Accelerating Sustainable Mobility: Empirical Insights into Machine Learning-Based Electric Vehicle Price Prediction

Reece Cavan Moraes

152022011

Dissertation written under the supervision of professor Pedro Afonso Fernandes

Dissertation submitted in partial fulfilment of requirements for the MSc in Business Analytics, at the Universidade Católica Portuguesa, on December 2023.

Accelerating Sustainable Mobility: Empirical Insights into Machine Learning-Based Electric Vehicle Price Prediction

Reece Cavan Moraes

Abstract

This thesis endeavours to discern the primary variables influencing pricing differentials among electric vehicle manufacturers. Leveraging advanced machine learning algorithms, the study scrutinizes the impact of fifteen distinct features that potentially contribute to pricing variations. The models are meticulously trained and constructed on a comprehensive dataset, subsequently tested on an independent sample to ascertain optimal precision and accuracy metrics. While too many studies have successfully implemented this innovative methodology within the domain of internal combustion vehicles, its application to the electric vehicle domain remains a nascent area of inquiry. This pioneering approach, coupled with the evolving landscape of machine learning, holds the promise of delivering dual benefits: affording companies the ability to establish an appropriate pricing spectrum for their vehicles, and providing consumers with access to value-added electric vehicles.

The study incorporates a diverse set of regression techniques, including Multiple Linear Regression, Support Vector Machine, Random Forest Regression, Decision Trees and XGBoost Regression. The target variable under consideration is price, characterized by its continuous nature. Consequently, the utilization and implementation of regression methodologies exclusively aligns with the nature of the output variable.

In the domain of electric vehicles (EVs), research focused on employing machine learning for pricing determination has been relatively limited in its momentum and significance within the automotive industry. Despite the anticipated proliferation of these battery-powered vehicles driven by global imperatives for carbon neutrality, their widespread adoption is still in its early stages in the 21st century.

Keywords: Machine learning; Electric Vehicles; Multiple Linear Regression; Support Vector Machine; Random Forest Regression; Decision Trees; XGBoost Regression.

Resumo

Esta tese procura identificar as principais variáveis que influenciam os diferenciais de preço no consumidor final entre veículos elétricos. Recorrendo a modelos avançados de aprendizagem automática, este estudo analisa o impacto nesse preço de quinze caraterísticas distintas dos referidos veículos. Os modelos foram meticulosamente estimados com recurso a uma base de dados exaustiva e testados numa sub-amostra distinta de modo a otimizar a precisão e a exatidão das predições. Esta metodologia tem vindo a ser aplicada ao estudo dos preços dos veículos tradicionais com motor de combustão interna, mas não tanto no caso dos veículos elétricos. Quando conciliada com a evolução dos métodos de aprendizagem automática, esta abordagem pode, por um lado, suportar o processo de fixação de preços por parte dos fabricantes e, por outro lado, informar os consumidores finais, facilitando o respetivo processo de compra de veículos elétricos. Este estudo incorpora um conjunto de técnicas de regressão, nomeadamente, regressão linear múltipla, vetores de suporte, florestas aleatórias, árvores de decisão e boosting. A variável dependente é o preço no consumidor final, caraterizando-se pelo seu caráter numérico e continuo. Assim, a adoção de técnicas de regressão é coerente com a natureza da variável explicada. No âmbito dos veículos elétricos, a investigação com recurso a métodos de aprendizagem automática é ainda muito limitada. Apesar de se vislumbrar a proliferação deste tipo de veículos por questões de neutralidade carbónica, a sua adoção está ainda numa fase preliminar em pleno século XXI.

*Keywords:*Aprendizagem automática; Veículos elétricos; Regressão linear múltipla; Vetores de suporte; Florestas aleatórias; Árvores de decisão; Regressão XGBoost.

Contents

List of Figures

List of Acronyms

- **EV** - Electric Vehicle
- **PEV** - Pure Electric Vehicle
- **BEV** - Battery operated electric Vehicle
- **HEV** - Hybrid Electric Vehicle
- **FCEV** - Fuel Cell Electric Vehicle
- **PHEV** - Plug-in Hybrid Electric Vehicle
- **ICEV** - Internal Combustion Engine Vehicle
- **NCAP** -New Car Assessment Program
- **MSRP** - Manufacturer's suggested retail price
- **ML** – Machine Learning
- **SL** - Supervised Learning
- **MLR** – Multiple Linear Regression
- **SVR** – Support Vector Regression
- **KNN** - K-Nearest Neighbour
- **MAE** - Mean Absolute Error
- **RMSE** - Root Mean Square Error
- **RBF** - Radial Basis Function
- **CART** - Classification and Regression Trees
- **SUV** - Sport Utility Vehicle
- **IDE** - Integrated Development Environment
- **VPN** - Virtual Private Network
- **UK** - United Kingdom
- **NL** - Netherlands
- **DE** - Deutschland

- **CSV** - Comma Seperated Values
- **NA** - Not Available
- **MCAR** - Missing Completely At Random
- **MAR** - Missing At Random
- **MNAR** - Missing Not At Random
- **AWD** - All Wheel Drive
- **FWD** - Forward Wheel Drive
- **RWD** - Rear Wheel Drive
- **IQR** - Inter Quartile Range
- **KDE** - Kernel Distribution Estimation

1 Introduction

1.1 Overview

In an age characterized by escalating environmental apprehensions, the global agenda is steadfastly gravitating towards the establishment of having a sustainable future. Nations worldwide are gradually adopting an array of measures to curtail carbon emissions, propelled by an unwavering commitment to achieve net-zero environmental impact. At the epicentre of this concerted effort lies the domain of transportation. As governments hasten to phase out conventional gas and diesel engines, the race to establish a formidable presence in this multi-billion-dollar industry has already started.

At the forefront of this transformative shift is the remarkable innovation of electric vehicle technology, signalling a new era of transportation with the promise of cleaner, greener, and more fuel-efficient mobility. Although the first battery powered vehicle was indeed a tricycle developed by Trouvé Kirsch (1995) invented in the 19th century, the use of electric vehicles did not really catch on until the early 21st century due to the high battery costs, low speeds, and low range. Considering recent strides in battery technology and the establishment of robust charging infrastructure, coupled with government-backed incentives, a favorable trajectory is anticipated for the widespread adoption of Electric Vehicles (EVs). Projections indicate an imminent influx of hundreds of thousands of new electric vehicles, encompassing a spectrum from Plug-in Hybrids to fully electric, battery-operated vehicles, set to grace the world's roads in the coming decade. This surge is poised to integrate seamlessly with the existing global fleet, which presently comprises over two million operational electric vehicles Quiros-Tortos et al. (2023).

1.2 Landscape of EV

The developing landscape of Electric Vehicles: The need to transition away from conventional energy resources, namely oil and natural gas, has activated a pivotal shift towards the use of cleaner fuels in vehicular propulsion systems. Noteworthy initiatives, such as Germany's ambitious aim to deploy one million electric vehicles on its roads by 2025, serves as compelling examples of the global commitment towards sustainable transportation. Additionally, nations like France and the United Kingdom have set forth stringent mandates, outlining definitive timelines for the end of combustible engine sales by the close of 2040 Sun et al. (2019). These resolute governmental initiatives, aimed at phasing out gas-powered vehicles, bear significant effect in mitigating CO2 emissions and reducing the prevalence of greenhouse gases in the atmosphere.

While the foundations of EV technology were laid over a century ago, it wasn't until the onset of the 21st century that we witnessed a notable surge in the use of Electric Vehicles. This surge was mainly fuelled by mounting concerns pertaining to environmental

preservation and energy conservation. Within the realm of electric vehicles, a hierarchy emerged, laying them into three distinct categories.

These classifications gave rise to three primary types of EVs: Pure Electric Vehicles (PEVs), commonly referred to as Battery Operated Electric Vehicles (BEVs), Hybrid Electric Vehicles (HEVs), and Fuel Cell Electric Vehicles (FCEVs) Sun et al. (2019). Among these, BEV's hold a preeminent status as the most viable alternative. Distinguished by their reliance solely on battery for power, they limit the necessity for supplementary energy sources. In contrast, Hybrid Electric Vehicles operate on a dual power paradigm, using both an internal combustion engine and a battery charge in tandem. This design is strategically engineered to optimize mileage and enhance overall performance. Finally, Hydrogen powered Electric Vehicles also known as FCEV's diverge from its battery-based counterparts, employing fuel cells in place of traditional batteries. In concurrence with a supercapacitor, the fuel cells harmoniously power the vehicle's drive train, presenting an innovative alternative within the electric vehicle landscape, despite the risks associated with the hydrogen which is highly inflamable.

1.3 The Pricing Conundrum

A comprehensive study conducted in Canada shows a comparative analysis of purchase decisions among HEVs, BEVs, Plug-in Hybrid Electric Vehicles (PHEVs), and Internal Combustion Engine vehicles (ICEVs) across diverse market segments, including economic, intermediate, full-sedan, luxury, minivan, SUV, and pickup. The research revealed noteworthy cost disparities, with BEVs being 51(%) more expensive than their ICEV counterparts, while PHEVs and HEVs carried price premiums of 45(%) and 28(%), respectively Higgins et al. (2017). Within the intermediate segment, respondents who were likely to purchase a vehicle demonstrated a preference for ICEV vehicles, with a 36(%) probability, followed by HEVs at 28(%), PHEVs at 24(%), and BEVs at 12(%).

While the electric vehicle market consists of three distinct categories of BEVs, HEVs and PHEVs, our research exclusively caters the pricing determinants of BEVs. The intricacies of BEV pricing are influenced by a multitude of pivotal factors. Most prominent among these are manufacturing costs, market demands, government-driven incentives, and technological innovations.

In defining the scope of our study, we explicitly aim to isolate the buyer-perceived characteristics from the broader spectrum of influential determinants. Consequently, considerations such as manufacturing costs, market demands, government incentives, and technological advancements, though recognized as pivotal in pricing dynamics, will not be incorporated into our analytical framework. This strategic omission allows us to concentrate our research efforts on identifying the key attributes that resonate with prospective buyers within the world of BEVs.

Within the scope of this study, the determination BEV pricing will be predicated on a comprehensive evaluation of multiple factors. Foremost among these is the pivotal characteristic of an electric vehicle—the size of its battery.

Supplementary attributes to be examined include towing capacity, seating capacity, acceleration metrics (0-100km/hr), top speed, efficiency, fast charging capabilities, real driving range, and drive configuration options, encompassing Front Wheel Drive, Rear Wheel Drive, and All Wheel Drive. Additionally, this analysis will also account for the safety ratings assessed by the New Car Assessment Program (NCAP), cargo volume specifications, turning radius, home charging speed, Horsepower and vehicle segment types consisting of cabriolet, coupe, hatchback, lift back-sedan, SUV, sedan, small passenger van and station estate.

It is noteworthy to emphasize that prior studies, often confined to a restricted set of variables, have predominantly focused on internal combustion engine-powered vehicles. This study, in contrast, aims to bridge this research gap by examining carefully an extensive array of features in the context of BEVs. The objective is to confirm the degree of influence these attributes exert on pricing determinations set by automotive manufacturers. This endeavour holds potential to serve valuable insights into the nuanced interplay between these critical factors and the pricing strategies of BEVs in the modern-day automotive landscape.

1.4 Research Objectives

This thesis leverages a comprehensive dataset sourced from the Electric Vehicle Database `https://bit.ly/3QaXrvy` to embark on an in-depth exploration of the pivotal factors influencing the pricing of Electric Vehicles. Employing a systematic approach encompassing machine learning techniques, statistical analyses, and graphical representations, the study explores the underlying dynamics governing pricing strategies in the global automotive market.

1.4.1 Empowering manufacturers

This research initiative holds high potential to offer invaluable insights to manufacturers. By identifying the salient attributes that resonate with consumers, manufacturers can refine their product offerings and pricing strategies, aligning them with consumer preferences. Moreover, this study enables manufacturers to assess and benchmark their pricing strategies with those of their competitors, fostering a more competitive and consumer-centric approach.

1.4.2 Moving towards sustainable transportation

Furthermore, this investigation extends its scope to the domain of environmental sustainability. By advocating for a transition towards eco-friendly and sustainable transportation models, this research strives to limit the collective carbon footprint associated with vehicular emissions. In promoting for more environmentally conscious consumer choices, the study infuses a shift towards a cleaner and more sustainable mode of transportation.

1.4.3 Government policy implications

The findings of this research also bear significant implications for government bodies. A rise in focus on reducing carbon emissions and embracing eco-friendly technologies may induce governments to institute favourable policies, including tax benefits and other incentives, aimed at bolstering the adoption and sales of BEVs. This, in turn, not only aligns with global sustainability objectives but also bodes well for a more environmentally-conscious and future-oriented transportation landscape.

In light of the essential role of Battery Electric Vehicles (BEVs) in today's automotive environment, this study seeks to tackle specific crucial challenges.

Evaluating Machine Learning Efficacy for Price Prediction: This study is designed to discover the effectiveness of machine learning algorithms in accurately predicting the pricing outcomes of Electric Vehicles (EVs) with the help of dataset consisting of 15 distinct features.

2 Literature Review

The ascent of EVs owes much of its success to the establishment of a robust supporting infrastructure. Previous examinations of BEV's have often delved into the intricate workings of its constituent elements beyond the vehicle itself. Extensive literature exists on projections related to charging infrastructure and energy consumption patterns. A notable example is the paper on "Electric Charging Power Demand in Urban Traffic Areas" authored by Arias et al. (2017). This scholarly paper provides a comprehensive overview on the use of temporal-spatial models for forecasting EV power charging demands within urban domains.

Researchers have increasingly turned to using Deep Neural Network models to recognize strategies for substantial reductions in charging costs. By harnessing the power of historical datasets, studies have been able to yield optimal decision-making frameworks Lopez et al. (2019). The research offers insights into how Deep Neural Network models can be effectively employed to make informed and cost-effective decisions regarding EV charging. Despite their critical relevance, such factors are frequently overlooked in empirical investigations of EV pricing. However, they play a pivotal role for end consumers contemplating the transition from conventional gas-powered engines to EVs.

In the pursuit of mitigating charging expenses and improving user convenience, EV manufacturers are swiftly deploying an extensive network of fast charging stations. It is widely recognized that the need for superfast chargers is paramount, particularly in vital situations where individuals cannot afford long charging durations of 30 to 60 minutes to achieve full battery charge. This urgency is particularly highlighted during emergenciesUllah et al. (2022)

Considerations also apply to long journeys, where drivers are compelled to meticulously plan routes, prioritizing access to superfast charging facilities. Empirical evidence suggests that drivers typically gravitate towards charging points located at commercial centers, workplaces, and residences, each exhibiting varying charging durations of 30 minutes, 3.8 hours, and 9.4 hours respectively Ullah et al. (2022).

One of the paramount concerns impeding the adoption of EVs among potential consumers is the apprehension surrounding driving range limitations Krishnan and Koshy (2021). To reduce range-related anxieties, consumers are willing to incurring additional costs. When factoring in expenses associated with expanding driving range, it has been observed that a significant segment of consumers demonstrates a willingness to invest a substantial premium beyond standard pricing for an extended battery range. This phenomenon exerts a clear impact on the overall purchase price of the vehicle, as consumers display an inclination to pay a premium for the need of an extended driving range and the alleviation of range-related anxieties.

Research also explains that consumers are inclined to allocate a considerable sum of

approximately ($)5600 for the implementation of fast charging capabilities, for a reduction of recharging time from 10 hours to a mere 1 hour Hidrue et al. (2011). Additionally, it is found that EV drivers manifest a willingness to invest an additional ($)35 to ($)75 for every added mile of driving range, further underscoring the pivotal role of driving range and battery size extension in consumer purchasing decisions.

The pivotal determinants of driving range and battery size will continue to hold paramount significance for manufacturers in the development and construction of BEVs. The empirical evidence by Liu et al. (2021) justifies that the battery component clearly constitutes a substantial 70(%)proportion of the total vehicular cost. While these factors yield substantial influence over the ultimate price point, the briskness with which EV manufacturers can advance their research and development in these domains remains a subject of scrutiny.

Numerous studies have consistently highlighted the reluctance towards overnight charging as a prevalent sentiment among potential EV buyers. The prospect of connecting one's vehicle to a power socket overnight is met with reluctance, constituting a noteworthy concern for a significant unit of prospective buyers Krishnan and Koshy (2021). This worry, rooted in the aversion to frequent recharging, stands as a prominent barrier in the EV adoption landscape. This inconvenience potentially deters prospective buyers from considering BEVs as a viable transportation option Krishnan and Koshy (2021). It is noteworthy to understand that recharging durations have witnessed substantial drop over the past half-decade. Although the convenience of gas refuelling still outpaces recharging times, the gradual improvements in battery recharge rates, signifies advancement towards the use of electric mobility.

While the pricing of EVs is undeniably tied to factors like battery capacity and charging infrastructure, a host of other elements wield significant influence over their cost. As the global market for EVs experiences an unprecedented surge in sales and manufacturing, it is imperative to acknowledge that pricing strategies may diverge significantly across nations.

For instance, Norway stands as a noteworthy example of a nation offering generous incentives for BEV purchases. In Norway, EV owners are exempt from a spectrum of taxes including registration tax, value-added tax, annual car tax, road tolls, and congestion charges Ajanovic and Haas (2016). Turning attention to India, a country boasting the world's third-largest vehicular network, the fiscal year of 2023 has witnessed an estimated 326.3 million vehicles in active operation Sun (2023). According to Deloitte's research , the Indian automotive market presents a dichotomy. A lower class of cost-conscious consumers seeks low-cost, minimally technological vehicles, while a significant segment of the new wealthy customers gravitates towards more luxurious brands Deloitte (2020). In developed regions, driver priorities are projected to pivot from fashionable add-ons towards features that enhance vehicular safety and efficiency Deloitte (2020). However, amidst this evolving

landscape, the paramount determinant will invariably remain price. Thus, automakers specializing in greener propulsion technology face the difficulty of aligning the pricing of these vehicles with the more established, internal combustion counterparts.

The strategic pricing for BEVs is pivotal to facilitate a meaningful comparison with internal combustion engines. While research highlights that approximately 70(%), of powertrain costs for BEVs is attributed to its battery packs further investigations reveal that an EV equipped with a fully electric powertrain commands an additional expenditure of ($)8630 in comparison to an ICEV Liu et al. (2021).

Furthermore, it is noticed that long-range BEVs exhibit a notable pricing differential, with costs ranging from ($)13,800 to ($)22,300 higher than their ICEV counterparts Liu et al. (2021). As per fastest-car report Staats (2023), a high-performance vehicle with a battery powertrain will set you back anywhere between ($)52,990 and ($)249,000. These findings signify the need for astute pricing strategies that encompass a diverse spectrum of options across varying price brackets.

While it is acknowledged that not all drivers prioritize owning high-performance, high-cost vehicles, empirical evidence suggests a pivotal relationship between driving speeds and BEV efficiency. Notably, maintaining steady speeds in BEVs has been observed to yield substantial gains in battery conservation, thereby enhancing driving range. The study also reveals that BEVs exhibit cost-effectiveness in conditions characterized by frequent stops and starts, as opposed to sustained high-speed travel on highways Galvin (2017). Furthermore, it is found that rapid accelerations exert a high strain on energy resources.

The variation in pricing across Battery Electric Vehicles (BEVs) can further be studied by considering the inclusion of safety features. Notably, within the European Union (EU), recent mandates stipulate the obligatory provision of airbags, including driver airbags, as standard safety components in all new vehicles Baltas and Saridakis (2009). The research further highlights that the incorporation of safety attributes such as airbags and electronic safety systems (EBS) present as highly influential factors, as evidenced by statistically significant coefficients at the 0.01 level of significance, accompanied by low standard errors, when assessing its impact on vehicle pricing Baltas and Saridakis (2009).

