9712
XGBoost (Optimized)	0.97	0.9464	0.8852
DecisionTree	0.97	0.9993	0.9712
XGBoost	0.97	0.9991	0.9691
DecisionTree (Optimized)	0.97	0.9992	0.9711
DNN-Modell	0.97	0.9992	0.9696

In this comprehensive evaluation of various classification models for predicting CO^2 emission classes, each model demonstrated a remarkable accuracy of 0.97. Notably, the RandomForest model emerged as a standout performer, boasting near-perfect mean ROC AUC (OvR) of 0.9992 and an impressive mean F1-Score (Macro) of 0.9712. Its ability to precisely classify CO^2 emission classes is evident, although its computational complexity might be a consideration for larger datasets.

The optimized XGBoost model, while maintaining high accuracy of 0.97, showed a slightly lower mean ROC AUC (OvR) of 0.9464 and a good mean F1-Score (Macro) of 0.8852. This suggests a nuanced trade-off between precision and recall, making it a strong contender with computational efficiency in mind.

The DecisionTree model, similar to RandomForest, achieved an excellent mean ROC AUC (OvR) of 0.9993 and a robust mean F1-Score (Macro) of 0.9712. However, DecisionTree models may face challenges related to overfitting and interpretability.

The non-optimized XGBoost model displayed exceptional performance, emphasizing its inherent capabilities for classification tasks, albeit not reaching the same accuracy levels as RandomForest. The optimized DecisionTree model mirrored the performance of RandomForest and non-optimized DecisionTree, showcasing high mean ROC AUC (OvR) and mean F1-Score (Macro).

The DNN model matched the performance of the RandomForest model, with an accuracy of 0.97, a mean ROC AUC (OvR) of 0.9992 and a slightly worse mean F1-Score of 0.9696.

Ultimately, the choice between these models hinges on specific requirements such as interpretability, computational complexity, and the nature of the application. RandomForest and optimized DecisionTree models shine in overall performance, while the optimized XGBoost model strikes a balance between accuracy and efficiency.

The Deep Neural Network (DNN) exhibits a generally robust performance across most classes, similar to the Random Forest and Decision Tree models.

It's noteworthy that the DNN outperforms the optimized XGBoost model in terms of recognizing instances of each class. This emphasizes the DNN's ability to learn complex representations and perform well in multi-class classification tasks.

Overall, the Deep Neural Network demonstrates strong performance across most classes, in comparison to the Random Forest and Decision Tree models.

Analyzing the feature importance across various CO^2 emission classification models reveals both commonalities and differences. 'z_Wh/km' (energy consumption per kilometer) and 'ep_KW' (engine power) emerge as pivotal features consistently across all models, including Random Forest, Decision Tree, XGBoost, and the optimized XGBoost model.

Notably, the optimized Decision Tree model places additional emphasis on features such as 'W_mm' (width) and 'At1_mm' (aspect ratio), indicating the significance of vehicle dimensions in emission classification. This model also highlights 'Ft_HYBRID' (HYBRID fuel type) more strongly.

The XGBoost algorithm, both in its base form and the optimized state, underscores 'z_Wh/km' and 'Ft_OTHER' (other fuel types) as crucial features. It also exhibits higher weighting for 'Ft_DIESEL' (DIESEL fuel type) and 'ec_cm3' (engine capacity in cubic centimeters).

Random Forest accentuates 'ep_KW' and 'm_kg' (vehicle mass), suggesting that both power and weight play pivotal roles.

Overall, these results emphasize that different models identify distinct features as critical for CO^2 emission classification. It is essential to consider these differences and interpret feature importance in the context of each model's architecture.

The reason feature importance was not analyzed for the DNN (Deep Neural Network) model may be attributed to the inherent complexity and interpretability challenges associated with neural networks. DNNs are designed to automatically learn hierarchical representations of data, and the intricate interactions within the network make it challenging to derive straightforward interpretations for the importance of individual features.

In DNNs, the learning process involves adjusting the weights associated with the connections between neurons during training. Analyzing the magnitude of these weights or the impact of specific features on the network's output may not provide a clear or meaningful measure of feature importance, as the network operates in a high-dimensional space with numerous non-linear transformations.