Moreover, the study employs hedonic regression as a scientific approach for estimating the implicit value of various car characteristics. In doing so, it emphasizes the existence of a positive economic valuation associated with environmental and safety attributes within the United States automotive market.

Another notable research conducted with respect to safety for vehicles in Kuala Lumpur, Malaysia, sought to understand the determinants impacting vehicle buying behaviour Lee et al. (2014). Employing a sample size of 200 respondents, the study aimed to present valuable insights for achieving a competitive edge within the Malaysian automotive market. The research integrated statistical analysis alongside graphical representations to

investigate consumer preferences and purchasing patterns.

The research adopted a straightforward yet robust statistical tool, employing simple linear regression. This analytical approach assessed safety as an independent variable, seeking to gauge its influence on consumer buying behaviour.

Malaysian Institute of Road Safety Research (MIROS) led safety initiatives in the automotive industry, mandating critical safety features like airbags, auto-lock systems, car sensors, and ABS. This directive notably saved around 6200 lives. The study found that consumer preferences leaned towards vehicles with comprehensive safety features, with a willingness to pay more for enhanced safety attributes. The study quantitatively showed that about 2.6(%) of variance in consumer purchase behavior relates to a vehicle's safety aspect.

The segmentation of vehicles into distinct classes or categories stands as another salient determinant significantly influencing vehicle pricing. A study conducted in China, focusing on conventionally powered vehicles, sought to explain the pricing patterns across various vehicle sizes, encompassing SUVs, Crossovers, and Sedans Ou et al. (2020).

The findings of this study revealed substantial variation in sedan pricing across different countries, despite having similar features. For instance, a midsize sedan from Japan exhibited a staggering 62(%) higher price tag compared to its Chinese counterpart. Similarly, in other international markets, the manufacturer's suggested retail price (MSRP) of a European SUV commanded an 82(%) premium over that of a Chinese SUV.

The study harnessed the power of Monte Carlo simulation to quantitatively assess potential distributions of MSRP. This was achieved by assuming variable feature ranges based on historical distributions. Subsequently, a linear multinomial regression model was developed, incorporating dummy variables to project vehicle MSRP. The model made use of vehicle attributes such as engine power, engine size, fuel efficiency, and vehicle weight. Additionally, it considered categorical distinctions including brand origin and vehicle segment, that were divided into sedan or SUV/crossover.

Employing a Multivariate Analysis of Variance (MANOVA) model, a study conducted by Higgins et al. (2017) illuminated significant between-subject effects within distinct market segments, categorized by preferred vehicle body type. Notably, the model indicated that BEVs had the highest appeal in the luxury and SUV markets, as evidenced by the magnitude of the constants.

In an evolving landscape where consumers place paramount importance on the value proposition of BEVs, manufacturers are tasked with making strategic decisions to ensure their offerings align with customer expectations and justify their price point Furcher et al. (2022). This necessitates a continuous commitment to innovation within the EV sector, as shifting consumer attitudes and preferences pave the way for new market entrants. Critical considerations such as transparent pricing structures and seamless purchasing experiences across diverse dealership networks should command the highest priority Furcher et al.

(2022).

While acknowledging potential minor drawbacks such as those of transparent pricing or the developing charging infrastructure, a huge global demand for electric vehicles persists. Driven by an unwavering commitment to boost economic sustainability, environmental conservation, and leveraging realistic technological advancements, extensive research indicates a compelling trend. Figenbaum et al. (2014) study highlights that an overwhelming 90(%) of households currently in possession of gasoline-powered vehicles express a distinct preference for Electric Vehicles as a viable option as their secondary vehicle.

2.1 Research within the automotive domain

While our research focuses on elucidating the pricing dynamics of newly released electric vehicles, it is noteworthy that much of the existing research predominantly centres' around old used ICE vehicles.

A study conducted by Mammadov (2021) to determine the pricing of gasoline-powered vehicles in the US market provides noteworthy insights. The research selected carwidth, curbweight, enginesize, horsepower, and citympg as the key features, employing Linear Regression due to their observed high linearity with price. The model exhibited a remarkable performance, achieving an R-square of 97(%) on the training set and 83(%) on the test set. Recursive feature elimination was utilized to address overfitting and manage dimensionality. Despite exclusively employing Linear Regression, the study concluded that none of the included variables exhibited a polynomial relationship with size.

Noor and Jan in 2017 Noor and Jan (2017) analysed a dataset comprising 2000 records of pre-owned ICE vehicles. This dataset encompassed critical variables including price, engine capacity, color, advertisement date, number of views, mileage in kilometers, power steering, alloy rims, transmission, type of engine, registered city, city, version, model, make, and model year. Following data preprocessing, a multiple linear regression model was employed with price as the dependent variable. The model demonstrated a high level of measure, yielding an R-squared value of 98.5(%). The research emphasized the significance of engine type and car model within the dataset. However, while the study achieved a high R-squared, it did not undertake a comparative assessment with other machine learning models.

In a 2019 study on pricing impacts of car characteristics in Bosnia and Herzegovina using a dataset of used cars, Gegic et al. (2019) employed diverse features like brand, model, condition, fuel type, age, kilowatts, and more to train their model. They tested three machine learning methods (ANN, SVM, Random Forest) both individually and as an ensemble. While a single model achieved 50(%) prediction accuracy, the ensemble, after preprocessing, soared to 92(%). However, this improved accuracy came at the cost

of increased computational resources compared to a single model.

The findings by Prof. Sejal Thakkar, Mital Donga, Johny Panjwari, Dharmil Kotak, and Kresha Savalia (2023) Thakkar1 et al. (2023) offer valuable insights into the used electric vehicle marketplace. The project leveraged a dataset comprising of 5500 records to provide an overview of price prediction for BEV's utilizing Linear Regression. Through their analysis, linear regression emerged as a reliable and accurate technique for price prediction, demonstrating an accuracy range of 78(%) to 86.5(%). However, it is worth noting a limitation in the study's approach, as it incorporated only a limited set of variables such as vehicle manufacturer, range, condition, make, and model. The paper also highlights the importance of incorporating additional features and employing advanced techniques like deep learning to further enhance prediction accuracy.

Samruddhi, K., and Ashok Kumar, R. (2020) Samruddhi and Kumar (2020) provides significant insights through their research paper, into the process of variable selection for determining vehicle price. The study encompassed a dataset of used cars comprising 14 variables, covering factors such as unnamed serial number, Name, Location, Mileage, Fuel Type, Engine, Transmission, Kilometers Driven, Power, New Price, Year, Seats, Owner Type, and Price. The paper proposed the application of a supervised learning model, specifically employing the K-Nearest Neighbour (KNN) algorithm. The model's accuracy was estimated across varying k values, ranging from 2 to 10. Moreover, the performance of the model was validated through both 5-fold and 10-fold cross-validation techniques. The highest level of measure obtained was an R-squared value of 82(%), achieved under 10-fold cross-validation, utilizing a k value of 4.

Research conducted by Sameerchand Pudaruth (2013) Pudaruth (2014) offers valuable insights into the determination of car prices in the context of Mauritius. The study demonstrates the application of various techniques, including multiple linear regression analysis, KNN, naïve Bayes, and decision trees, for the purpose of price prediction. The dataset used in the study comprised of attributes such as brand name, model, horsepower, mileage, year of manufacture, paint color, transmission type (manual/automatic), and the corresponding price. The results indicated that the accuracy of the naïve Bayes algorithm varied within the range of 60(%) to 70(%) across different parameter combinations. The study, however, highlighted the significance of augmenting the dataset with a larger volume of records and implementing advanced methodologies such as neural networks, fuzzy logic, and genetic algorithms to enhance the precision of price predictions.

3 Resources

3.1 Coding Language, Libraries & Software

In navigating the evolving landscape of this research, several procedural strides were necessary, spanning from the acquisition of the EV dataset to the eventual model prediction phase. The research methodology adhered to established guidelines, drawing insights from the literature review. Given the convenience of consolidating multifaceted tasks within a unified environment, Python was the preferred programming language throughout the research lifecycle. The language's utilization extended from dataset extraction and cleaning to data visualization and the execution of machine learning models.

3.1.1 Coding Language and Libraries

Python's extensive library ecosystem, including NumPy, pandas, matplotlib, seaborn, sklearn, statsmodels, SciPy, BeautifulSoup, requests, and time, was crucial for streamlining tasks in this research. Leveraging these libraries improved operational efficiency, eliminated repetitive code, and capitalized on pre-built computations, promoting code reusability and reducing workload.

3.1.2 Python IDE

PyCharm, an integrated development environment (IDE), served as the coding platform for its array of essential tools tailored for data science development. Leveraging PyCharm offered advantages inherent in its intelligent Python assistance, featuring functionalities such as smart code completion, code inspection through error-highlighting, suggestions for quick fixes, and automated code refactoring capabilities.

3.1.3 Other softwares

Proton VPN, a virtual private network platform, safeguarded our online identity by hiding the computer's IP address during data extraction. This proactive measure helped evade potential access blocks to the webpage and prevented disruptions during scraping. It also enabled iterative IP address changes for each dataset, reducing the risk of website blocks.

4 Methodology

4.1 The Integration of Machine Learning in the EV Industry

Machine learning (ML) represents a significant branch of scientific research, addressing two closely intertwined studies: firstly, how systems can autonomously enhance their performance, and secondly, the underlying statistical or theoretical principles guiding learning algorithms, surrounding both human and corporate entities Jordan and Mitchell (2015). Over recent years, machine learning has emerged as a force in the world of technological advancement. Corporations are leveraging this nifty toolset to make more judicious and data-driven decisions, thereby mitigating risks and enhancing predictive capabilities. Notably, machine learning has become the preferred instrument for formulating software solutions in domains like computer vision, natural language processing (NLP), image recognition, and others Jordan and Mitchell (2015).

Machine learning operates on the fundamental principle of partitioning datasets into distinct training and testing subsets. In this process, the model is initially exposed to a comprehensive set of input-output variables for training. Subsequently, the model's effectiveness is assessed using the reserved test set, wherein various metric scores are employed to gauge its accuracy Jordan and Mitchell (2015). The resultant predictions are then compared against actual outcomes, yielding a precise measure of the model's performance.

In the context of EVs, particularly the BEVs, the integration of Machine Learning techniques for price determination is a relatively new field. Notably, the surge in EV sales commenced merely half a decade ago. According to the IEA Global EV Outlook 2023 report Outlook (2023), the global sales figures in the past 5 years experienced a remarkable surge, jumping from approximately 1 million to over 10 million. A comparative analysis over a five-year span, from 2012 to 2017, highlights the trajectory, with sales escalating from 100,000 to 1 million units. This progression emphasizes that the annual growth rate of EV sales saw a remarkable surge in the most recent years. As a result, the availability of extensive and comprehensive data pertaining to this domain remains somewhat limited.

Hence, considering this data scarcity, an alternative approach was needed, necessitating an exploration beyond the field of BEVs and into other propulsion systems. Notably, extensive research has ventured into employing Machine Learning methodologies within the domain of ICEVs, providing a valuable collection of insights. In this context, our study aims to draw upon the knowledge gained from ICEV studies and adapt relevant determinants to the unique context of BEVs, aligning variables with our dataset for a comprehensive analysis.

It is crucial to emphasize that due to the differences in gas powered vehicles to battery powered vehicles, not all factors pertaining to ICEVs can be seamlessly implemented to

our study.

Given the focus is on determining the price of new BEVs, our approach will hinge on the use of Supervised Learning models. Specifically, we will employ techniques like Multiple Linear Regression, Decision Trees, Support Vector Machines, XGBoost Regression and Ensemble Learning coupled with Bootstrap Aggregating - Random Forest algorithm. This selection of models is strategically tailored to yield a comprehensive and robust analysis of BEV pricing.

4.2 Key concepts to consider

Supervised learning, a pivotal facet of machine learning, encompasses a process wherein it establishes a mapping between input and output variables. This is achieved using labelled data, consisting of input-output pairs, during the training phase. Within this model, the training dataset is characterized by a dependent variable necessitating prediction or classification given the attributes.

4.2.1 Decision Tree:

A decision tree represents a graphical depiction with a series of choices and their respective outcomes, taking on the form of a hierarchical tree structure. It has nodes and edges, wherein each node denotes specific attributes, while the edges depict the decision rules or conditions. This design facilitates the process of classification and regression tasks through an iterative, rule-based approach.

The mechanism of a decision tree necessitates careful examination. It initiates by partitioning the multi-dimensional space of joint predictor variable values, denoted as x, into J distinct regions. Each region, denoted as Rj, is associated with a corresponding predicted dependent variable, y hat j. Subsequently, for any given set of joint predictor values x, the decision tree predicts

$$\hat{y} = T_j(x) \tag{1}$$

signifying the assigned value in the region containing x. Optimal response values for each set of regions are readily derived, typically representing values that minimize prediction risk within the respective region Friedman (2006).

$$\hat{y}_j = \arg\min_{y'} E_y[L(y, y') \,|\, \mathbf{x} \in R_j].$$

4.2.2 Support Vector machine:

SVM is a powerful supervised learning algorithm that is used for classification and regression tasks. Unlike traditional linear models, SVM can handle non-linear relationships by

employing a mathematical construct called a kernel. This allows SVM to map data into higher-dimensional spaces, revealing complex patterns. SVM strives to establish optimal class boundaries with maximum separation and thereby minimizing classification errors.

SVM Kernels- The preeminent and extensively applied kernel within Support Vector Machines (SVM) is the Radial Basis Function (RBF). This kernel exhibits a notable resemblance to the Gaussian distribution. For two points,

$$X_1, X_2$$

, the RBF kernel function quantifies their similarity or proximity Sreenivasa (2020). The maximum value that the RBF kernel can be is 1.

4.2.3 Ensemble learning:

Ensemble learning is a technique that leverages the collective intelligence of multiple models to address complex computational challenges. By strategically integrating these models, ensemble learning aims to enhance overall performance and mitigate errors, leading to increased accuracy in predictions or classifications. This approach offers notable benefits, including the identification of salient features, the capacity for incremental learning, and automated error rectification.

4.2.4 Bagging:

Bagging, a component of ensemble learning, is instrumental in enhancing the precision and stability of machine learning algorithms, rendering it applicable to a spectrum of tasks encompassing both classification and regression domains. A major advantage of employing bagging lies in its proficiency in reducing variance and mitigating overfitting, thereby boosting the robustness and reliability of predictive models.

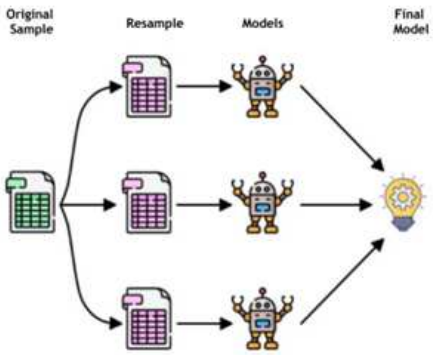


Figure 1: Bagging

Algorithm Steps –

1. Generate N random samples of training data with replacement.

2. Train a regression function using each resample. Calculate the average on predicted values.

4.2.5 Random Forest Regression:

This methodology constitutes a pivotal element of the bagging technique, predominantly employing bootstrap aggregation in conjunction with tree learners. Given a training set represented by,

$$X = x_1, ..., x_n \tag{2}$$

and a response variable

$$Y = y_1, ..., y_n \tag{3}$$

the process of bagging systematically selects, with replacement, random samples from the training set B times. Random forest selects the regressor X randomly from the full pool of attributes. Subsequently, trees are fitted to these selected samples iteratively. Taddy (2019).

For b = 1, ..., B:

1. Sample, with replacement, n training examples from X, Y; call these Xb, Yb.
2. Train a classification or regression tree fb on Xb, Yb.

Following the completion of computations, predictions for unseen samples (x') are typically generated by aggregating the predictions from individual regression trees.

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x')$$

4.2.6 XGBoost:

XGBoost is structured on the premise of sequentially constructing D classification and regression trees. Each subsequent model (tree) is trained using the residuals of the preceding tree, allowing the new model to rectify errors made by the previously trained tree and consequently enhance prediction accuracy. In the context of XGBoost, each ensemble model utilizes the summation of D functions to predict the output Pesantez-Narvaez et al. (2019).

4.3 Metrics

4.3.1 Description

1. R-Squared (R2) – The coefficient of determination also known as R-squared is the percent of variance explained by the model. It is simply the correlation between

the dependent variable and the regression models predictions. It is essential to note that while R-squared serves as an indicator of how well the model fits the data, it does not intrinsically measure accuracy or provide information about the predictive errors.

2. Root Mean Square Error (RMSE)- RMSE quantifies the average magnitude of the discrepancies between observed values and the predictions generated by a model. This metric computes the square of individual errors, averaging them before taking the square root. As a result, RMSE amplifies the impact of larger errors due to the squaring process, providing a comprehensive view of the model's performance.

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

3. Mean Absolute Error (MAE) – MAE serves as a metric to gauge the average magnitude of errors between predicted and observed values in a set of forecasts. Unlike other metrics, MAE disregards the direction of errors, focusing solely on their absolute values. It is a linear score which implies that all individual differences are weighted equally on an average.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right| = \frac{1}{n}\sum_{i=1}^{n}|e_i|$$

5 Data

5.1 Dataset Description

Brand- Manufacturer or a company under a particular name
Model - Vehicle name or the version
Battery size - Capacity of the usable battery in kWh
Towing capacity - Weight the car can pull in Kg
Seats – Number of seats (4,5 & 7 seaters)
Acceleration – Time from 0 to 100km/h in seconds
Top speed – Maximum achievable speed in km/h
Efficiency – Energy used per km measured in Wh/km
Fast charge speed – Speed for charge of battery measured in km/h
Real range – Manufacturers estimated driving range with a full battery in km.
Drive type - All Wheel Drive, Rear Wheel or Forward Wheel Drive
Safety Rating - Euro NCAP safety rating from 0 to 5
Cargo Volume - Cargo space in Litres
Turning Radius - Space needed to make a full 360-degree turn measured in meters
Home Charge Speed - Speed at which battery charges at home measured in km/h
Horsepower- measurement of the force produced from a vehicle's engine in PS
Vehicle Segment - Sedan, Cabriolet, Coupe, Hatchback, Lift-back Sedan, SUV, Small Passenger Van, Station Estate
Price UK - Price of the vehicle in British Pounds
Price NL - Price of the vehicle in European Euros (Netherlands)
Price DE - Price of the vehicle in European Euros (Germany)

5.2 Data Extraction

The data acquisition phase necessitated a ground-up approach, given the limited availability of datasets pertaining to new electric vehicles. In response to this challenge, web scraping emerged as the instrumental technique for data procurement. Web scraping entails the extraction of specific data from online sources, with the collected information typically stored in a local database for subsequent analysis. This methodology operates on the principle of deploying web crawlers to systematically retrieve and extract targeted content. Web crawlers function discreetly, navigating online systems without detection, consuming resources judiciously. The process commences by providing a URL to the crawler, directing it to the specific web location from which the desired data is to be extracted.

The Python requests library functioning as a web crawler interacted with the URL `https://bit.ly/3QaXrvy` using the GET method, storing data in a variable named "re-

sults." The Beautiful Soup package parsed the HTML content, transforming the information into a hierarchical Python object structure. This parsed data, presented as a tree-like representation, allowed for easy search and iteration using Python loops, simplifying HTML exploration with intuitive Python code.