Additionally, DNNs are often chosen for their capacity to capture intricate patterns and relationships in data without the need for explicit feature engineering. The focus is more on learning complex representations rather than isolating the importance of individual features.

Therefore, in the context of DNNs, interpreting feature importance in a manner similar to decision trees or ensemble methods might not be as relevant or informative, and other evaluation metrics and visualization techniques may be more suitable for assessing the model's performance and behavior.

Interpretation

In this section, we will analyze the optimized Random Forest Regression Model to understand how the car properties affect the CO^2 emission. The selected regression model is well suited to use to draw conclusions about the relationship between the features and the target for two main reasons. First, the model has a strong goodness of fit, with a R^2 of 99.69%, meaning that the model can explain 99.69% of the variance of CO^2 emission using the data about the cars. Second, the feature importance of the random forest algorithm is very reliable compared to other algorithms such as Decision Trees. Since Random Forest is an Ensemble Method, it offers stability and resilience to changes in the data. Therefore, the structure of the model, particularly in terms of feature importance, is less likely to be

drastically affected by changes in the data. This enhances the generalizability of the conclusions drawn from the data.

To obtain an overall interpretation of our model and determine which variables have the most predictive power, we will analyze the feature importances. In the case of the 'RandomForestRegressor' in 'sklearn', feature importance is measured using the Mean Decrease in Impurity (MDI). This metric quantifies the improvement achieved by splitting the data based on each variable. The feature importance is calculated by evaluating how much each feature reduces the variance in predictions when used for splitting across all trees in the ensemble. These importances are normalized, ensuring that they add up to one. Therefore, feature importance can be interpreted as the percentage of the total decrease in variance that is contributed by each feature. By examining the feature importance, we can identify the variables that have the most significant impact on predicting CO^2 emissions. Initially we had planned to utilize the 'shap' library for the overall interpretation of our model. This library provides useful tools like the 'TreeExplainer' and 'summary_plot', which allow for visualizing the impact of features on predictions using SHAP values. However, we encountered a significant challenge in the form of the computational expense associated with calculating SHAP values. After dedicating 3000 minutes to the computation, we made the decision to halt the process and use the feature importance for our interpretation instead. While SHAP values offer a more detailed and nuanced understanding of feature contributions, the normalized feature importance provide a suitable alternative for identifying the variables that exert the most significant influence on our CO^2 emissions predictions.

The plot below illustrates the feature importance of the top 10 variables in our dataset.