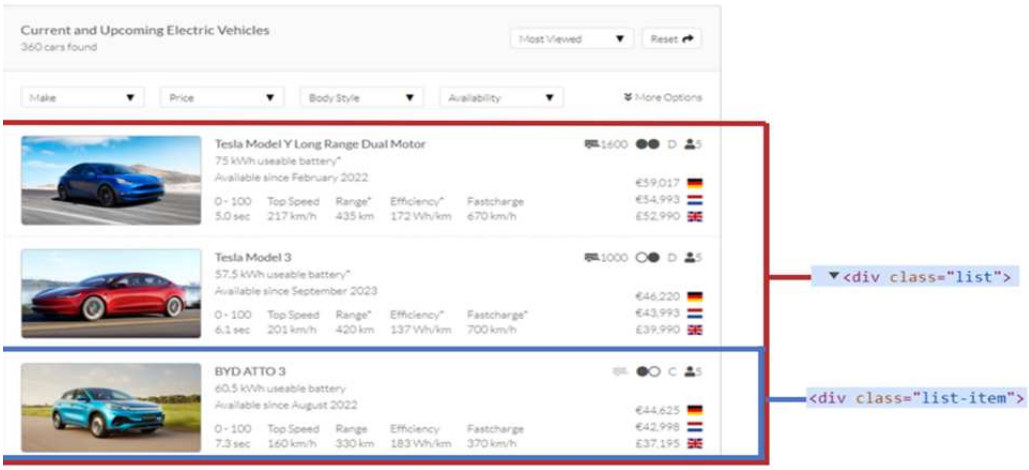


Figure 2: Main page car listings on the website (Left). Div tag with class name (Right)

At the time of scraping a total of 367 newly introduced Electric Vehicle (EV) models were listed on the website. The data extraction phase involved a two-stage process. Initially, visible data from the website's main page was targeted. The first web scraping iteration focused on capturing variables like Brand, Model, Battery Size, Towing Capacity, Seats, Acceleration, Top Speed, Efficiency, Fast Charge Speed, Real Range, Drive Type, and pricing for the UK, Netherlands, and Germany, depicted in Figure 2.

The structural layout and design of the website can be viewed from Figure 2. The HTML code underlying the website can be accessed through any web browser by right-clicking on the page and selecting "Inspect Element." Upon examination, it was observed that the details of all 367 listed cars were encapsulated within a primary div tag labelled "list," with each individual car subsequently nested within distinct div tags denoted by the class name "list-item." Exploiting the structural uniqueness of class names facilitated a straightforward parsing process through this information. By utilizing the .find(_)all method of Beautiful Soup, all cars identified through the class name "list-item" were collected and stored in a new variable designated as "car(_)list." Subsequently, this list underwent a for loop iteration, systematically extracting the desired variables for each vehicle from the main page. The features, including Battery Size, Towing Capacity, Acceleration, Top Speed, Efficiency, Fast Charge Speed, Real Range, Drive Type, and pricing details for the UK, Netherlands, and Germany, were found to be encapsulated within span tags featuring nomenclature corresponding to the respective feature names. The extracted data was stored in a dataframe for further analysis.

In the second phase of data scraping, the objective was to access individual car pages

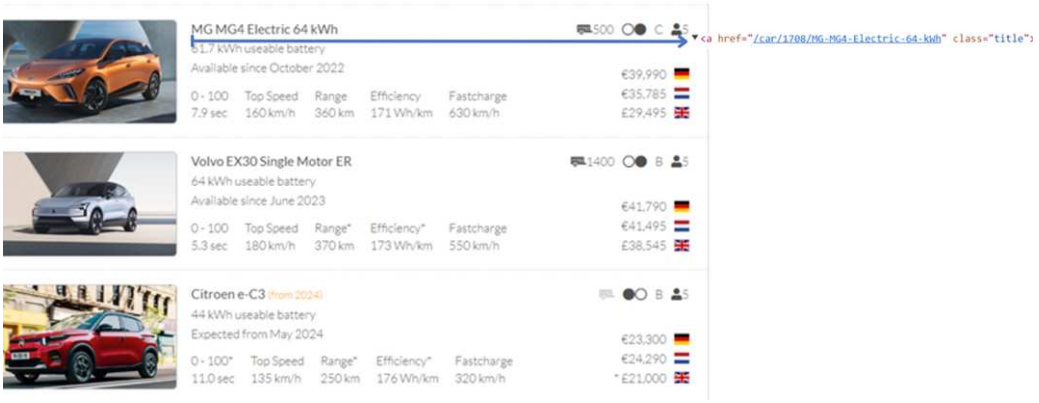


Figure 3: Illustration of Anchor tag

to extract additional information not available on the main pages. To achieve this, it was vitally important to aggregate all 367-car links into a new variable. In the context of HTML, anchor tags define hyperlinks, creating a connection from one page to another. The content within the anchor tag specifies the destination link, and the presence of the href attribute facilitates link activation upon clicking. To gather all 367 car links, the anchor tags were parsed once again using the .find(_)all method from BeautifulSoup, and the resulting links were stored in a variable named "car(_)links." Since the anchor tags contained partial link information, a subsequent step involved appending 'https://ev-database.org' to all the partial car links through the utilization of looping. Figure 3 illustrates the anchor tag.

5.2.1 Avoiding Website Blocks

To circumvent website restrictions, several proactive measures were incorporated within the python code. These measures included the utilization of headers and a user agent, the integration of a timeout attribute within the request.get function, the implementation of try and except blocks, and the incorporation of time.sleep from the time package within Python.

1. Headers - Headers are vital for web scraping, allowing access to restricted content by mimicking a legitimate browser. Our code used the "user-agent" header, which mimics a browser's identity, including its name, version, and platform. This simulation helps the scraping tool appear as a genuine user, reducing the risk of being blocked during operations.

2. Timeout- The timeout attribute in requests.get sets a limit for server response time. It's crucial for optimizing program execution by preventing stalling if the server delays. Our code used a default timeout of 10 seconds, preventing prolonged waits for a response.

3. Try and Except - The try and except blocks typically serve as fundamental constructs for managing exceptions, commonly referred to as error handling mechanisms. The try block tests for errors, while the except block manages them. We utilized requests.exceptions.RequestException to intercept general exceptions that might occur during code execution involving the request module.

4. Sleep - The integration of time.sleep introduces a deliberate delay in the Python program's execution. Specifically, our implementation included a 40-second pause to proactively manage potential blocks during web scraping. We experimented with varying delay times—10, 20, 30, 40 seconds, and 1 minute—to optimize without risking IP blocks. Although a 1-minute delay ensured safety from blocks, it significantly slowed the scraping process. To balance urgency and reliability, we settled on a 40-second interval, breaking down the code into multiple sets to generate several dataframes swiftly.

The ultimate set of six pivotal features, integral to our analysis, were encapsulated and seamlessly integrated to operate in unison with the implemented safety protocols. The features included safety rating, cargo volume, turning radius, home charge speed, horsepower, and vehicle segment. The HTML code revealed the structuring of these elements encased within table tags. The visual representation of the HTML code with respect to horsepower of the Tesla Model Y is illustrated in Figure 4.

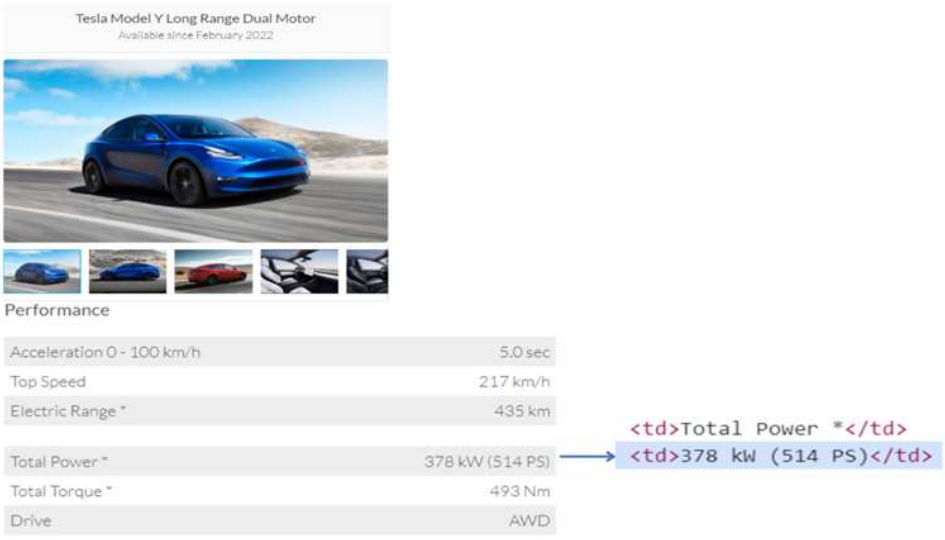


Figure 4: Illustrating the HTML table tag

The extracted features of individual cars were systematically stored into dataframes. The code sequence was executed iteratively, generating one dataframe during each run, with each dataframe containing information related to approximately 30 to 40 cars. This approach was adopted due to observations that scraping data beyond 40 cars triggered

access restrictions on the website. Hence, the variable 'links(_)limit' facilitated the specification of the desired number of vehicles for scraping. Finally, the resulting 11 dataframes were exported to CSV format and consolidated into a unified Excel file through a merging process.

5.3 Data Cleaning

During data cleaning phase, meticulous checks were conducted to eliminate unwanted characters like spaces, commas, and brackets from columns like acceleration, top speed, efficiency, fast charge speed, real range, cargo volume, turning radius, home charge speed, and horsepower. 'NONE' values were replaced with 'NA,' and all numerical values were standardized to a two-decimal format. Vehicle prices in Euros were converted to British Pounds using a 'calculate(_)price' function, with 1 Pound Sterling equating to 1.15 Euros. Columns with prices in Euros were then removed for consistency.

The subsequent phase involved data type reconfiguration to optimize the dataset for analysis. Specific columns, such as Acceleration, Turning Radius, Cargo Volume, and Home Charge Speed, were transformed from an object data type to float. Concurrently, columns encompassing features like Top Speed, Efficiency, Fast Charge Speed, Real Range, and Horsepower underwent conversion to an integer data type.

Examining the presence of missing values - 'NA':

Brand	0
Model	0
Battery Size	0
Towing Capacity	120
Number of Seats	0
Acceleration	0
Top Speed	0
Efficiency	0
Fast Charge Speed	0
Real Range	0
Drive Type	0
Safety Rating	200
Cargo Volume	11
Turning Radius	72
Home Charge Speed	3
Vehicle Segment	0
HP	0
Uk Price	0

Figure 5: Total Null Values

Properly handling missing data is critical during data preprocessing to avoid biases that could skew statistical analyses and subsequently impact business decisions. Understanding the types of missing data is also crucial:

1. MCAR (Missing Completely at Random): Missingness occurs with equal probability across all variables or observations.

2. MAR (Missing at Random): The missingness probability relates to observed variables' data points within the dataset.

3. MNAR (Missing Not at Random): The missingness probability varies for reasons that are unknown to us.

By leveraging the isna() function within the pandas library, we quantified the absence of values in our dataset. Comprehensive details regarding these missing values can be found in Figure 5

5.3.1 Steps to handle missing data

Excluding/Deleting rows with missing values was not a viable choice in our context owing to the dataset's limited size, that was made up of only 367 data points. The removal of any such rows would have exerted a significant impact on the dataset's volume, potentially resulting in a substantial reduction in available information for analysis. Such an action could have notably hampered the predictive accuracy of models trained on the dataset.

Imputing missing data in Home charge speed, cargo volume, and safety rating followed Missing at Random (MAR) assumptions. Vehicles without an NCAP safety rating were assigned a safety value of 0, potentially suggesting a link between missingness and safety testing status. Our study solely considered NCAP safety evaluations, possibly overlooking alternative safety assessments for these vehicles.

In our ongoing efforts to handle missing data, a subset of entries pertaining to the cargo volume and home charge speed variables were noticeably absent, amounting to 11 and 3 instances, respectively.

To address these missing data instances, we opted for the mean imputation strategy, replacing these missing values with the mean computed across the entire column. This method ensured a systematic approach to mitigating missingness while upholding data integrity.

5.3.2 Iterative Imputations

Priming the dataset for Iterative Imputations: Considering the abundance of missing values in towing capacity and turning radius, traditional imputation methods such as zero-filling or mean-value substitution were deemed unsuitable. An alternative approach was imperative to address these instances of substantial missingness.

Iterative imputation— Iterative imputation, using scikit-learn's Iterative Imputer, tackles extensive missing data by intelligently estimating values. It operates by treating each missing data point as an output variable while using other features as inputs.

Through a regression model trained on known values, it predicts and fills missing values iteratively for each feature column.

Prior to Iterative imputation categorical variables like safety ratings, drive types (like all-wheel, front-wheel, and rear-wheel drive), and vehicle segments (Cabriolet, Coupe, Hatchback, etc.) were converted into numerical formats. Safety ratings were set as binary values: 0 for no NCAP rating and 5 for safety rated vehicles. The Drive type and vehicle segment columns, underwent one-hot encoding, creating individual binary columns for each category using pandas' get(_)dummies function.

Following the data's conversion into the appropriate format, an IterativeImputer object was instantiated to manage missing values effectively. This object incorporated essential parameters, namely estimator, max(_)iter, and random(_)state. The estimator parameter was configured to utilize a Linear Regression model, serving as the underlying algorithm to predict and impute missing values. Meanwhile, the max(_)iter parameter was set to 10, defining the maximum number of iterative rounds performed by the imputer before delivering the imputations derived from the final iteration. Additionally, the random(_)state parameter was fixed at 0 to ensure the reproducibility of results.

A comprehensive check was conducted to identify any duplicated rows within the dataset, revealing the absence of any redundant or replicated datapoints. This verification process ensured the dataset's integrity, confirming that each data entry was unique and distinct.

5.4 Data Visualization

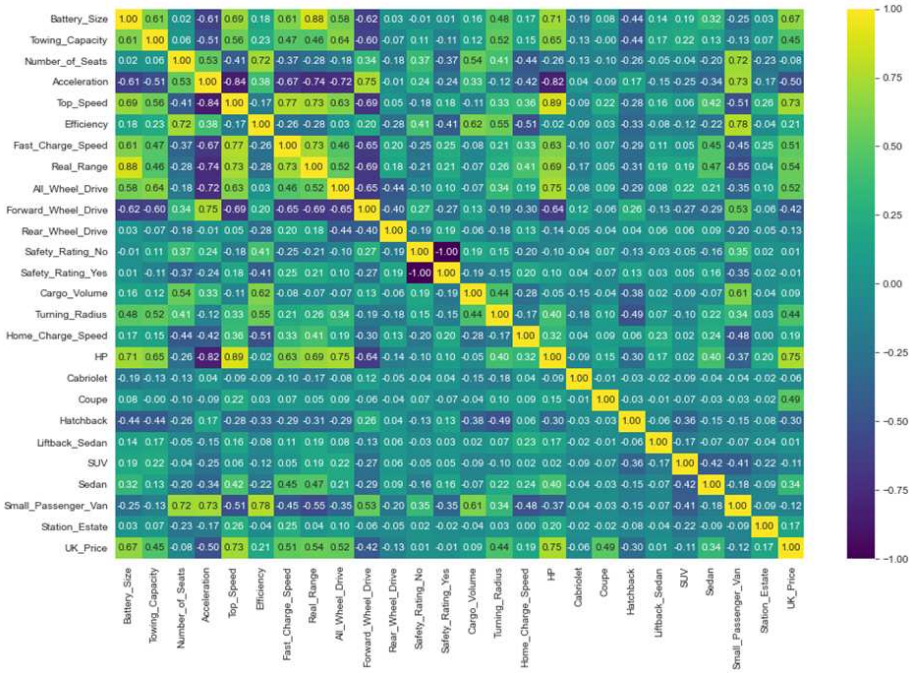


Figure 6: The Correlation Plot

The initial phase of the data visualization process invariably involves assessing correlations among variables. Correlation, a fundamental statistical tool, quantifies the strength of the linear relationship between two variables through a numerical metric known as the correlation coefficient.

In this context, identifying correlations was vital for predictive modelling. Analysis of the visual data in Figure 6 revealed significant correlation coefficients of 0.67, 0.73, and 0.75 for battery size, top speed, and horsepower, respectively, depicting robust positive associations with the UK(_)price. These coefficients imply that as these attributes—battery size, top speed, and horsepower—rise, so does the price.

Subsequent visualization through scatter plots reinforced this strong assertion with the price of BEVs. Figures listed under 7.0 provide compelling evidence of the positive relationships existing among these key variables and their influence on the pricing dynamics.

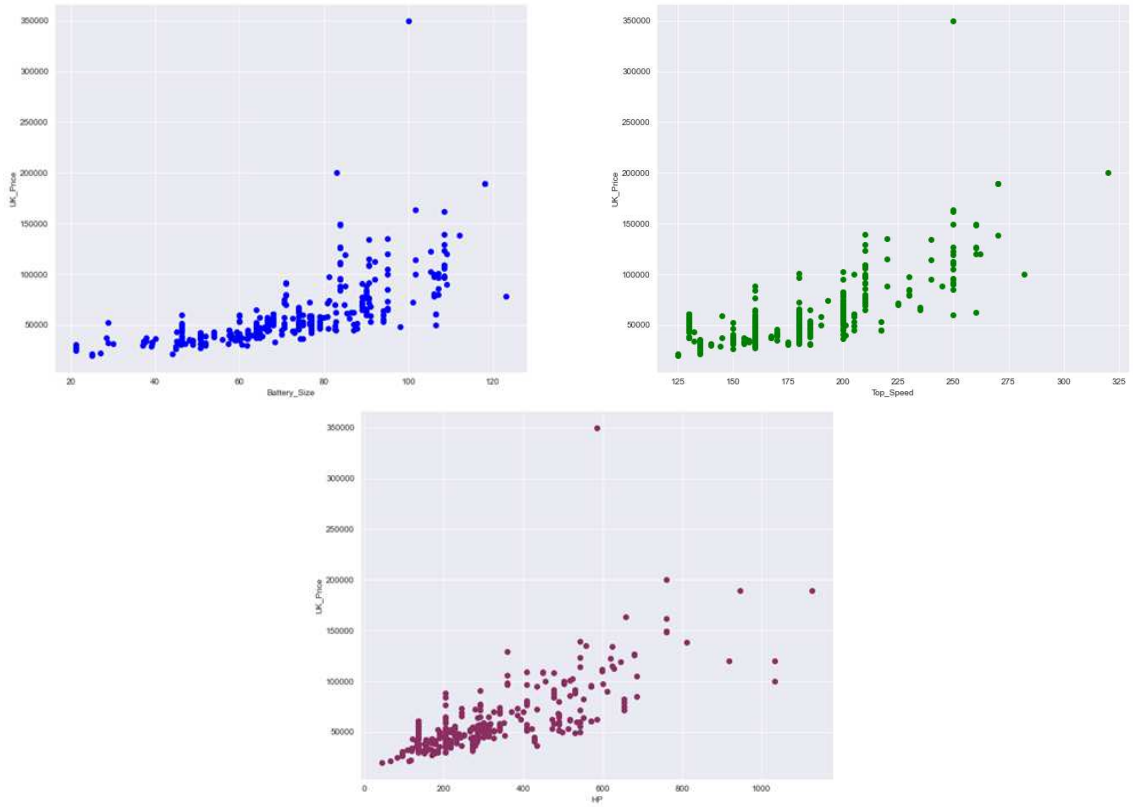


Figure 7: Scatter plots : (7a)Battery size vs UK price (7b)Top speed vs UK price (7c)HP vs UK price

Figure 7a illustrates the unmistakable link between battery size and vehicle price. A clear trend emerges, indicating that larger battery sizes align with increased vehicle prices. Remarkably, the data is densely packed within the 45kWh to 80kWh range, predominantly clustered around the £50,000 mark. This clustering pattern hints at a common pricing zone for BEVs falling within specific battery size brackets. Some outliers are visible in

the image (7a)

In figure 7b, the visualization reveals a compelling association between top speed and vehicle price. The plot unravels that vehicles boasting higher top speeds typically demand higher prices, aligning with the prevalent understanding that premium or supercar categories often carry heftier price tags due to their exceptional performance attributes. Moreover, the plot draws attention to select outliers that need deeper examination due to their deviation from the established trend.

In figure 7c, a distinct positive correlation between horsepower and price is evident. The visualization highlights a predominant clustering of vehicles within the dataset, ranging between 100 to 380 horsepower, often associated with lower price ranges. This trend suggests a link between lower horsepower measurements and comparatively reduced-price points, hinting at a potential relationship between horsepower and pricing dynamics. Additionally, it hints at the interdependence of horsepower and top speed, indicating that vehicles having higher horsepower typically also possess superior speed characteristics, hence correlating with higher prices. Some outliers are visible in the image (7c)

A thorough analysis of drive type configurations in BEVs was conducted using violin plots (8) to explore their implications. These plots, combining box plot features, visually displayed data distribution and concentration within the interquartile ranges for RWD, FWD, and AWD drivetrains.

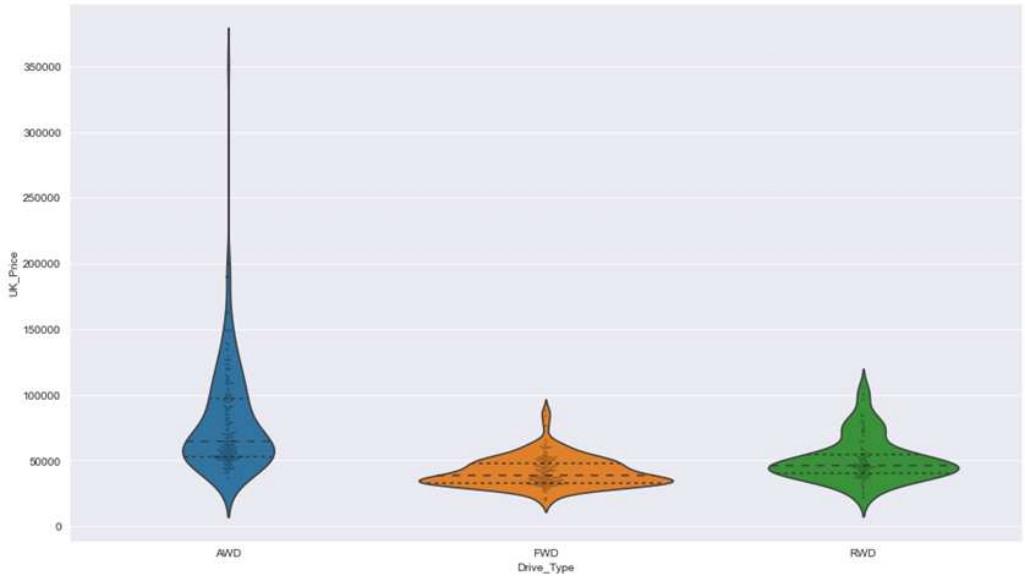


Figure 8: Illustrating drive type configurations

Within the AWD category, the bulk of data is concentrated within the 25(%) to 50(%) quartile ranges, prominently around the £50,000 mark. However, the elongated tail in the plot signifies a wide dispersion of data points, possibly indicating outliers or substantial variation within this category.

In contrast, the distribution of FWD showcases narrower tails with more concentrated

data, predominantly spanning between the 25(%) to 75(%) quartile ranges, creating evident peaks around £40,000 and £50,000. This pattern implies a generally lower average price range in comparison to AWD and RWD setups.

RWD exhibits reduced prominence in peaks compared to FWD, yet it doesn't display more dispersion than AWD. The data appears densely concentrated, notably around the £50,000 mark, suggesting a more uniform pricing trend within this specific drivetrain category. These visuals provide valuable insights into the pricing patterns of BEVs across diverse drive type configurations.

Continuing with exploratory analysis, the study delved deep into distinct car segment types—SUVs, hatchbacks, and sedans—conforming to established literature on battery size comparisons. Figure 9, a color-coded scatter plot, highlights varying densities among SUVs, hatchbacks, and sedans in the electric vehicle market. SUVs and sedans are prevalent, while hatchbacks have lower densities. The pricing trends show hatchbacks as the most affordable due to smaller size and lower battery capacities. Conversely, sedans tend to have larger batteries and potentially higher prices. This inclination toward larger batteries and potentially higher prices in sedans could be indicative of an emerging market trend. Manufacturers seem to be emphasizing battery-powered sedan models more prominently, possibly targeting a specific consumer base with a preference for this vehicle type.

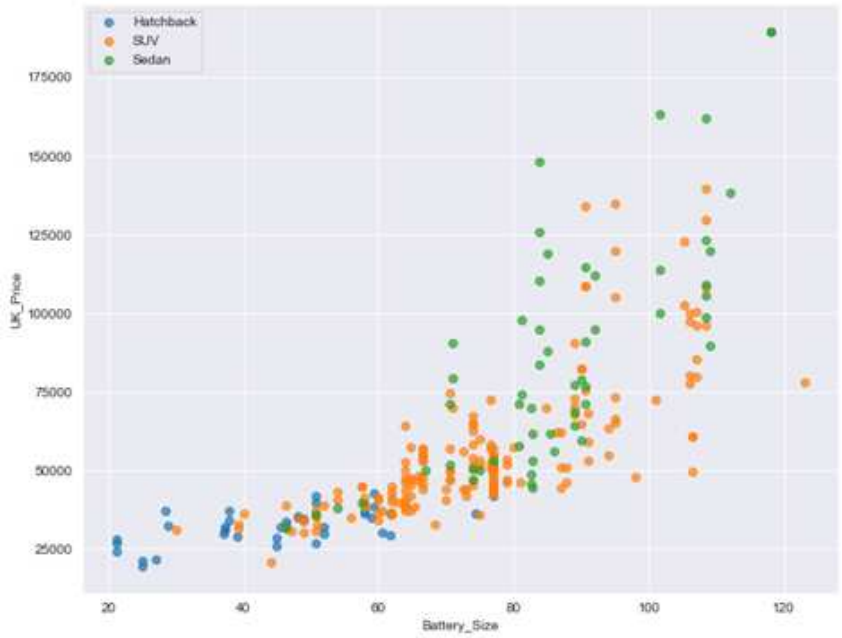


Figure 9: Segment types (Hatchback, Sedan, SUV)- Battery Size vs UK Price

Finally, understanding the distribution of EV prices was crucial for determining the necessity of data normalization. The histogram (Figure 10) provided insights into the price distribution of BEVs.

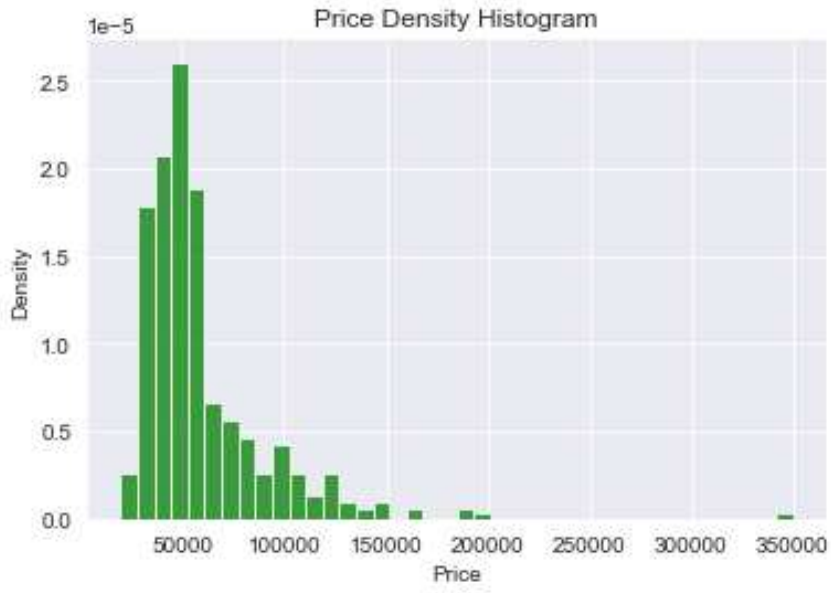


Figure 10: EV Price Distribution

It highlighted outliers, notably vehicles priced at £350,000, prompting their exclusion to avoid biases in the coefficients. Furthermore, the histogram revealed a distinct positive skewness, showcasing a longer right tail compared to the left. This asymmetry indicated that higher-priced vehicles exerted a more pronounced influence on the dataset's mean, pulling it towards the right. As a result, a plan to normalize data was formulated to address skewness. This strategic approach aimed to ensure a more balanced dataset, vital for reliable and unbiased analysis in subsequent stages.

5.4.1 Outlier removal using z-score

Preventing outliers from distorting analyses and machine learning models is crucial. Three common approaches for outlier management include: visually identifying and removing extreme data points, using interquartile ranges (IQR) to filter out data beyond Q1 and Q3, and employing z-scores to handle outliers based on their deviation from the mean in terms of standard deviations.

Z = (x- mu)/ sigma

Z = standard score

x = observed value

mu = mean of the sample

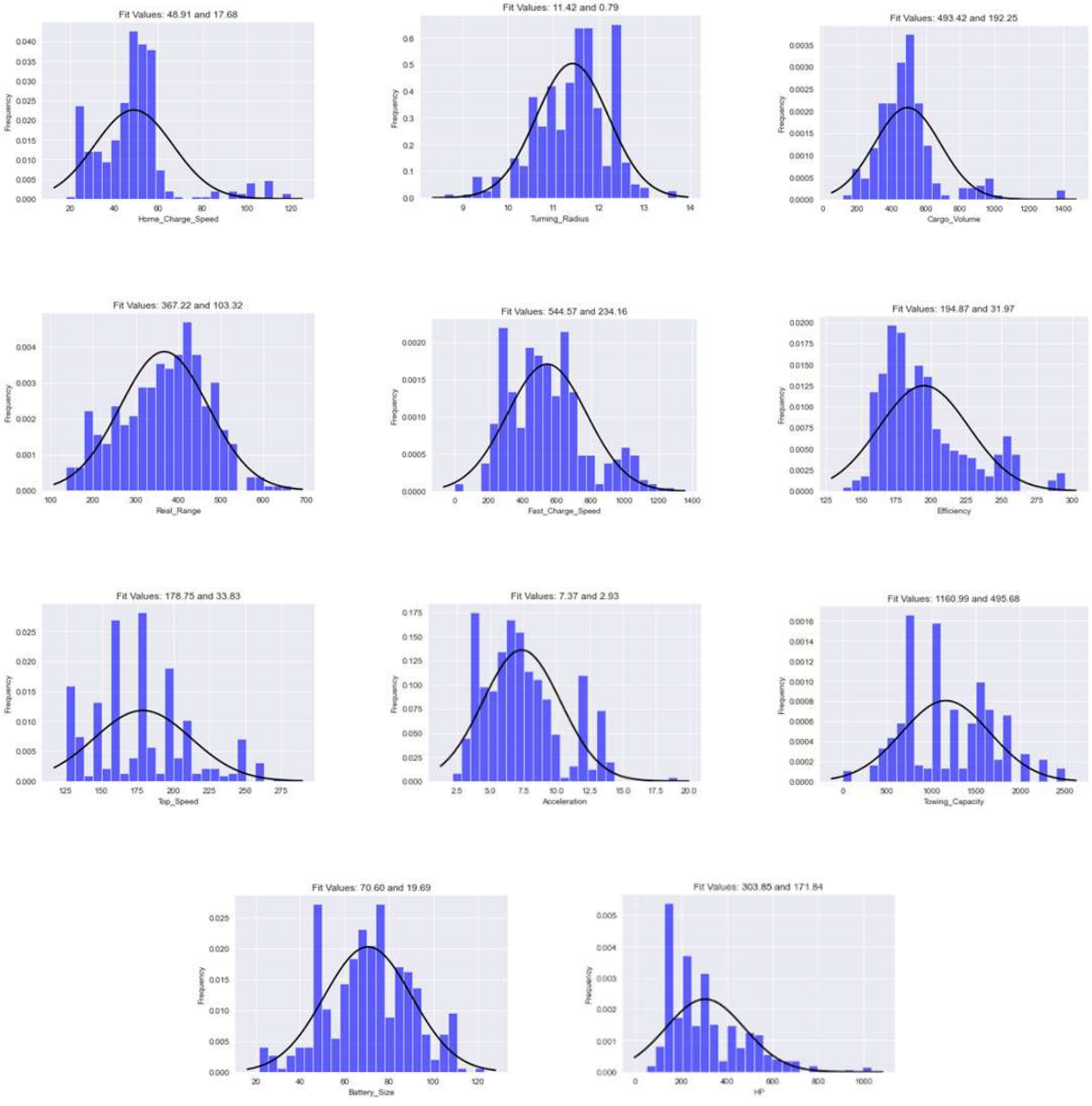
sigma = standard deviation of the sample

In our study, we opted against the first approach and instead focused on employing the second and third methods for outlier handling. When utilizing the interquartile range (IQR) method, the removal of outliers led to a substantial drop of approximately 35 data points, reducing our dataset from 367 to 332. However, considering the limitations

imposed by data scarcity, this magnitude of reduction was deemed unsuitable for our study, prompting the exploration of alternative strategies.

Consequently, we turned to the z-score method, facilitated by the Scipy stats package, for outlier identification and removal. By calculating z-scores specifically for the UK price column, data points exceeding a z-score of 3 or falling below -3 were systematically eliminated. This process resulted in the removal of 6 data points, ultimately leaving us with a refined dataset of 361 data points for subsequent analyses. This final dataset, free of outliers identified through z-score analysis, served as the foundation for all predictive model runs, ensuring a more robust and reliable dataset.

Following outlier removal, an evaluation was conducted to assess the normality of the dataset, containing attributes such as battery size, towing capacity, acceleration, top speed, efficiency, fast charge speed, real range, cargo volume, turning radius, home charge speed, and horsepower.



This analysis included an examination of the skewness measures, the means, and the standard deviations for these attributes. This comprehensive scrutiny aimed to deduce the distribution characteristics and statistical properties of each attribute post-outlier handling.

Findings:

Negatively skewed - battery size, real range, turning radius.

Positively skewed - towing capacity, acceleration, top speed, efficiency, fast charge speed, cargo volume, home charge speed, horsepower.

5.5 Data Validation Approach

Holdout Evaluation, a common validation technique, involves dividing a dataset into a training set for model fitting and a separate test set for evaluating the model's performance, typically measured by holdout accuracy. This metric serves as an approximation of the model's generalization accuracy.

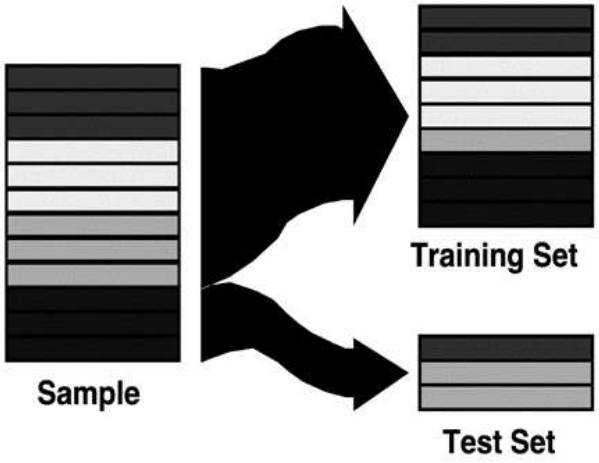


Figure 11: Dataset Split

Nevertheless, this method comes with its set of limitations. The validation estimate of the test error can show significant variability based on which observations are part of the training set versus the validation set. In this approach, only the observations in the training set, not in the validation set, are used to build the model. Consequently, this implies that the validation set error might tend to overstate the test error for the model trained on the complete dataset.

Utilizing the train(_)test(_)split class from the sklearn.model(_)selection library, our dataset always underwent a 75-25 split. This process allocated 75(%) of the data to the training set, reserving the remaining 25(%) for the test set. To ensure reproducibility, the random(_)state parameter was set to 0, ensuring identical datasets across multiple code runs. This careful setup facilitated a more robust and consistent model evaluation by

maintaining a standardized division of data for training and testing purposes.

5.5.1 K- Fold cross validation

The K-fold cross-validation approach randomly divides the training set into k folds, iteratively training and testing k times. Each fold acts as a unique test set while the rest form the training set. Calculating average and variance of accuracies across folds reduces overfitting risks and provides a dependable evaluation of the model's performance. This technique, especially with tools like Python's cross(_)val(_)score in sklearn, offers robust estimation by using diverse test sets and repeated iterations to gauge the model's generalization capability.

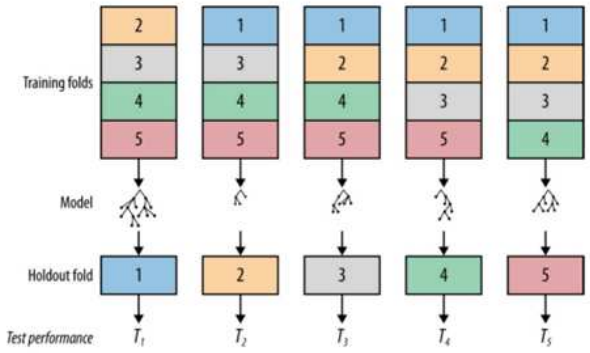


Figure 12: K-Fold Cross Validation

5.5.2 Variable normalization

Normalization serves to standardize features onto a comparable scale, a crucial step benefiting machine learning models by enhancing their performance and fostering training stability. Various methods exist for data normalization, involving transformations like log scaling, z-score normalization, and scaling to a specific range. Our approach, involved scaling data to a specific range, specifically mapping all numerical values within the range of 0 to 1. This transformation facilitated the alignment of varying feature scales, ensuring uniformity and aiding in effective model training.

$$x' = (x - x_{min})/(x_{max} - x_{min})$$

Python enables normalization using the MinMaxScaler class from the sklearn.preprocessing library. This transformation was applied universally across all variables, excluding dummy variables, which inherently reside within a range of 0 to 1. Scaling was performed on X(_)train, X(_)test, and y(_)train using separate scaler objects for X and y. Post-normalization, the inverse(_)transform method was crucial for reverting scaled values back

to their original scale, aiding in interpreting predictions and accuracy metrics with the test set.

5.6 Creating two datasets

In the process of refining potential predictors for our dependent variable, it's crucial to select relevant variables to prevent the "Garbage in, garbage out" phenomenon, ensuring the model's reliability and interpretability. This selection process aims to retain only the most influential variables that substantively contribute to predicting outcomes. To achieve this, several methods, including All-in, Backward Elimination, Forward Selection, Bidirectional Elimination, and Score Comparison, can be employed.

In our methodology, we adopt two distinct approaches: Firstly, the "All-in" method involves using all available data without discarding any variables. This approach enables leveraging the entirety of the dataset for modelling purposes. This formed our **Dataset 1**.

Secondly, implementing the Backward Elimination technique, known as stepwise regression, systematically removes the less relevant variables from the multiple linear regression model.

Steps for Backward Elimination:

Step 1: Select a significance level (SL) to stay with the model. We made use of SL = 0.05

Step 2: Fit the full model with all the possible predictors

Step 3: Consider the predictor with the highest P-Value. If P_{SL} , go to step 4

Step 4: Remove the predictor

Step 5: Fit the model without this variable. Recreate the model. The coefficient and constants will be different.

Following step 5 in the variable selection process, the subsequent step involves revisiting step 3, continuing the iterative cycle until all variables meet the specified significance level (SL) criterion. This iterative process progresses through steps 3 and 5, pruning variables until the remaining ones attain p-values below the designated significance level.

Through the Backward Elimination technique, several variables, namely sedan, efficiency, station estate, cargo volume, cabriolet, number of seats, turning radius, coupe, and towing capacity, were systematically removed from consideration. These variables, found to have limited or negligible impact on the statistical significance within the multiple linear regression model, were excluded from the final selection process.