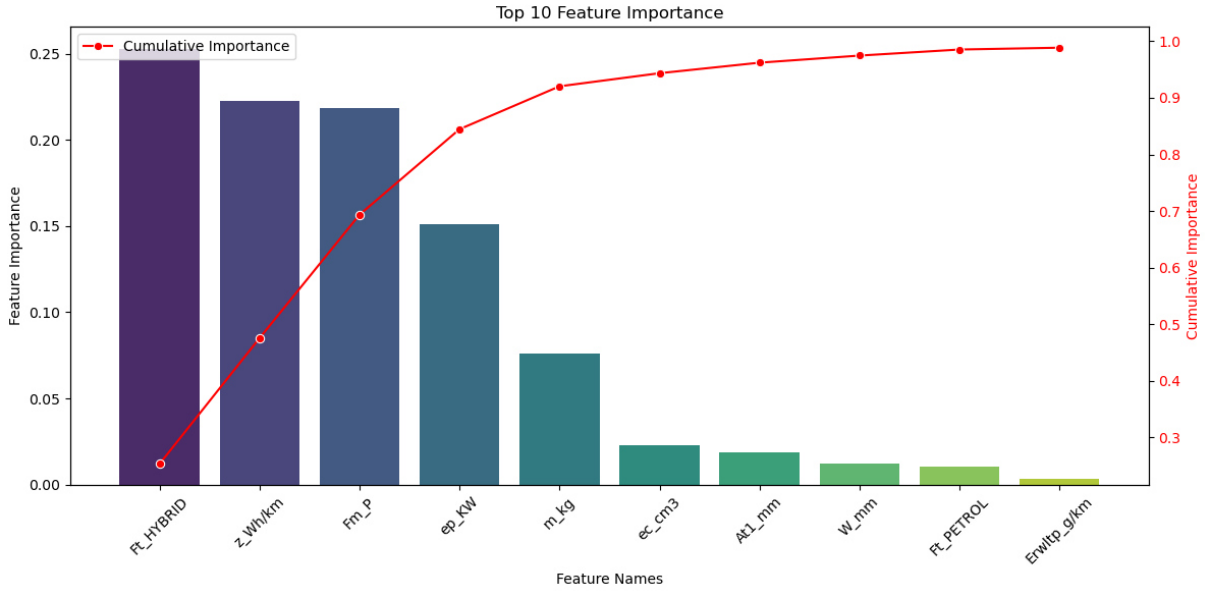


Figure 46. Top 10 Feature Importance and Cumulative Importance

On the left axis, we can observe the scale for the feature importance of the individual features, which are represented by the blue bars. The right axis corresponds to the red curve, which represents the cumulative feature importance. The curve demonstrates a significant decrease in feature importance, as the most important features account for a substantial proportion of the overall feature importance. The top 10 features contribute to 98.87% of the impact on our predictions, indicating that the remaining 18 features have minimal influence. Among the top five features alone, we see that they contribute to 92.04% of the cumulative feature importance, highlighting their significant impact on CO^2 emissions.

The most influential feature is a categorical variable indicating the Fuel Type Hybrid ('Ft_HYBRID') with a feature importance of 25.29%. This is followed by the electric energy consumption ('z_Wh/km') and an indicator variable for the Fuel mode Petrol ('Fm_P') with feature importances of 22.24% and 21.28%, respectively. The engine power ('ep_KW') and car mass ('m_kg') also demonstrate a relatively high impact on the predictions, accounting for 15.11% and 7.58% of the total feature importance. The remaining variables depicted in the plot contribute minimally to the predictions, with feature importance ranging from 2.31% for the axle width of the steering axle in mm ('At1_mm') to 0.32% for the Emissions reduction through innovative technologies in g/km ('Erwltp_g/km'). It is interesting to note that the 'Erwltp_g/km' feature, which directly measures the decrease in CO^2 emissions using innovative technologies, has a relatively small impact on the predictions. This can be attributed to the fact that 28.99% of the cars in the dataset do not utilize innovative technologies, resulting in no decrease in emissions. This reduces the feature importance of the 'Erwltp_g/km' feature.

To examine the influence of the most important categorical features, specifically the fuel type hybrid and the fuel mode petrol, we conducted a boxplot analysis on the training set.