Post the detailed Backward Elimination procedure, our final dataset consisted solely of variables exhibiting sustained statistical significance below the set threshold of 0.05. These variables were Battery Size, Acceleration, Top Speed, FastCharge Speed, Real Range, All Wheel Drive, Forward Wheel Drive, Rear Wheel Drive, Safety Rating No,

Safety Rating Yes, Home Charge Speed, HP, Hatchback, Liftback Sedan, SUV, and Small Passenger Van. This carefully curated set of variables constituted our Ordinary Least Squares (OLS) dataset, poised for in-depth analysis and modelling purposes in further stages of our investigation. This formed our **Dataset 2**.

6 Findings

6.1 General Model Overview

Model Assessments

Sr No.	Model	Description
1	Regressor	Linear Regression with dataset 1
2	Regressor1	Linear Regression with dataset 2
3	Regressor2	Support Vector Regression with dataset 1
4	Regressor3	Support Vector Regression with dataset 2
5	Regressor4	Decision tree with dataset 1
6	Regressor5	Decision tree with dataset 2
7	Regressor6	Random Forest regression with dataset 1
8	Regressor7	Random Forest regression with dataset 2
9	Regressor8	XGBoost with dataset 1

6.1.1 Regressor – Linear Regression with dataset 1

The LinearRegression class from the sklearn library was imported to work as the first model. Notably, this model was constructed without normalization, given that Linear Regression inherently maintains scale invariance concerning predictors. In this context, the coefficients associated with each predictor signify the change in the dependent variable per unit alteration in the predictor. As a result, rescaling predictors does not impact the interpretation of these coefficients.

During performance evaluation, the model exhibited an MAE of £8005, an RMSE of £11022, and an R-squared measure of 77.41(%), indicating the variance explained by the model. The MAE of £8005 in the context of predicting the price of battery-powered electric vehicles signifies that, on average, the model's forecasts deviate by approximately £8005 from the actual observed prices. Subsequently, employing K-Fold Cross-Validation with a parameter setting of cv=10, the model achieved a mean R-squared measure of 75.17(%), accompanied by a standard deviation of 8.65(%) on the training set. This outcome implies that the model's R-squared value may fluctuate within the range of 75.17(%) plus or minus 8.65(%).

6.1.2 Regressor1 – Linear Regression with dataset 2

The second iteration of implementing the LinearRegression class from the sklearn library involved a different approach, incorporating data normalization for columns excluding dummy variables. Employing the MinMaxScaler class from the sklearn package facilitated the normalization process, transforming non-dummy columns to a consistent scale. After restoring the scaled values to their original form, the model was used to predict on the test set, yielding performance metrics.

The model demonstrated a MAE of £7806, a RMSE of £10929, and an R-squared measure of 77.79(%). These metrics indicate the model's predictive capabilities, with the MAE suggesting an average deviation of approximately £7806 between predicted and observed prices for BEV's. Employing K-Fold Cross-Validation with cv=10, the model exhibited a mean R-squared measure of 75.97(%) alongside a standard deviation of 8.43(%). This outcome suggests that the model's R-squared accuracy may fluctuate within the range of 75.97(%) plus or minus 8.43(%).

Utilizing a Kernel Distribution Estimation Plot (KDE) offers a visual means to assess our model's performance by comparing predicted values against actual values.

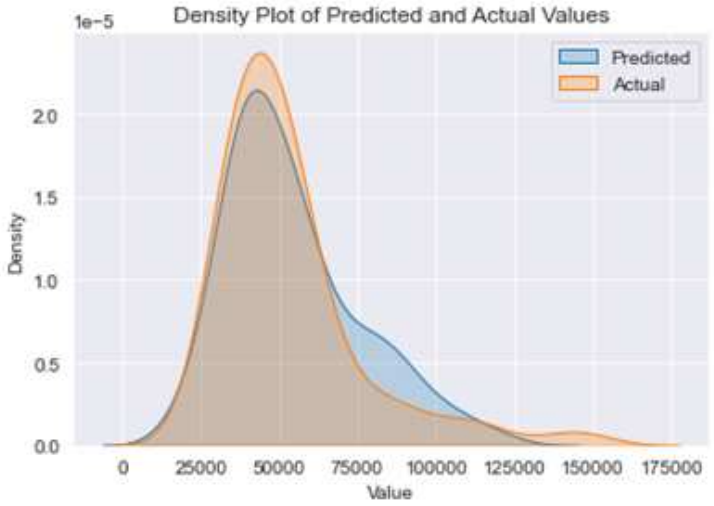


Figure 13: Linear Regression KDE Plot- Predicted values vs Actual values

Based on the visualization depicted in Figure 13, the model's performance reveals certain shortcomings. It appears that the model consistently underestimates prices within the £50,000 range while tending to overestimate values in the price range spanning from £75,000 to £100,000.

6.1.3 Regressor2 – Support Vector Regression with dataset 1

The implementation of the Support Vector Regression (SVR) from the sklearn.svm package necessitated feature scaling to gather meaningful results. Utilizing the MinMaxScaler class, the dataset after splits, that includes X(_)train, X(_)Test, and y(_)train, underwent normalization. The model had the kernel parameter set to "rbf," signifying the use of a radial basis function kernel.

Upon normalization and subsequent prediction phases, the model exhibited performance metrics with an MAE of £8523, an RMSE of £11290, and an R-squared measure of 76.30(%). The MAE value suggests an average deviation of £8523 between predicted and actual prices, highlighting the model's predictive accuracy.

Furthermore, employing K-fold Cross-Validation with cv=10, the SVR model displayed an R-squared measure of 75.79(%) alongside a standard deviation of 5.80(%). This outcome implies that the model's R-squared values might vary within the range of ±5.80(%) from the mean value of 75.79(%), signifying potential fluctuations in the model's predictive accuracy across different data subsets.

Hyperparameter tuning - In the context of fine-tuning the SVM model, two essential parameters underwent optimization. Initially, the parameter "C," functioning as a regularization parameter, was explored within a range spanning from 1 to 100 in steps of 2. This parameter plays a critical role in determining the strength of regularization, operating inversely to C's value. A higher C leads to more careful fitting to the training data, potentially resulting in overfitting, while a lower C value allows for a more generalizable model by tolerating classification errors.

Simultaneously, the "gamma" parameter, pivotal in defining the influence range for calculating the decision boundary, was explored across a range from 0.001 to 1 in steps of 0.1. A higher gamma emphasizes nearby points in defining the decision boundary, while a lower gamma extends the influence to distant points.

The GridSearchCV class, configured with a cross-validation parameter set to cv=6 and utilizing the R-squared metric for scoring, was used with these specified tuning parameters. This exploration resulted in identifying the most optimal parameters, showcasing a peak R-squared measure of 78.90(%). The best parameters deduced from this search were C:3 and gamma:0.101, signifying the ideal parameter combination that maximizes the model's predictive performance.

Employing the optimized grid parameters derived from the hyperparameter tuning phase, the model was applied to the test dataset, resulting in enhanced predictive performance metrics. Specifically, the model demonstrated an improved MAE of £8291, a RMSE of £11003, and an R-squared value of 77.49(%).

6.1.4 Regressor3 - Support Vector Regression with dataset 2

Upon importing the SVR class from the sklearn.svm package, the dataset underwent normalization using the MinMaxScaler class, excluding dummy variables. Employing the radial basis function (rbf) kernel, the SVR model was used for predictions on the test set after inverse transformation.

However, the evaluation revealed less favourable performance metrics than the optimal results achieved with the Regression2 model. Specifically, this model exhibited a MAE of £8642, a RMSE of £11510, and an R-squared measure of 75.36(%). The MAE measure suggests that, on average, the model's predictions deviate by approximately £8642 from the actual prices.

Further assessment through K-fold Cross-Validation with CV=10 indicated an R-

squared measure of 73.51(%) alongside a standard deviation of 6.80(%). The lower scores obtained from this validation method discouraged the implementation of GridSearchCV due to the model's relatively poorer performance compared to previous iterations.

6.1.5 Regressor4 – Decision tree with dataset 1

In this instance, variables excluding dummy variables underwent normalization utilizing two separate scalers for X and y sets post data splits. Employing the DecisionTreeRegressor class from the sklearn.tree package, specific parameters were fine-tuned to enhance model performance.

The "max(_)depth" parameter was utilized to control the depth of the decision tree, influencing the maximum levels or layers the tree could reach during construction. Setting this parameter to 7 limited the tree's growth to seven levels, preventing over-complexity. Experiments were conducted using different max(_)depth values, including 3,4, 5, 6, 8, 9, 10, 11, 12, 13, 14, and 15. Upon increasing the depth values beyond 15, the model exhibited a decline in performance. Additionally, the "ccp(_)alpha" parameter, guiding the pruning of decision nodes, was adjusted across various values, ranging from 0.1 to 0.0001 in small increments. Among the explored values, ccp(_)alpha = 0.0001 in combination with a max(_)depth of 7 yielded the most favourable results.

Post-normalization and subsequent transformations, the model showcased notable improvements in metric measures when compared to the linear regression and SVM models, achieving a MAE of £5500, a RMSE of £7933, and an R-squared measure of 88.29(%). K-fold cross-validation was employed to gauge the extent of overfitting in the model and to assess its deviation from the true results. Employing a cross-validation parameter (cv) set to 10, the model's performance was evaluated.

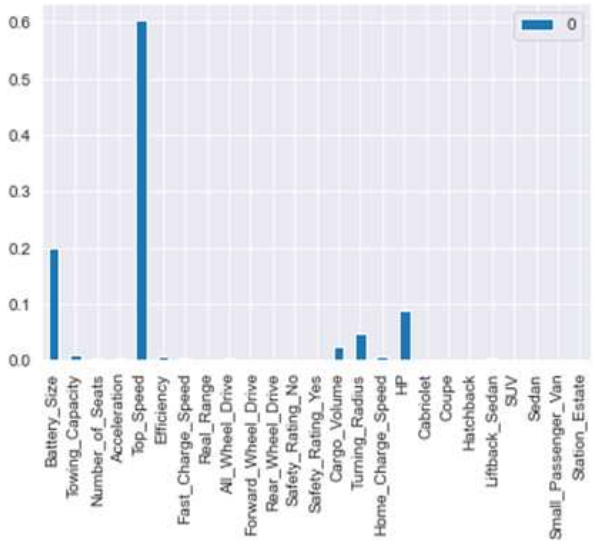


Figure 14: Feature importance – Decision Tree

The outcomes revealed an average R-squared measure of 75.49(%), coupled with a standard deviation of 10.18(%). These findings suggest potential disparities between our initial model results and the actual outcomes. The wide variance observed in the R-squared values across different folds implies a significant fluctuation in the model's predictive performance.

Utilizing a bar plot visualization, we determined the pivotal features that significantly influenced the model's decision-making process. Notably, Battery size, top speed, horsepower, and turning radius emerged as the prominent contributors to price, carrying substantial weight in the model's construction. Figure 14 effectively illustrates the feature importance according to the model.

6.1.6 Regressor5 – Decision tree with dataset 2

Following the same preprocessing steps as Regressor4, the dataset underwent normalization for non-dummy variables using the MinMaxScaler Class. Through manual tuning, the parameters max(_)depth and ccp(_)alpha were adjusted, revealing that when ccp(_)alpha was set to 0.00117 and max(_)depth to 3, the model demonstrated the most optimal accuracy measures.

However, the model's performance fell short compared to the Regressor4 model. The evaluation exhibited an MAE of £7354, indicating that, on average, the model's price predictions for BEVs deviated by approximately £7354 from the actual prices. The RMSE stood at £11276, and the R-squared value was 76.36(%).

Conducting K-fold Cross-Validation with 10 folds unveiled an R-squared value of 67.53(%), accompanied by a substantial standard deviation of 21.74(%). This lower R-squared value implies that decision trees tend to perform sub optimally when working with datasets limited in independent variables, preferring a more extensive array of features to make accurate predictions. Due to the notable disparity between predicted and actual values, no further grid cross-validation was pursued for this model.

6.1.7 Regressor6 – Random Forest regression with dataset 1

Post standard preprocessing steps, the dataset underwent normalization via the MinMaxScaler class. Next, the RandomForestRegressor class from the sklearn.ensemble package was utilized, with particular changes to default parameters, specifically targeting n(_)estimators and random(_)state.

The n(_)estimators parameter, signifying the number of trees within the forest, was manually fine-tuned across a range from 5 to 50 trees. Optimal model performance was observed with n(_)estimator=35, showcasing the best accuracy metrics during evaluation. The random(_)state parameter was set to 0.

During model performance evaluation with test predictions, the MAE indicated a potential average deviation of ±£5045 in the model's price predictions for BEVs. The RMSE stood at £7520, and the R-squared metric demonstrated a high measure of 89.48(%). Conducting K-fold Cross-Validation served to validate the model's generalizability and its susceptibility to overfitting. The R-squared measure obtained from the K-fold analysis revealed a value of 85.41(%), with a relatively low standard deviation of 5.98(%). This suggests that the model performs consistently well on out-of-sample data, indicating robust performance and a reduced risk of overfitting.

The KDE plot (figure 15), showcasing the comparison between predicted and actual values from the test set, revealed an almost accurate performance by the model. However, an evident overestimation was noticed within the price range of £75,000 to £100,000. This discrepancy could potentially be attributed to the scarcity of data points within this price range, limiting the model's ability to effectively learn and generalize from a smaller pool of available data.

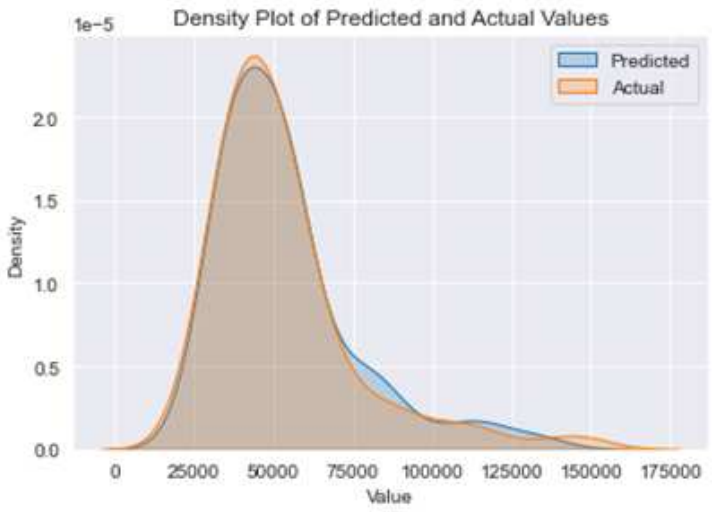


Figure 15: Random Forest KDE Plot – Predicted values vs Actual values

Hyperparameter Tuning - The hyperparameter tuning process utilized the GridSearchCV class from the sklearn package to optimize the parameters for the Random Forest model. Specifically, two key parameters, max(_)features, representing the number of features considered for each split, and n(_)estimators, denoting the number of trees in the forest, were tuned. Max(_)features were explored within a range from 1 to 15, with increments of 1. This exploration aimed to evaluate the impact of various subsets of features for optimal splitting. To assess the performance differences, trials were conducted with smaller subsets of features, such as 8, 10, and 12. However, these trials did not yield satisfactory results. In exploring n(_)estimators, the optimal range tested was from 10 to 280 with increments of 4. Further experimentation extended to ranges from 10 to 450 in increments of 5 and 10 to 350 in increments of 5, aimed at thoroughly assessing the influ-

ence of different tree quantities on model performance. Cross-validation was implemented with a 5-fold split, utilizing the R-squared scoring parameter to assess model performance during the hyperparameter tuning phase.

The best-performing configuration achieved an R-squared measure of 87.66(%), with optimized parameters set to max(_)features = 12 and n(_)estimators = 114. Subsequently, these tuned parameters were applied to make predictions on the test set, resulting in accuracy metrics of MAE = £5028, RMSE = £7830, and an R-squared of 88.60%).

While only a slight improvement in MAE was observed compared to the performance without hyperparameter tuning, the optimized model demonstrates its ability to generalize to unseen datasets, providing a more accurate reflection of its performance on new data.

The density plot (Figure 16) visually showcases the best combinations of features and n(_)estimators derived from the grid search. Lighter shades in yellow indicate where the model achieved its highest R-squared measures. This graphical representation offers a clear view of the most effective parameter combinations for optimal predictive accuracy.

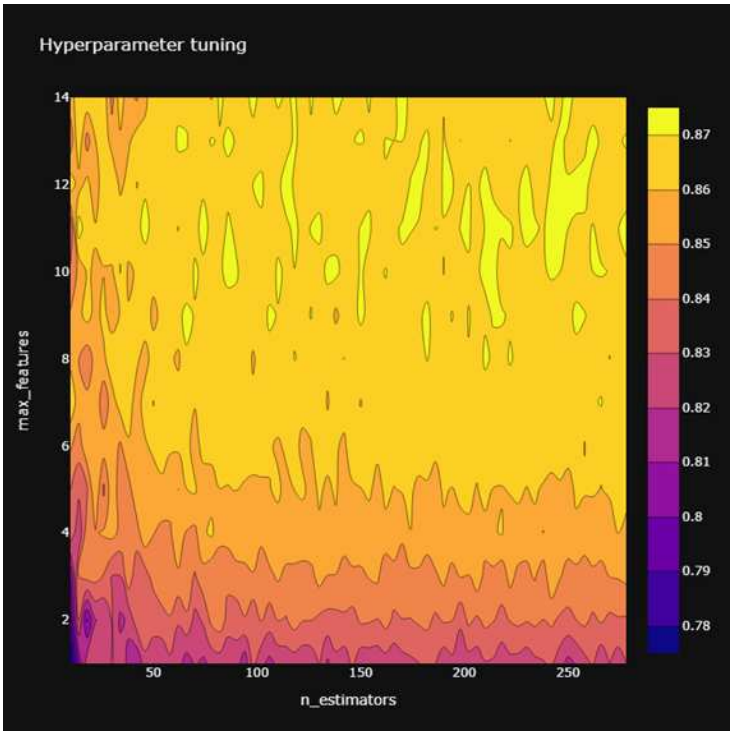


Figure 16: Density Plot – n estimators vs max features

6.1.8 Regressor7 – Random Forest regression with dataset 2

The feature set underwent standard preprocessing, including dataset splitting and numerical variable normalization across X(_)train, X(_)test, and y(_)train. Using RandomForestRegressor from sklearn.ensemble with n(_)estimators set to 26 and random(_)state

as 0, the model was built. Evaluations across varying n(_)estimators values depicted the highest accuracy at 26, offering the most optimal metric measures among tested values.

The model's performance revealed an MAE of £5146, RMSE of £8004, and an R-squared value of 88.09(%). These metrics suggest that, on average, predictions might deviate by approximately £5146. Regrettably, these results did not surpass the performance of the regressor 6 model without hyperparameter tuning. As a result, Grid SearchCV was not run on this model.

To further evaluate the model's robustness and validate our assumptions, K-Fold cross-validation with a 10-fold configuration was conducted. The computed accuracies yielded an R-squared average of 83.59(%) with a standard deviation of 6.81(%). The model's predictive capability exhibited variability, indicating an R-squared range between 76.78(%) and 90.4(%). This large variance highlights the potential fluctuation in the model's performance across different datasets.

6.1.9 Regressor8 – XGBoost with dataset 1

Following the manual installation of the XGBoost package due to the unavailability of the XGBRegressor class within the sklearn package in Python, the dataset underwent standard preprocessing steps. This involved splitting and normalizing independent variables (X(_)train and X(_)test) along with the dependent variable (y(_)train) using the MinMaxScaler class. Subsequently, the XGBRegressor class was imported from the xgboost package, where only the random(_)state parameter, was set to 0. Predictions were generated post inverse transformation from scaled to original values.