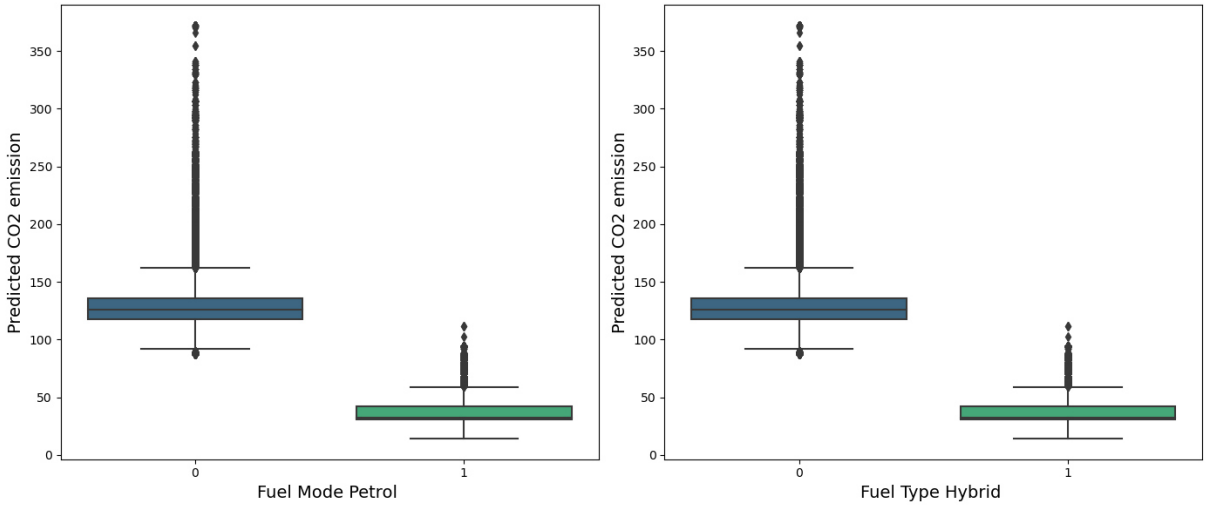


Figure 47. Predicted CO^2 Emission by Fuel Mode Petrol and Fuel Type Hybrid

We observed a significant decrease in the predicted CO^2 emissions for cars falling under these categories. This suggests that the fuel mode petrol and the fuel type hybrid is associated with lower CO^2 emissions. However, it is crucial to clarify that the features themselves do not directly cause the decrease in predicted CO^2 emissions. Instead, the model has learned from the training data that cars with these features tend to exhibit lower CO^2 emissions.

In our analysis of the most important numerical features and their impact on predicted CO^2 emissions, we conducted a correlation analysis on the training set. The electric energy consumption ('z_Wh/km') exhibited a strong negative correlation of -0.79 with CO^2 emissions. To visualize the relationship between these variables, we created a scatterplot.

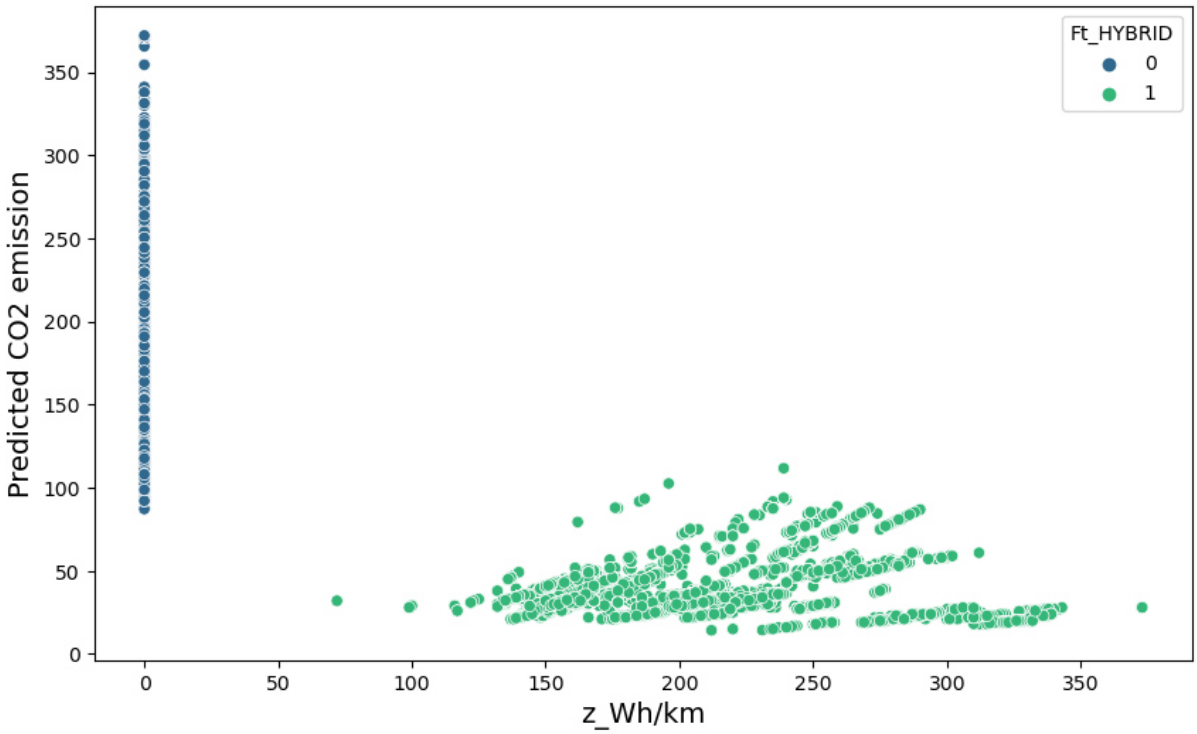


Figure 48. Impact of Electric Energy Consumption on predictions

The scatterplot revealed that cars with no electric energy consumption tended to have higher CO^2 emissions. For cars that did use electricity, there was no clear linear relationship between energy consumption and CO^2 emissions. However, this finding reinforced the understanding that hybrid cars, which utilize electric energy, generally exhibit lower CO^2 emissions.

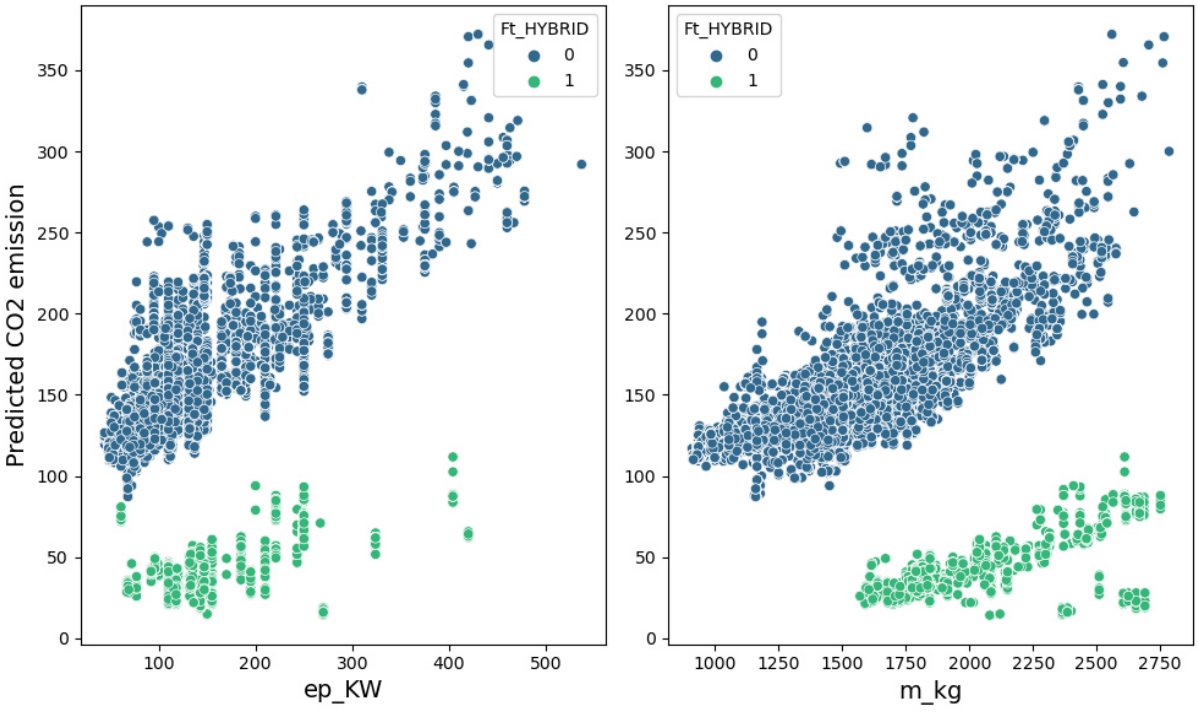


Figure 49. Impact of mass and engine power on predictions

Furthermore, the engine power in KW showed no correlation (0.00) with predicted CO^2 emissions, while car mass demonstrated a weak negative correlation of -0.24. Again, the scatterplots helped explain these results.

Both variables did not demonstrate a linear relationship with CO^2 emissions due to their strong interaction with the fuel type. When examining hybrid and non-hybrid cars separately, there appeared to be linear positive relationships between mass and predicted CO^2 emissions for both groups. Non-hybrid cars exhibited higher predicted CO^2 emissions with increasing engine power, although the relationship between CO^2 emissions and engine power was less clear for hybrid cars. This can be attributed to the fact that hybrid cars have the ability to switch between engine and electric power, which reduces the impact of the engine power variable on the predictions.

Conclusion

Insights & Difficulties

The overall goal of this project is to examine the impact of car properties on CO^2 emissions. To achieve this, we selected a comprehensive dataset from the European Environment Agency (EEA) that encompasses information on all cars registered in France in 2021. Our initial step involved preprocessing the data and analyzing the numerical and categorical features in relation to our target variable. During this phase, we removed irrelevant features, eliminated non-polluting electric cars, addressed missing values, and created dummy variables for categorical data. The WLTP score, which quantifies CO^2 emissions in grams per kilometer, was chosen as our target variable. Furthermore, we employed CO^2-Etiquettes to transform the WLTP Score into a categorical variable for classification purposes.