Notably, this model demonstrated superior performance compared to previously employed models. It achieved high accuracies, with an MAE of £5105, RMSE of £7200, and an impressive R-squared value of 90.36(%). Furthermore, to measure its reliability and consistency, K-fold cross-validation with a 10-fold configuration was conducted. The resulting R-squared score of 84.91(%) with a standard deviation of 5.19(%) indicated a potential range of R-squared scores between 90.1(%) and 79.2(%). Given that we achieved an initial R-squared result of 90.36(%), this suggests the model's robustness in accurately predicting on unseen datasets.

To validate these findings, a KDE plot was utilized to visualize the model's performance by comparing predicted values against the test set values. The visualization 17 revealed the model's strong predictive capability within the price range of £25,000 to £75,000. However, beyond £75,000, the model tended to either overpredict or underpredict prices. This discrepancy might be attributed to limited data points and a scarcity of BEV cars within this higher price bracket.

Hyperparameter tuning - Using GridSearchCV, hyperparameter tuning for the XGBoost model was performed, targeting parameters like n(_)estimators, max(_)depth,

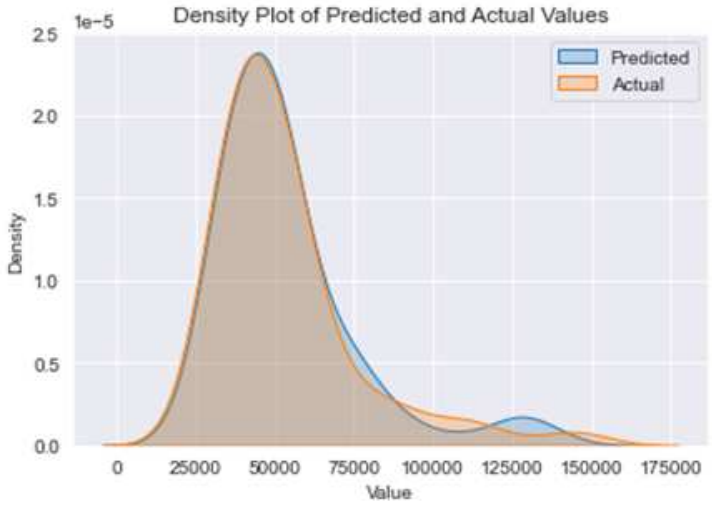


Figure 17: XGBoost KDE Plot – Predicted values vs Actual values

gamma, learning(_)rate, and min(_)child(_)weight. A comprehensive range was defined for each parameter: n(_)estimators spanned from 10 to 200 in steps of 5, max(_)depth varied from 1 to 24 in increments of 2, gamma ranged from 0.001 to 0.02 with steps of 0.005, learning(_)rate ranged from 0.01 to 0.1 with steps of 0.03, and min(_)child(_)weight was set from 1 to 20 with increments of 2. Due to their specificity and computational intensity, no additional combinations were explored. The cross-validation strategy employed 5 folds using the R-squared metric for scoring.

Upon fitting the grid model to the training datasets, the best score achieved was an R-squared value of 87.14(%), attained with specific parameters: gamma: 0.001, learning(_)rate: 0.099, max(_)depth: 5, min(_)child(_)weight: 5, and n(_)estimators: 60.

Subsequently, using these optimized grid parameters, predictions were made on the test set. The model exhibited impressive performance, showcasing an MAE of £4839, RMSE of £7683, and an R-squared value of 89.02(%). Notably, this model yielded the best MAE score among all the other models assessed, indicating an average prediction deviation of approximately £4839 for BEV prices.

7 Results Discussion

Model	Dataset	K-Fold CV R2, Std Dev	Hyperparameter Tuning (GRID cv)	MAE £	RMSE £	R-2
Linear Regression	1	75.17%, 8.65%	-	8005.60	11022.21	77.41%
Linear Regression	2	75.97%, 8.43%	-	7806.98	10929.64	77.79%
SVR	1	75.79%, 5.80%	-	8523.38	11290.15	76.30%
SVR	1	-	YES	8291.42	11003.47	77.49%
SVR	2	73.51%, 6.80%	-	8642.94	11510.83	75.36%
Decision Tree	1	75.49%, 10.18%	Parameters	5500.40	7933.20	88.29%
Decision Tree	2	67.53%, 21.74%	Parameters	7354.61	11276.31	76.36%
Random Forest	1	85.41%, 5.98%	Parameters	5045.11	7520.64	89.48%
Random Forest	1	-	YES	5028.62	7830.15	88.60%
Random Forest	2	83.59%, 6.81%	Parameters	5146.18	8004.04	88.09%
XGBoost Regression	1	84.91%, 5.19%	-	5105.07	7200.30	90.36%
XGBoost Regression	1	-	YES	4839.95	7683.59	89.02%

The identification and extraction of significant independent variables for our research drew insights from various scholarly papers. Research findings by Hidrue et al. (2011) highlighted the pivotal role of battery size in influencing purchase decisions of new vehicle owners. Manufacturers, aligning with this research, prioritize the battery size as a critical determinant in setting vehicle prices. Liu et al. (2021) Liu et al. (2021) further reinforced this notion by indicating that the battery component constitutes a substantial 70(%) of the total price of BEVs. Our decision tree model (regressor4) reflected these insights, showcasing battery size as the pivotal variable significantly impacting accuracy, resulting in an MAE of £5500 and an R-squared of 88.29(%). The coherence observed between our model's emphasis on battery size and the findings established in the literature review is in sync.

Additionally, insights from the fastest-car report Staats (2023) cited in our literature review component, affirms that higher performance in electric vehicles correlates with higher prices. The inclusion of the top speed variable in our analysis further supported this, with visual representations via scatter plots revealing a clear positive correlation between top speed and price. Moreover, our decision tree model assigned substantial weight to this feature, contributing significantly to achieving an R-squared value of 88.29(%).

Further, drawing from the study by Mammadov (2021), which emphasized the influence of horsepower on vehicle pricing, our findings align with their observations. While

their study employing a linear regression attained an R-squared measure of 83(%) on the test set, our employment of multiple models, especially XGBoost, outperformed this result. The XGBoost model achieved the highest R-squared measure of 90.36(%) on the test set, justifying the impact of horsepower on pricing. This correlation was reaffirmed by the feature(_)importance bar chart of the decision tree model, where horsepower emerged as one of the most influential features affecting price. Visual representations via scatter plots further validated this relationship between horsepower and UK(_)Price.

8 Conclusion and Limitations

Our research emphasizes the superiority of the XGBoost Regression model applied to dataset 1 among the array of models investigated. This model remarkably achieves an R-squared value of 90.36(%), signifying the robust fit of our chosen data to the model and its ability to effectively interpret dependencies within the dependent variable - the price of Battery Electric Vehicles. Even when considering the Mean Absolute Error, the XGBoost model, post hyperparameter tuning, exhibits an average prediction difference of £4839, further affirming its predictive ability.

The second most promising model for unseen data emerges as the Random Forest model, also applied to dataset 1. This model showcases a high R-squared score of 89.48(%) coupled with an MAE of £5045. Application of K-fold cross-validation justifies its stability and low variance compared to other models, a reliable indicator of avoiding overfitting and supporting our hypothesis.

Furthermore, the execution of decision trees on both dataset 1 and dataset 2 emphasizes the model's preference for a more extensive set of variables. The substantial disparity in MAE values, approximately £1700, between the two models reinforces the advantage of utilizing a richer feature set, emphasizing dataset 1 as the preferable choice. This positions the decision tree model applied to dataset 1 as the third-best performer when dealing with Electric Vehicle datasets, asserting the significance of feature richness in model performance.

The mean error in predicted prices, approximately £5000, can be attributed to unaccounted factors, notably brand name, car model, infrastructure setup, government incentives etc. These factors wield substantial influence on manufacturers decisions, compelling them to make adjustments in car pricing to comply with regulations and promote EV adoption. We hypothesize that this unexplained residual of £5000 signifies the value that could potentially enhance the model's accuracy by incorporating additional attributes. However, computational limitations restrict our capacity to expand the model due to higher processing requirements.

Moreover, the dataset's limited size, comprising only 367 data points, poses a risk of overfitting the model to this finite data pool. This constraint limits the model's capacity to generalize effectively beyond the provided dataset.

Additionally, our analysis, visualized through KDE plots generated for the XGBoost and Random Forest models, highlights a performance dip for BEV vehicles priced above £75000. This segment represents a niche market with limited BEV offerings, therefore impacting the model's predictive accuracy. In this price range, the model tends to either overestimate or underestimate prices, reflecting the scarcity of data points within this specific market segment.

The automobile industry's pricing dynamics for Electric Vehicles are significantly in-

fluenced by safety standards. While our dataset included safety ratings according to NCAP standards, it's crucial to acknowledge the significance of considering additional safety standards like NHTSA or IIHS. These standards hold substantial importance in the industry's assessment of safety parameters, impacting EV pricing strategies. Therefore, incorporating a broader spectrum of safety ratings beyond NCAP could offer a more comprehensive evaluation of safety aspects influencing EV pricing.

While our research delved into specific models like Linear Regression, SVR, Decision Trees, Random Forest, and XGBoost, there are several unexplored models such as Neural Network Regression, Lasso Regression, and Ridge Regression that warrant further analysis.

9 Package overview

Packages overview

Sr No	Python Packages
1	from bs4 import BeautifulSoup
2	import requests
3	import time
4	import pandas as pd
5	import numpy as np
6	import matplotlib.pyplot as plt
7	import re
8	from sklearn.impute import IterativeImputer
9	import seaborn as sb
10	from scipy import stats
11	from scipy.stats import norm
12	import sklearn.metrics as metrics
13	import statsmodels.api as sm
14	from sklearn.model_selection import train_test_split
15	from sklearn.linear_model import LinearRegression
16	from sklearn.model_selection import cross_val_score
17	from sklearn.preprocessing import MinMaxScaler
18	from sklearn.svm import SVR
19	from sklearn.model_selection import GridSearchCV
20	from sklearn.tree import DecisionTreeRegressor
21	from sklearn.ensemble import RandomForestRegressor
22	import plotly.graph_objects as go
23	from xgboost import XGBRegressor

References

A. Ajanovic and R. Haas. Dissemination of electric vehicles in urban areas: Major factors for success. *Energy*, 115:1451–1458, 11 2016. ISSN 03605442. doi: 10.1016/j.energy.2016.05.040. URL https://bit.ly/3GtiHHb.

M. B. Arias, M. Kim, and S. Bae. Prediction of electric vehicle charging-power demand in realistic urban traffic networks. *Applied Energy*, 195:738–753, 6 2017. ISSN 03062619. doi: 10.1016/j.apenergy.2017.02.021.

G. Baltas and C. Saridakis. Brand-name effects, segment differences, and product characteristics: an integrated model of the car market. *Journal of Product Brand Management*, 18:143–151, 4 2009. ISSN 1061-0421. doi: 10.1108/10610420910949040.

Deloitte. A new era accelerating toward 2020 — an automotive industry transformed, 2020. URL https://www2.deloitte.com/content/dam/Deloitte/in/Documents/manufacturing/a-new-era-auto-transformation-report-online.pdf.

Figenbaum, Erik, Kolbenstvedt, Marika, Elvebakk, and Beate. Electric vehicles–environmental, economic and practical aspects. *As seen by current and potential users*, 2014. URL https://www.toi.no/getfile.php/1337250-1411472277/Publikasjoner/T%C3%98I%20rapporter/2014/1329-2014/1329-2014-el.pdf.

J. H. Friedman. Recent advances in predictive (machine) learning. *Journal of Classification*, 23:175–197, 9 2006. ISSN 0176-4268. doi: 10.1007/s00357-006-0012-4.

T. Furcher, I. H. Giraldo, F. Rupalla, and A.-S. Smith. Electric-vehicle buyers demand new experiences. *McKinsey Company*, pages 1–6, 2022. URL https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/electric-vehicle-buyers-demand-new-experiences#/.

R. Galvin. Energy consumption effects of speed and acceleration in electric vehicles: Laboratory case studies and implications for drivers and policymakers. *Transportation Research Part D: Transport and Environment*, 53:234–248, 6 2017. ISSN 13619209. doi: 10.1016/j.trd.2017.04.020.

E. Gegic, B. Isakovic, D. Keco, Z. Masetic, and J. Kevric. Car price prediction using machine learning techniques. *TEM Journal*, 8:113–118, 2 2019. URL https://temjournal.com/content/81/TEMJournalFebruary2019_113_118.pdf.

M. K. Hidrue, G. R. Parsons, W. Kempton, and M. P. Gardner. Willingness to pay for electric vehicles and their attributes. *Resource and Energy Economics*, 33:686–705, 9 2011. ISSN 09287655. doi: 10.1016/j.reseneeco.2011.02.002. URL

http://sntbberry.cityofsanteeca.gov/sites/FanitaRanch/Public/Remainder%20of%20the%20Record/(14)%20Documents%20Received%20After%20Release%20of%20Draft%20EIR%20for%20Comment/A.%20Reference%20Documents/Tab%2002%20-%202011%20Willingness%20to%20pay%20for%20electric%20vehicles.pdf.

C. D. Higgins, M. Mohamed, and M. R. Ferguson. Size matters: How vehicle body type affects consumer preferences for electric vehicles. *Transportation Research Part A: Policy and Practice*, 100:182–201, 6 2017. ISSN 09658564. doi: 10.1016/j.tra.2017.04.014. URL https://www.researchgate.net/profile/Christopher-Higgins-5/publication/316582439_Size_Matters_How_Vehicle_Body_Type_Affects_Consumer_Preferences_for_Electric_Vehicles/links/59d3101baca2721f4369f438/Size-Matters-How-Vehicle-Body-Type-Affects-Consumer-Preferences-for-Electric-Vehicl pdf.

M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349:255–260, 7 2015. ISSN 0036-8075. doi: 10.1126/science.aaa8415. URL http://www.cs.cmu.edu/~tom/pubs/Science-ML-2015.pdf.

D. A. Kirsch. *History of the Electric Automobile: Battery-Only Powered Cars by Ernest H. Wakefield*, volume 36. SAE publications group, 7 1995. doi: 10.1353/tech.1995.0082.

V. V. Krishnan and B. I. Koshy. Evaluating the factors influencing purchase intention of electric vehicles in households owning conventional vehicles. *Case Studies on Transport Policy*, 9:1122–1129, 9 2021. ISSN 2213624X. doi: 10.1016/j.cstp.2021.05.013.

Lee, T. Wee, and S. Govindan. Emerging issues in car purchasing decision. *Academic Research International*, pages 1–5, 2014.

Z. Liu, J. Song, J. Kubal, N. Susarla, K. W. Knehr, E. Islam, P. Nelson, and S. Ahmed. Comparing total cost of ownership of battery electric vehicles and internal combustion engine vehicles. *Energy Policy*, 158:112564, 11 2021. ISSN 03014215. doi: 10.1016/j.enpol.2021.112564.

K. L. Lopez, C. Gagne, and M.-A. Gardner. Demand-side management using deep learning for smart charging of electric vehicles. *IEEE Transactions on Smart Grid*, 10:2683–2691, 5 2019. ISSN 1949-3053. doi: 10.1109/TSG.2018.2808247.

H. Mammadov. Car price prediction in the usa by using liner regression. *International Journal of Economic Behavior*, 11, 2021. URL file:///C:/Users/reece/Desktop/6+(21-12)+-+MammadovC.pdf.

K. Noor and S. Jan. Vehicle price prediction system using machine learning techniques. *International Journal of Computer Applications*, 167:27–31, 6 2017. ISSN 09758887. doi:

10.5120/ijca2017914373. URL https://www.ijcaonline.org/archives/volume167/number9/noor-2017-ijca-914373.pdf.

S. Ou, W. Li, J. Li, Z. Lin, X. He, J. Bouchard, and S. Przesmitzki. Relationships between vehicle pricing and features: Data driven analysis of the chinese vehicle market. *Energies*, 13:3088, 6 2020. ISSN 1996-1073. doi: 10.3390/en13123088.

I. G. E. Outlook. Trends in electric light-duty vehicles, 2023. URL https://www.iea.org/reports/global-ev-outlook-2023/trends-in-electric-light-duty-vehicles.

J. Pesantez-Narvaez, M. Guillen, and M. Alcañiz. Predicting motor insurance claims using telematics data—xgboost versus logistic regression. *Risks*, 7:70, 6 2019. ISSN 2227-9091. doi: 10.3390/risks7020070.

S. Pudaruth. Predicting the price of used cars using machine learning techniques. *International Journal of Information Computation Technology*, 4:753–764, 2014. URL https://www.researchgate.net/publication/319306871_Predicting_the_Price_of_Used_Cars_using_Machine_Learning_Techniques.

J. Quiros-Tortos, L. Ochoa, and T. Butler. How electric vehicles and the grid work together: Lessons learned from one of the largest electric vehicle trials in the world. *IEEE Power and Energy Magazine*, 21:64–76, 3 2023. ISSN 1540-7977. doi: 10.1109/MPAE.2023.10083043.

K. Samruddhi and R. A. Kumar. Used car price prediction using k-nearest neighbor based model. *International Journal of Innovative Research in Applied Sciences and Engineering*, 4:629–632, 8 2020. ISSN 2456-8910. doi: 10.29027/IJIRASE.v4.i2.2020.629-632. URL https://ijirase.com/assets/paper/issue_1/volume_4/V4-Issue-2-629-632.pdf.

S. Sreenivasa. Radial basis function (rbf) kernel: The go-to kernel, 10 2020. URL https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a.

N. Staats. Why you shouldn't go for the fastest, most powerful electric vehicle. *Top Speed*, 2023. URL https://www.topspeed.com/why-shouldnt-buy-fastest-most-powerful-electric-vehicle/.

S. Sun. Number of vehicles in operation across india from financial year 1951 to 2020, 2023. URL https://www.statista.com/statistics/664729/total-number-of-vehicles-india/.

X. Sun, Z. Li, X. Wang, and C. Li. Technology development of electric vehicles: A review. *Energies*, 13:1–15, 12 2019. ISSN 1996-1073. doi: 10.3390/en13010090.

M. Taddy. *Business Data Science: Combining Machine Learning and Economics to Optimize, Automate, and Accelerate Business Decisions*, volume 1. McGraw Hill, 1 edition, 8 2019.

P. S. Thakkar1, M. Donga2, J. Panjwari3, D. Kotak4, and K. Savalia5. Price prediction of electric vehicles. *International Journal of Research Publication and Reviews*, 4, 4 2023. URL https://ijrpr.com/uploads/V4ISSUE4/IJRPR11942.pdf.

I. Ullah, K. Liu, T. Yamamoto, M. Zahid, and A. Jamal. Prediction of electric vehicle charging duration time using ensemble machine learning algorithm and shapley additive explanations. *International Journal of Energy Research*, 46:15211–15230, 9 2022. ISSN 0363-907X. doi: 10.1002/er.8219.

10 Appendix

In [1]:
```
#USE THIS FILE AS A CODE FOR THE THESIS DOCUMENT - FINAL FILE
```

Two part scraping was carried out - First part included only outside variables from the website

Second Part included 4 new variables from the inner pages.