To analyze the connection between car properties and CO^2 emissions, we trained and compared several classification and regression models. Among the models selected were decision trees, as well as more sophisticated algorithms such as Random Forest (a bagging algorithm), XGBoost (a boosting algorithm), and Deep Neural Networks. These models were chosen for their ability to capture non-linear relationships and provide feature importance for better model interpretation. Additionally, these algorithms are versatile and can be applied to both regression and classification tasks, which we utilized in our analysis.

To find the most optimal models for our data, we conducted a comparison of baseline models for both regression and classification tasks, ultimately selecting the algorithm that performed the best. Our initial challenge involved the baseline models primarily relying on two features – fuel consumption and electric range – for predicting CO^2 emissions. Our findings indicate that fuel consumption is the primary determinant of CO^2 emissions, given its strong correlation ($r = 0.89$). Consequently, it overshadowed all other features in terms of feature importance. When excluding fuel consumption, the electric range became the dominant feature for predictions, likely due to its high correlations with both fuel consumption ($r = -0.81$) and CO^2 emissions (-0.83). Electric range measures the distance hybrid cars can drive solely on electricity, thus providing information about fuel consumption. We made the decision to exclude these two features in order to develop a model that is not overly reliant on a single feature for predictions, allowing for a deeper understanding of the relationship between car properties and CO^2 emissions.

We have successfully identified classification and regression models that exhibit high accuracy in predicting CO^2 emissions, making them suitable for interpretation and understanding the relationship

between the remaining car properties and CO^2 emissions. Our top-performing regression model, Random Forest Regression, achieved an impressive R^2 value of 99.7%. It surpassed the performance of both XGBoost Regressor and Dense Neural Network, which achieved R^2 values of 99.5% and 98.8% respectively. In terms of explaining the variance of the target, the Random Forest Regressor performed similarly to the Decision Tree with an R^2 of 99.7%. However, the Random Forest model demonstrated smaller errors in terms of Root Mean Squared Error (RMSE). Moreover, the Random Forest model is better suited for model interpretation due to its ensemble nature, making it more stable compared to Decision Trees. As a result, the feature importance is less affected by small changes in the data. For the classification task, our best model (Decision Tree) achieved excellent results based on Mean Receiver Operating Characteristic Area Under the Curve (One vs. Rest) with 99.9%. Initially, we encountered challenges with hyperparameter tuning due to the large number of hyperparameters and computational complexity associated with the selected algorithms. However, we were able to overcome this by carefully selecting a smaller parameter grid, resulting in further improvements to the models. We managed to slightly reduce the RMSE of the Random Forest Regression by 0.003 and 0.001. The classification model's performance slightly decreased through hyperparameter tuning, with the Mean ROC (Ovr) and the Mean F1-Score Macro decreasing by 0.01, while maintaining a consistent accuracy. This minimal decline in performance metrics can be attributed to fine-tuning adjustments during the tuning process, having only marginal effects on the overall accuracy of the model.

For the model interpretation, we ultimately chose the regression model over the classification model. While both models performed exceptionally well, we found the regression model to be slightly superior with an impressive R^2 value of 99.7% compared to 97.0% accuracy of the classification model. Additionally, predicting the exact CO^2 emissions in grams per kilometer provides more nuanced insights compared to predicting simplified CO^2 labels. To understand the impact of car properties on CO^2 emissions, we began by analyzing the feature importance. Notably, the hybrid fuel type, electric energy consumption, petrol fuel type, engine power, and mass of the cars emerged as the most significant factors, contributing to 92.04% of the cumulative feature importance among all 28 features. Further analysis revealed that cars with a hybrid fuel type and those running on petrol exhibited significantly lower predicted CO^2 emissions. On the other hand, there was a positive relationship between the mass of the car and engine power, both of which were associated with higher predicted CO^2 emissions. The relationship between electric consumption and predicted CO^2 emissions was not clear-cut, as the variable was strongly influenced by the hybrid fuel type. In conclusion, the data suggests that opting for hybrid cars is a better choice for minimizing CO^2 emissions. Additionally, lighter cars with lower engine power are more favorable in terms of reducing CO^2 emissions. Furthermore, cars running on petrol fuel also perform better in terms of CO^2 emissions reduction.