In [2]:
```
from bs4 import BeautifulSoup
import requests
import time
import pandas as pd
```

In [3]:
```
# result = requests.get("https://ev-database.org/#sort:path~type~order=.rank~number~desc|range-slider-range:pre
```

In [4]:
```
# content = result.text
```

In [5]:
```
# soup = BeautifulSoup(content,'lxml')
# print(soup.prettify())
```

In [6]:
```
#Save the soup object to a file
# with open("HTML_output_file.html", "w") as file:
#     file.write(str(soup))
```

In [7]:
```
# Read the soup object from a file
with open('HTML_output_file.html', 'r') as f:
    contents = f.read()
    soup = BeautifulSoup(contents, 'lxml')
```

In [8]:
```
box= soup.find('div',class_='list')
```

In [1]:
```
# box
```

In [10]:
```
car_list = box.find_all('div',class_ = 'list-item')
```

In [11]:
```
data = []

for car in car_list:
    # Extracting title and brand
    title = car.find('div', class_='title-wrap').find('a', class_='title').text.strip()
    brand, model = title.split(maxsplit=1)

    # Extracting battery size
    battery_size = car.find('span', class_='battery').text.strip()

    # Extracting towing capacity
    towing_capacity = car.find('span', class_='towweight').text.strip()

    # # Extracting market segment
    # market_segment = car.find('span', class_='size-d').text.strip()

    # Extracting number of seats
    if 'seats-5 fas fa-user' in str(car):
        seats= 5
    elif 'seats-4 fas fa-user' in str(car):
        seats = 4
    else:
        seats = 7

    # Extracting acceleration
    acceleration = car.find('span', class_='acceleration').text.strip()

    # Extracting top speed
    top_speed = car.find('span', class_='topspeed').text.strip()

    # Extracting efficiency
    efficiency = car.find('span', class_='efficiency').text.strip()

    # Extracting fast charge speed
    fast_charge_speed = car.find('span', class_='fastcharge_speed_print').text.strip()

    # Extracting erange_real
    erange_real = car.find('span', class_='erange_real').text.strip()

    # Checking if it's all-wheel drive or rear wheel drive
    if "achter far fa-circle" in str(car):
        drive_type = "Rear Wheel Drive"
    elif "awd fas fa-circle" in str(car):
        drive_type = "All Wheel Drive"
    else:
        drive_type = "Forward Wheel Drive"

    # Extracting country UK price
```

```
    price_uk = car.find('span', class_='country_uk').text.strip()

    # Extracting country NL price
    price_nl = car.find('span', class_='country_nl').text.strip()

    # Extracting country DE price
    price_de = car.find('span', class_='country_de').text.strip()


    data.append([brand, model, battery_size, towing_capacity, seats, acceleration, top_speed, efficiency, fast_
```

In [12]:
```
columns = ['Brand', 'Model', 'Battery Size', 'Towing Capacity', 'Number of Seats', 'Acceleration', 'Top Speed',
df = pd.DataFrame(data, columns=columns)
```

In [13]:
```
df['Country UK Price']=df['Country UK Price'].str.replace('*', '',regex=True)
df['Country UK Price']=df['Country UK Price'].str.replace('£', '',regex=True)
df['Country NL Price']=df['Country NL Price'].str.replace('€', '',regex=True)
df['Country DE Price']=df['Country DE Price'].str.replace('€', '',regex=True)
```

In [2]:
```
# df
```

In [15]:
```
df.to_csv('Car_Dataset_Part1.csv',index=False)
```

Webscraping Part 2

In [16]:
```
car_links = [a['href'] for a in box.find_all('a',href=True)]
```

In [18]:
```
# #adding prefix https to the car links
# text_to_add = 'https://ev-database.org'
# my_list = [text_to_add + item for item in car_links]
```

In [ ]:
```
text_to_add = 'https://ev-database.org'
my_list = []
for item in car_links:
    url = text_to_add + item
    if url not in my_list:  # Check if the URL is already in my_list
        my_list.append(url)
# print(my_list)
```

In [4]:
```
# my_list
```

In [26]:
```
#Getting the internal variables from the website. Safety rating, cargo volume, turning radius & home charge spe
df_list = []  # List to store individual DataFrames
#Defining a user agent - Basically a chrome browser.
processed_links = set()  # Creating an empty set to keep track of processed links
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
count = 0
links_limit = 30

for link in my_list:
    try:
        if link in processed_links: #if the link has been entered once we can skip it
            print(f"Link {link} already processed. Skipping...")
            continue

        response = requests.get(link,headers=headers,timeout=10)#sets a timer for HTTPS requests
        if response.status_code == 200: #if request is sucessfull
            page_content = response.content
            inner_soup = BeautifulSoup(page_content, 'lxml')
            inner_box = inner_soup.find('div', class_='content')

            h1_tag_element = inner_box.find('header', class_='sub-header')
            safety_rating_element = inner_box.find('td', string='Safety Rating')
            cargo_volume_element = inner_box.find('td', string='Cargo Volume')
            turning_element = inner_box.find('td', string='Turning Circle')
            home_charge_speed_element = inner_box.find('td', string='Charge Speed')
            total_power_element = inner_box.find('td', string='Total Power')
            total_power_element_star = inner_box.find('td', string='Total Power *')
            car_body_element = inner_box.find('td', string='Car Body')

            data = {} #Dictionary to store data

            if h1_tag_element:
                data['Car Name'] = h1_tag_element.find_next('h1').text.strip()
            else:
                data['Car Name'] = None

            if safety_rating_element:
                safety_rating_icons = safety_rating_element.find_next('td').find_all('i')
                data['Safety Rating'] = len(safety_rating_icons)
            else:
```

```
                data['Safety Rating'] = None

            if cargo_volume_element:
                data['Cargo Volume'] = cargo_volume_element.find_next('td').text.strip()
            else:
                data['Cargo Volume'] = None

            if turning_element:
                data['Turning Radius'] = turning_element.find_next('td').text.strip()
            else:
                data['Turning Radius'] = None

            if home_charge_speed_element:
                data['Home Charge Speed'] = home_charge_speed_element.find_next('td').text.strip()
            else:
                data['Home Charge Speed'] = None

            if total_power_element:
                data['Horsepower'] = total_power_element.find_next('td').text.strip()
            elif total_power_element_star:
                data['Horsepower'] = total_power_element_star.find_next('td').text.strip()
            else:
                data['Horsepower'] = None

            if car_body_element:
                data['Vehicle Segment'] = car_body_element.find_next('td').text.strip()
            else:
                data['Vehicle Segment'] = None

            df_list.append(pd.DataFrame([data]))  # Append DataFrame to the list
            processed_links.add(link) # adding the link to the processed link set
            time.sleep(40) #adding a delay between requests.
            count += 1

            if count==links_limit: #count the number of links processed and check if there are 30
                break
        else:
            print(f"Failed to extract content for link: {link}")
    except requests.exceptions.RequestException as e:
        print(f"Error with link {link}: {e}")
    except Exception as e:
        print(f"Error processing link {link}: {e}")

# Concatenate all DataFrames in df_list
df_1 = pd.concat(df_list, ignore_index=True)
df_1
```

Out[26]:

	Car Name	Safety Rating	Cargo Volume	Turning Radius	Home Charge Speed	Horsepower	Vehicle Segment
0	Tesla Model Y Long Range Dual Motor	5	854 L	12.1 m	54 km/h	378 kW (514 PS)	SUV
1	BYD ATTO 3	5	440 L	10.7 m	34 km/h	150 kW (204 PS)	SUV
2	Tesla Model Y	5	854 L	12.1 m	57 km/h	220 kW (299 PS)	SUV
3	MG MG4 Electric 64 kWh	5	363 L	10.6 m	55 km/h	150 kW (204 PS)	Hatchback
4	Tesla Model 3	None	594 L	No Data	68 km/h	208 kW (283 PS)	Sedan
5	BMW i4 eDrive40	5	470 L	12.5 m	60 km/h	250 kW (340 PS)	Sedan
6	BYD SEAL 82.5 kWh AWD Excellence	None	400 L	11.4 m	55 km/h	390 kW (530 PS)	Sedan
7	Tesla Model Y Performance	5	854 L	12.1 m	52 km/h	393 kW (534 PS)	SUV
8	BMW iX1 xDrive30	None	490 L	11.9 m	53 km/h	230 kW (313 PS)	SUV
9	Tesla Model S Plaid	5	709 L	12.3 m	55 km/h	760 kW (1033 PS)	Liftback Sedan
10	Tesla Model 3 Long Range Dual Motor	None	594 L	No Data	63 km/h	366 kW (498 PS)	Sedan
11	Skoda Enyaq iV 80	5	585 L	9.3 m	53 km/h	150 kW (204 PS)	SUV
12	BMW iX xDrive40	5	500 L	12.8 m	47 km/h	240 kW (326 PS)	SUV
13	Kia Niro EV	5	475 L	10.6 m	56 km/h	150 kW (204 PS)	SUV
14	Volvo EX30 Single Motor ER	None	318 L	10.6 m	54 km/h	200 kW (272 PS)	SUV
15	Hyundai Kona Electric 64 kWh	None	332 L	10.6 m	58 km/h	150 kW (204 PS)	SUV
16	Tesla Model S Dual Motor	5	709 L	12.3 m	57 km/h	504 kW (685 PS)	Liftback Sedan
17	BYD HAN	None	410 L	12.3 m	31 km/h	380 kW (517 PS)	Sedan
18	Dacia Spring Electric 45	5	300 L	9.5 m	37 km/h	33 kW (45 PS)	Hatchback
19	BMW iX3	None	510 L	12.1 m	49 km/h	210 kW (286 PS)	SUV
20	BMW i4 eDrive35	5	470 L	12.5 m	60 km/h	210 kW (286 PS)	Sedan
21	Mercedes EQC 400 4MATIC	5	500 L	No Data	43 km/h	300 kW (408 PS)	SUV
22	Renault Megane E-Tech EV60 220hp	5	440 L	10.4 m	110 km/h	160 kW (218 PS)	SUV
23	BMW iX xDrive50	5	500 L	12.8 m	45 km/h	385 kW (523 PS)	SUV
24	BMW i4 M50	5	470 L	12.5 m	52 km/h	400 kW (544 PS)	Sedan
25	Smart #1	5	323 L	11 m	34 km/h	200 kW (272 PS)	SUV
26	BYD DOLPHIN 60.4 kWh	None	345 L	10.5 m	53 km/h	150 kW (204 PS)	Hatchback
27	BMW i5 eDrive40 Sedan	None	490 L	12.3 m	55 km/h	250 kW (340 PS)	Sedan
28	Renault Zoe ZE50 R110	5	338 L	10.6 m	110 km/h	80 kW (109 PS)	Hatchback
29	BMW i7 xDrive60	None	500 L	12.3 m	47 km/h	400 kW (544 PS)	Sedan

In [ ]:

```
df_list = []  # List to store individual DataFrames
processed_links_1 = set()  # Creating an empty set to keep track of processed links
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
count = 0
links_limit = 30

for link in my_list:
    try:
        if link in processed_links or link in processed_links_1:
            print(f"Link {link} already processed. Skipping...")
            continue

        response = requests.get(link,headers=headers,timeout=10)#sets a timer for HTTPS requests
        if response.status_code == 200: #if request is sucessfull
            page_content = response.content
            inner_soup = BeautifulSoup(page_content, 'lxml')
            inner_box = inner_soup.find('div', class_='content')

            h1_tag_element = inner_box.find('header', class_='sub-header')
            safety_rating_element = inner_box.find('td', string='Safety Rating')
            cargo_volume_element = inner_box.find('td', string='Cargo Volume')
            turning_element = inner_box.find('td', string='Turning Circle')
            home_charge_speed_element = inner_box.find('td', string='Charge Speed')
            total_power_element = inner_box.find('td', string='Total Power')
            total_power_element_star = inner_box.find('td', string='Total Power *')
            car_body_element = inner_box.find('td', string='Car Body')

            data = {}  # Dictionary to store data

            if h1_tag_element:
                data['Car Name'] = h1_tag_element.find_next('h1').text.strip()
            else:
```

```
                data['Car Name'] = None

            if safety_rating_element:
                safety_rating_icons = safety_rating_element.find_next('td').find_all('i')
                data['Safety Rating'] = len(safety_rating_icons)
            else:
                data['Safety Rating'] = None

            if cargo_volume_element:
                data['Cargo Volume'] = cargo_volume_element.find_next('td').text.strip()
            else:
                data['Cargo Volume'] = None

            if turning_element:
                data['Turning Radius'] = turning_element.find_next('td').text.strip()
            else:
                data['Turning Radius'] = None

            if home_charge_speed_element:
                data['Home Charge Speed'] = home_charge_speed_element.find_next('td').text.strip()
            else:
                data['Home Charge Speed'] = None

            if total_power_element:
                data['Horsepower'] = total_power_element.find_next('td').text.strip()
            elif total_power_element_star:
                data['Horsepower'] = total_power_element_star.find_next('td').text.strip()
            else:
                data['Horsepower'] = None

            if car_body_element:
                data['Vehicle Segment'] = car_body_element.find_next('td').text.strip()
            else:
                data['Vehicle Segment'] = None

            df_list.append(pd.DataFrame([data]))  # Append DataFrame to the list
            processed_links_1.add(link) # adding the link to the processed link set
            time.sleep(40) #adding a delay between requests.
            count += 1

            if count==links_limit: #count the number of links processed and check if there are 30
                break
        else:
            print(f"Failed to extract content for link: {link}")
    except requests.exceptions.RequestException as e:
        print(f"Error with link {link}: {e}")
    except Exception as e:
        print(f"Error processing link {link}: {e}")

# Concatenate all DataFrames in df_list
df_2 = pd.concat(df_list, ignore_index=True)

df_2
```

In [ ]:

```
df_list = []  # List to store individual DataFrames
processed_links_2 = set()  # Creating an empty set to keep track of processed links
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
count = 0
links_limit = 30

for link in my_list:
    try:
        if link in processed_links or link in processed_links_1 or link in processed_links_2:
            print(f"Link {link} already processed. Skipping...")
            continue

        response = requests.get(link,headers=headers,timeout=10)#sets a timer for HTTPS requests
        if response.status_code == 200: #if request is sucessfull
            page_content = response.content
            inner_soup = BeautifulSoup(page_content, 'lxml')
            inner_box = inner_soup.find('div', class_='content')

            h1_tag_element = inner_box.find('header', class_='sub-header')
            safety_rating_element = inner_box.find('td', string='Safety Rating')
            cargo_volume_element = inner_box.find('td', string='Cargo Volume')
            turning_element = inner_box.find('td', string='Turning Circle')
            home_charge_speed_element = inner_box.find('td', string='Charge Speed')
            total_power_element = inner_box.find('td', string='Total Power')
            total_power_element_star = inner_box.find('td', string='Total Power *')
            car_body_element = inner_box.find('td', string='Car Body')

            data = {}  # Dictionary to store data

            if h1_tag_element:
                data['Car Name'] = h1_tag_element.find_next('h1').text.strip()
            else:
                data['Car Name'] = None
```

```
            if safety_rating_element:
                safety_rating_icons = safety_rating_element.find_next('td').find_all('i')
                data['Safety Rating'] = len(safety_rating_icons)
            else:
                data['Safety Rating'] = None

            if cargo_volume_element:
                data['Cargo Volume'] = cargo_volume_element.find_next('td').text.strip()
            else:
                data['Cargo Volume'] = None

            if turning_element:
                data['Turning Radius'] = turning_element.find_next('td').text.strip()
            else:
                data['Turning Radius'] = None

            if home_charge_speed_element:
                data['Home Charge Speed'] = home_charge_speed_element.find_next('td').text.strip()
            else:
                data['Home Charge Speed'] = None

            if total_power_element:
                data['Horsepower'] = total_power_element.find_next('td').text.strip()
            elif total_power_element_star:
                data['Horsepower'] = total_power_element_star.find_next('td').text.strip()
            else:
                data['Horsepower'] = None

            if car_body_element:
                data['Vehicle Segment'] = car_body_element.find_next('td').text.strip()
            else:
                data['Vehicle Segment'] = None

            df_list.append(pd.DataFrame([data]))  # Append DataFrame to the list
            processed_links_2.add(link) # adding the link to the processed link set
            time.sleep(40) #adding a delay between requests.
            count += 1

            if count==links_limit: #count the number of links processed and check if there are 30
                break
        else:
            print(f"Failed to extract content for link: {link}")
    except requests.exceptions.RequestException as e:
        print(f"Error with link {link}: {e}")
    except Exception as e:
        print(f"Error processing link {link}: {e}")

# Concatenate all DataFrames in df_list
df_3 = pd.concat(df_list, ignore_index=True)

df_3
```

In [ ]:

```
df_list = []  # List to store individual DataFrames
processed_links_3 = set()  # Creating an empty set to keep track of processed links
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
count = 0
links_limit = 35

for link in my_list:
    try:
        if link in processed_links or link in processed_links_1 or link in processed_links_2 or link in process
            print(f"Link {link} already processed. Skipping...")
            continue

        response = requests.get(link,headers=headers,timeout=10)#sets a timer for HTTPS requests
        if response.status_code == 200: #if request is sucessfull
            page_content = response.content
            inner_soup = BeautifulSoup(page_content, 'lxml')
            inner_box = inner_soup.find('div', class_='content')

            h1_tag_element = inner_box.find('header', class_='sub-header')
            safety_rating_element = inner_box.find('td', string='Safety Rating')
            cargo_volume_element = inner_box.find('td', string='Cargo Volume')
            turning_element = inner_box.find('td', string='Turning Circle')
            home_charge_speed_element = inner_box.find('td', string='Charge Speed')
            total_power_element = inner_box.find('td', string='Total Power')
            total_power_element_star = inner_box.find('td', string='Total Power *')
            car_body_element = inner_box.find('td', string='Car Body')

            data = {}  # Dictionary to store data

            if h1_tag_element:
                data['Car Name'] = h1_tag_element.find_next('h1').text.strip()
            else:
                data['Car Name'] = None

            if safety_rating_element:
                safety_rating_icons = safety_rating_element.find_next('td').find_all('i')
```

```
                data['Safety Rating'] = len(safety_rating_icons)
            else:
                data['Safety Rating'] = None

            if cargo_volume_element:
                data['Cargo Volume'] = cargo_volume_element.find_next('td').text.strip()
            else:
                data['Cargo Volume'] = None

            if turning_element:
                data['Turning Radius'] = turning_element.find_next('td').text.strip()
            else:
                data['Turning Radius'] = None

            if home_charge_speed_element:
                data['Home Charge Speed'] = home_charge_speed_element.find_next('td').text.strip()
            else:
                data['Home Charge Speed'] = None

            if total_power_element:
                data['Horsepower'] = total_power_element.find_next('td').text.strip()
            elif total_power_element_star:
                data['Horsepower'] = total_power_element_star.find_next('td').text.strip()
            else:
                data['Horsepower'] = None

            if car_body_element:
                data['Vehicle Segment'] = car_body_element.find_next('td').text.strip()
            else:
                data['Vehicle Segment'] = None

            df_list.append(pd.DataFrame([data]))  # Append DataFrame to the list
            processed_links_3.add(link) # adding the link to the processed link set
            time.sleep(40) #adding a delay between requests.
            count += 1

            if count==links_limit: #count the number of links processed and check if there are 30
                break
        else:
            print(f"Failed to extract content for link: {link}")
    except requests.exceptions.RequestException as e:
        print(f"Error with link {link}: {e}")
    except Exception as e:
        print(f"Error processing link {link}: {e}")

# Concatenate all DataFrames in df_list
df_4 = pd.concat(df_list, ignore_index=True)

df_4
```