Continuation

Generally, we succeeded in training a model which accurately predicts CO^2 emissions. Despite this accomplishment, there is room for further improvement. Our initial approach involved utilizing data solely from cars registered in France, as our computational resources were limited. To enhance the model's performance and obtain more comprehensive insights, it is advisable to rerun the script using data from all cars registered within the EU. The expansion of the dataset would lead to a better generalizability of the model. Our approach to model selection involved a pragmatic strategy of comparing baseline models and solely optimizing the best-performing algorithm. This decision was driven by the computationally expensive nature of hyperparameter tuning. However, it is worth considering that we could potentially achieve a superior model by prioritizing parameter optimization before comparing performances. This would be particularly important for the Deep Neural Network, as refining the model's parameters and structure likely would probably have led to improved

outcomes. Additionally, due to limited computational resources and costs, we utilized a compact parameter grid for hyperparameter optimization using Grid Search. Conducting a more exhaustive hyperparameter optimization would likely have resulted in superior results.

In the process of model interpretation, we relied on feature importance because we couldn't calculate the SHAP values in a reasonable amount of time, again due to limited computational resources. Although traditional feature importance methods can provide valuable insights into the significance of features in Machine Learning models, SHAP values offer a more comprehensive, consistent, and interpretable approach to comprehending model predictions. Therefore, utilizing SHAP values would be the preferable option for achieving a deeper understanding of the model's behavior.

A notable limitation of this project is its failure to consider the CO^2 emissions resulting from the electricity consumption of hybrid cars. One of our key findings suggests that hybrid cars are a preferable option for reducing CO^2 emissions. However, this conclusion is misleading since the CO^2 emissions are measured using the WLTP test, which solely accounts for tailpipe emissions and disregards the CO^2 emissions associated with electricity production for hybrid cars. To estimate the CO^2 emissions of electric and hybrid cars, one approach is to utilize the electric energy consumption as an approximation.

The CO^2 emissions (g/km) can be calculated by multiplying the energy consumption (kWh/km) by the CO^2 intensity (g/kWh):

$$CO2\ emissions\ (g/km) = Energy\ consumption\ (kWh/km) \times CO2\ intensity\ (g/kWh)$$

The energy consumption data is provided in the 'z_Wh/km' feature. The CO^2 intensity represents the amount of CO^2 emitted per unit of electricity generated in a specific area (in our case, France) and is typically measured in grams of CO^2 per kilowatt-hour (g/kWh). We could use this estimation for electric cars, which were excluded from our project, and add it to the CO^2 emissions of hybrid cars. However, it is uncertain whether the energy consumption is measured under the same conditions as the WLTP score, potentially introducing bias into this estimation. To make accurate statements about the CO^2 emissions of hybrid and electric cars, ideally the WLTP testing procedure should account for these factors in their scoring system or measure the energy consumption under the same testing conditions, in order to standardize both scores.

Application

In practical applications, our model and insights can have significant value. Firstly, government agencies and policymakers can leverage the insights on the relationship between car properties and CO^2 emissions. Policymakers can develop more effective regulations and incentives to promote cleaner transportation options by gaining a comprehensive understanding of the primary factors contributing to increased CO^2 emissions from cars. Insurance companies can also benefit from these insights by creating tailored insurance products and pricing strategies that encourage customers to choose cleaner vehicles. Educational institutions, such as universities and schools, can utilize the insights to raise awareness about the environmental impact of cars. Our Machine Learning algorithm can serve as a valuable teaching tool for educators, helping students develop a deeper understanding of the factors influencing CO^2 emissions and the significance of sustainable transportation solutions.

Car manufacturers can directly use our model as well. While the WLTP procedure is a more reliable method for measuring CO^2 emissions, it is time-consuming, costly, and causes delays in vehicle development cycles. By regularly retraining the model on new car data and using it for estimation, manufacturers can reduce testing costs and development time. However, it is crucial to consider whether the trade-off between the accuracy of CO^2 emissions and the reduced testing costs is favorable and meets regulatory requirements.

References

https://circabc.europa.eu/sd/a/d9cff59f-5117-48f4-9a37-07b94027110c/MS%20Guidelines%202019

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2, pp. 1-758). New York: springer.

Murphy, K. P. (2022). *Probabilistic machine learning: an introduction*. MIT press.