In [ ]:

```
df_list = []  # List to store individual DataFrames
processed_links_4 = set()  # Creating an empty set to keep track of processed links
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
count = 0
links_limit = 40

for link in my_list:
    try:
        if link in processed_links or link in processed_links_1 or link in processed_links_2 or link in process
            print(f"Link {link} already processed. Skipping...")
            continue

        response = requests.get(link,headers=headers,timeout=10)#sets a timer for HTTPS requests
        if response.status_code == 200: #if request is sucessfull
            page_content = response.content
            inner_soup = BeautifulSoup(page_content, 'lxml')
            inner_box = inner_soup.find('div', class_='content')

            h1_tag_element = inner_box.find('header', class_='sub-header')
            safety_rating_element = inner_box.find('td', string='Safety Rating')
            cargo_volume_element = inner_box.find('td', string='Cargo Volume')
            turning_element = inner_box.find('td', string='Turning Circle')
            home_charge_speed_element = inner_box.find('td', string='Charge Speed')
            total_power_element = inner_box.find('td', string='Total Power')
            total_power_element_star = inner_box.find('td', string='Total Power *')
            car_body_element = inner_box.find('td', string='Car Body')

            data = {}  # Dictionary to store data

            if h1_tag_element:
                data['Car Name'] = h1_tag_element.find_next('h1').text.strip()
            else:
                data['Car Name'] = None

            if safety_rating_element:
                safety_rating_icons = safety_rating_element.find_next('td').find_all('i')
                data['Safety Rating'] = len(safety_rating_icons)
            else:
```

```
                data['Safety Rating'] = None

            if cargo_volume_element:
                data['Cargo Volume'] = cargo_volume_element.find_next('td').text.strip()
            else:
                data['Cargo Volume'] = None

            if turning_element:
                data['Turning Radius'] = turning_element.find_next('td').text.strip()
            else:
                data['Turning Radius'] = None

            if home_charge_speed_element:
                data['Home Charge Speed'] = home_charge_speed_element.find_next('td').text.strip()
            else:
                data['Home Charge Speed'] = None

            if total_power_element:
                data['Horsepower'] = total_power_element.find_next('td').text.strip()
            elif total_power_element_star:
                data['Horsepower'] = total_power_element_star.find_next('td').text.strip()
            else:
                data['Horsepower'] = None

            if car_body_element:
                data['Vehicle Segment'] = car_body_element.find_next('td').text.strip()
            else:
                data['Vehicle Segment'] = None

            df_list.append(pd.DataFrame([data]))  # Append DataFrame to the list
            processed_links_4.add(link) # adding the link to the processed link set
            time.sleep(40) #adding a delay between requests.
            count += 1

            if count==links_limit: #count the number of links processed and check if there are 30
                break
        else:
            print(f"Failed to extract content for link: {link}")
    except requests.exceptions.RequestException as e:
        print(f"Error with link {link}: {e}")
    except Exception as e:
        print(f"Error processing link {link}: {e}")

# Concatenate all DataFrames in df_list
df_5 = pd.concat(df_list, ignore_index=True)

df_5
```

In [ ]:

```
df_list = []  # List to store individual DataFrames
processed_links_5 = set()  # Creating an empty set to keep track of processed links
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
count = 0
links_limit = 40

for link in my_list:
    try:
        if link in processed_links or link in processed_links_1 or link in processed_links_2 or link in process
            print(f"Link {link} already processed. Skipping...")
            continue

        response = requests.get(link,headers=headers,timeout=10)#sets a timer for HTTPS requests
        if response.status_code == 200: #if request is sucessfull
            page_content = response.content
            inner_soup = BeautifulSoup(page_content, 'lxml')
            inner_box = inner_soup.find('div', class_='content')

            h1_tag_element = inner_box.find('header', class_='sub-header')
            safety_rating_element = inner_box.find('td', string='Safety Rating')
            cargo_volume_element = inner_box.find('td', string='Cargo Volume')
            turning_element = inner_box.find('td', string='Turning Circle')
            home_charge_speed_element = inner_box.find('td', string='Charge Speed')
            total_power_element = inner_box.find('td', string='Total Power')
            total_power_element_star = inner_box.find('td', string='Total Power *')
            car_body_element = inner_box.find('td', string='Car Body')

            data = {}  # Dictionary to store data

            if h1_tag_element:
                data['Car Name'] = h1_tag_element.find_next('h1').text.strip()
            else:
                data['Car Name'] = None

            if safety_rating_element:
                safety_rating_icons = safety_rating_element.find_next('td').find_all('i')
                data['Safety Rating'] = len(safety_rating_icons)
            else:
                data['Safety Rating'] = None
```

```
            if cargo_volume_element:
                data['Cargo Volume'] = cargo_volume_element.find_next('td').text.strip()
            else:
                data['Cargo Volume'] = None

            if turning_element:
                data['Turning Radius'] = turning_element.find_next('td').text.strip()
            else:
                data['Turning Radius'] = None

            if home_charge_speed_element:
                data['Home Charge Speed'] = home_charge_speed_element.find_next('td').text.strip()
            else:
                data['Home Charge Speed'] = None

            if total_power_element:
                data['Horsepower'] = total_power_element.find_next('td').text.strip()
            elif total_power_element_star:
                data['Horsepower'] = total_power_element_star.find_next('td').text.strip()
            else:
                data['Horsepower'] = None

            if car_body_element:
                data['Vehicle Segment'] = car_body_element.find_next('td').text.strip()
            else:
                data['Vehicle Segment'] = None

            df_list.append(pd.DataFrame([data]))  # Append DataFrame to the list
            processed_links_5.add(link) # adding the link to the processed link set
            time.sleep(40) #adding a delay between requests.
            count += 1

            if count==links_limit: #count the number of links processed and check if there are 30
                break
        else:
            print(f"Failed to extract content for link: {link}")
    except requests.exceptions.RequestException as e:
        print(f"Error with link {link}: {e}")
    except Exception as e:
        print(f"Error processing link {link}: {e}")

# Concatenate all DataFrames in df_list
df_6 = pd.concat(df_list, ignore_index=True)

df_6
```

In [ ]:

```
df_list = []  # List to store individual DataFrames
processed_links_6 = set()  # Creating an empty set to keep track of processed links
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
count = 0
links_limit = 40

for link in my_list:
    try:
        if link in processed_links or link in processed_links_1 or link in processed_links_2 or link in process
            print(f"Link {link} already processed. Skipping...")
            continue

        response = requests.get(link,headers=headers,timeout=10)#sets a timer for HTTPS requests
        if response.status_code == 200: #if request is sucessfull
            page_content = response.content
            inner_soup = BeautifulSoup(page_content, 'lxml')
            inner_box = inner_soup.find('div', class_='content')

            h1_tag_element = inner_box.find('header', class_='sub-header')
            safety_rating_element = inner_box.find('td', string='Safety Rating')
            cargo_volume_element = inner_box.find('td', string='Cargo Volume')
            turning_element = inner_box.find('td', string='Turning Circle')
            home_charge_speed_element = inner_box.find('td', string='Charge Speed')
            total_power_element = inner_box.find('td', string='Total Power')
            total_power_element_star = inner_box.find('td', string='Total Power *')
            car_body_element = inner_box.find('td', string='Car Body')

            data = {}  # Dictionary to store data

            if h1_tag_element:
                data['Car Name'] = h1_tag_element.find_next('h1').text.strip()
            else:
                data['Car Name'] = None

            if safety_rating_element:
                safety_rating_icons = safety_rating_element.find_next('td').find_all('i')
                data['Safety Rating'] = len(safety_rating_icons)
            else:
                data['Safety Rating'] = None

            if cargo_volume_element:
                data['Cargo Volume'] = cargo_volume_element.find_next('td').text.strip()
```

```
            else:
                data['Cargo Volume'] = None

            if turning_element:
                data['Turning Radius'] = turning_element.find_next('td').text.strip()
            else:
                data['Turning Radius'] = None

            if home_charge_speed_element:
                data['Home Charge Speed'] = home_charge_speed_element.find_next('td').text.strip()
            else:
                data['Home Charge Speed'] = None

            if total_power_element:
                data['Horsepower'] = total_power_element.find_next('td').text.strip()
            elif total_power_element_star:
                data['Horsepower'] = total_power_element_star.find_next('td').text.strip()
            else:
                data['Horsepower'] = None

            if car_body_element:
                data['Vehicle Segment'] = car_body_element.find_next('td').text.strip()
            else:
                data['Vehicle Segment'] = None

            df_list.append(pd.DataFrame([data]))  # Append DataFrame to the list
            processed_links_6.add(link) # adding the link to the processed link set
            time.sleep(40) #adding a delay between requests.
            count += 1

            if count==links_limit: #count the number of links processed and check if there are 30
                break
        else:
            print(f"Failed to extract content for link: {link}")
    except requests.exceptions.RequestException as e:
        print(f"Error with link {link}: {e}")
    except Exception as e:
        print(f"Error processing link {link}: {e}")

# Concatenate all DataFrames in df_list
df_7 = pd.concat(df_list, ignore_index=True)

df_7
```

In [ ]:

```
df_list = []  # List to store individual DataFrames
processed_links_7 = set()  # Creating an empty set to keep track of processed links
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
count = 0
links_limit = 40

for link in my_list:
    try:
        if link in processed_links or link in processed_links_1 or link in processed_links_2 or link in process
            print(f"Link {link} already processed. Skipping...")
            continue

        response = requests.get(link,headers=headers,timeout=10)#sets a timer for HTTPS requests
        if response.status_code == 200: #if request is sucessfull
            page_content = response.content
            inner_soup = BeautifulSoup(page_content, 'lxml')
            inner_box = inner_soup.find('div', class_='content')

            h1_tag_element = inner_box.find('header', class_='sub-header')
            safety_rating_element = inner_box.find('td', string='Safety Rating')
            cargo_volume_element = inner_box.find('td', string='Cargo Volume')
            turning_element = inner_box.find('td', string='Turning Circle')
            home_charge_speed_element = inner_box.find('td', string='Charge Speed')
            total_power_element = inner_box.find('td', string='Total Power')
            total_power_element_star = inner_box.find('td', string='Total Power *')
            car_body_element = inner_box.find('td', string='Car Body')

            data = {}  # Dictionary to store data

            if h1_tag_element:
                data['Car Name'] = h1_tag_element.find_next('h1').text.strip()
            else:
                data['Car Name'] = None

            if safety_rating_element:
                safety_rating_icons = safety_rating_element.find_next('td').find_all('i')
                data['Safety Rating'] = len(safety_rating_icons)
            else:
                data['Safety Rating'] = None

            if cargo_volume_element:
                data['Cargo Volume'] = cargo_volume_element.find_next('td').text.strip()
            else:
                data['Cargo Volume'] = None
```

```
            if turning_element:
                data['Turning Radius'] = turning_element.find_next('td').text.strip()
            else:
                data['Turning Radius'] = None

            if home_charge_speed_element:
                data['Home Charge Speed'] = home_charge_speed_element.find_next('td').text.strip()
            else:
                data['Home Charge Speed'] = None

            if total_power_element:
                data['Horsepower'] = total_power_element.find_next('td').text.strip()
            elif total_power_element_star:
                data['Horsepower'] = total_power_element_star.find_next('td').text.strip()
            else:
                data['Horsepower'] = None

            if car_body_element:
                data['Vehicle Segment'] = car_body_element.find_next('td').text.strip()
            else:
                data['Vehicle Segment'] = None

            df_list.append(pd.DataFrame([data]))  # Append DataFrame to the list
            processed_links_7.add(link) # adding the link to the processed link set
            time.sleep(40) #adding a delay between requests.
            count += 1

            if count==links_limit: #count the number of links processed and check if there are 30
                break
        else:
            print(f"Failed to extract content for link: {link}")
    except requests.exceptions.RequestException as e:
        print(f"Error with link {link}: {e}")
    except Exception as e:
        print(f"Error processing link {link}: {e}")

# Concatenate all DataFrames in df_list
df_8 = pd.concat(df_list, ignore_index=True)

df_8
```

In [ ]:

```
df_list = []  # List to store individual DataFrames
processed_links_8 = set()  # Creating an empty set to keep track of processed links
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
count = 0
links_limit = 40

for link in my_list:
    try:
        if link in processed_links or link in processed_links_1 or link in processed_links_2 or link in process
            print(f"Link {link} already processed. Skipping...")
            continue

        response = requests.get(link,headers=headers,timeout=10)#sets a timer for HTTPS requests
        if response.status_code == 200: #if request is sucessfull
            page_content = response.content
            inner_soup = BeautifulSoup(page_content, 'lxml')
            inner_box = inner_soup.find('div', class_='content')

            h1_tag_element = inner_box.find('header', class_='sub-header')
            safety_rating_element = inner_box.find('td', string='Safety Rating')
            cargo_volume_element = inner_box.find('td', string='Cargo Volume')
            turning_element = inner_box.find('td', string='Turning Circle')
            home_charge_speed_element = inner_box.find('td', string='Charge Speed')
            total_power_element = inner_box.find('td', string='Total Power')
            total_power_element_star = inner_box.find('td', string='Total Power *')
            car_body_element = inner_box.find('td', string='Car Body')

            data = {}  # Dictionary to store data

            if h1_tag_element:
                data['Car Name'] = h1_tag_element.find_next('h1').text.strip()
            else:
                data['Car Name'] = None

            if safety_rating_element:
                safety_rating_icons = safety_rating_element.find_next('td').find_all('i')
                data['Safety Rating'] = len(safety_rating_icons)
            else:
                data['Safety Rating'] = None

            if cargo_volume_element:
                data['Cargo Volume'] = cargo_volume_element.find_next('td').text.strip()
            else:
                data['Cargo Volume'] = None

            if turning_element:
```

```
                data['Turning Radius'] = turning_element.find_next('td').text.strip()
            else:
                data['Turning Radius'] = None

            if home_charge_speed_element:
                data['Home Charge Speed'] = home_charge_speed_element.find_next('td').text.strip()
            else:
                data['Home Charge Speed'] = None

            if total_power_element:
                data['Horsepower'] = total_power_element.find_next('td').text.strip()
            elif total_power_element_star:
                data['Horsepower'] = total_power_element_star.find_next('td').text.strip()
            else:
                data['Horsepower'] = None

            if car_body_element:
                data['Vehicle Segment'] = car_body_element.find_next('td').text.strip()
            else:
                data['Vehicle Segment'] = None

            df_list.append(pd.DataFrame([data]))  # Append DataFrame to the list
            processed_links_8.add(link) # adding the link to the processed link set
            time.sleep(40) #adding a delay between requests.
            count += 1

            if count==links_limit: #count the number of links processed and check if there are 30
                break
        else:
            print(f"Failed to extract content for link: {link}")
    except requests.exceptions.RequestException as e:
        print(f"Error with link {link}: {e}")
    except Exception as e:
        print(f"Error processing link {link}: {e}")

# Concatenate all DataFrames in df_list
df_9 = pd.concat(df_list, ignore_index=True)

df_9
```

In [ ]:

```
df_list = []  # List to store individual DataFrames
processed_links_9 = set()  # Creating an empty set to keep track of processed links
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
count = 0
links_limit = 40

for link in my_list:
    try:
        if link in processed_links or link in processed_links_1 or link in processed_links_2 or link in process
            print(f"Link {link} already processed. Skipping...")
            continue

        response = requests.get(link,headers=headers,timeout=10)#sets a timer for HTTPS requests
        if response.status_code == 200: #if request is sucessfull
            page_content = response.content
            inner_soup = BeautifulSoup(page_content, 'lxml')
            inner_box = inner_soup.find('div', class_='content')

            h1_tag_element = inner_box.find('header', class_='sub-header')
            safety_rating_element = inner_box.find('td', string='Safety Rating')
            cargo_volume_element = inner_box.find('td', string='Cargo Volume')
            turning_element = inner_box.find('td', string='Turning Circle')
            home_charge_speed_element = inner_box.find('td', string='Charge Speed')
            total_power_element = inner_box.find('td', string='Total Power')
            total_power_element_star = inner_box.find('td', string='Total Power *')
            car_body_element = inner_box.find('td', string='Car Body')

            data = {}  # Dictionary to store data

            if h1_tag_element:
                data['Car Name'] = h1_tag_element.find_next('h1').text.strip()
            else:
                data['Car Name'] = None

            if safety_rating_element:
                safety_rating_icons = safety_rating_element.find_next('td').find_all('i')
                data['Safety Rating'] = len(safety_rating_icons)
            else:
                data['Safety Rating'] = None

            if cargo_volume_element:
                data['Cargo Volume'] = cargo_volume_element.find_next('td').text.strip()
            else:
                data['Cargo Volume'] = None

            if turning_element:
                data['Turning Radius'] = turning_element.find_next('td').text.strip()
            else:
```

```
                data['Turning Radius'] = None

            if home_charge_speed_element:
                data['Home Charge Speed'] = home_charge_speed_element.find_next('td').text.strip()
            else:
                data['Home Charge Speed'] = None

            if total_power_element:
                data['Horsepower'] = total_power_element.find_next('td').text.strip()
            elif total_power_element_star:
                data['Horsepower'] = total_power_element_star.find_next('td').text.strip()
            else:
                data['Horsepower'] = None

            if car_body_element:
                data['Vehicle Segment'] = car_body_element.find_next('td').text.strip()
            else:
                data['Vehicle Segment'] = None

            df_list.append(pd.DataFrame([data]))  # Append DataFrame to the list
            processed_links_9.add(link) # adding the link to the processed link set
            time.sleep(40) #adding a delay between requests.
            count += 1

            if count==links_limit: #count the number of links processed and check if there are 30
                break
        else:
            print(f"Failed to extract content for link: {link}")
    except requests.exceptions.RequestException as e:
        print(f"Error with link {link}: {e}")
    except Exception as e:
        print(f"Error processing link {link}: {e}")

# Concatenate all DataFrames in df_list
df_10 = pd.concat(df_list, ignore_index=True)

df_10
```

In [ ]:

```
df_list = []  # List to store individual DataFrames
processed_links_10 = set()  # Creating an empty set to keep track of processed links
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
count = 0
links_limit = 40

for link in my_list:
    try:
        if link in processed_links or link in processed_links_1 or link in processed_links_2 or link in process
            print(f"Link {link} already processed. Skipping...")
            continue

        response = requests.get(link,headers=headers,timeout=10)#sets a timer for HTTPS requests
        if response.status_code == 200: #if request is sucessfull
            page_content = response.content
            inner_soup = BeautifulSoup(page_content, 'lxml')
            inner_box = inner_soup.find('div', class_='content')

            h1_tag_element = inner_box.find('header', class_='sub-header')
            safety_rating_element = inner_box.find('td', string='Safety Rating')
            cargo_volume_element = inner_box.find('td', string='Cargo Volume')
            turning_element = inner_box.find('td', string='Turning Circle')
            home_charge_speed_element = inner_box.find('td', string='Charge Speed')
            total_power_element = inner_box.find('td', string='Total Power')
            total_power_element_star = inner_box.find('td', string='Total Power *')
            car_body_element = inner_box.find('td', string='Car Body')

            data = {}  # Dictionary to store data

            if h1_tag_element:
                data['Car Name'] = h1_tag_element.find_next('h1').text.strip()
            else:
                data['Car Name'] = None

            if safety_rating_element:
                safety_rating_icons = safety_rating_element.find_next('td').find_all('i')
                data['Safety Rating'] = len(safety_rating_icons)
            else:
                data['Safety Rating'] = None

            if cargo_volume_element:
                data['Cargo Volume'] = cargo_volume_element.find_next('td').text.strip()
            else:
                data['Cargo Volume'] = None

            if turning_element:
                data['Turning Radius'] = turning_element.find_next('td').text.strip()
            else:
                data['Turning Radius'] = None
```

```
            if home_charge_speed_element:
                data['Home Charge Speed'] = home_charge_speed_element.find_next('td').text.strip()
            else:
                data['Home Charge Speed'] = None

            if total_power_element:
                data['Horsepower'] = total_power_element.find_next('td').text.strip()
            elif total_power_element_star:
                data['Horsepower'] = total_power_element_star.find_next('td').text.strip()
            else:
                data['Horsepower'] = None

            if car_body_element:
                data['Vehicle Segment'] = car_body_element.find_next('td').text.strip()
            else:
                data['Vehicle Segment'] = None

            df_list.append(pd.DataFrame([data]))  # Append DataFrame to the list
            processed_links_10.add(link) # adding the link to the processed link set
            time.sleep(40) #adding a delay between requests.
            count += 1

            if count==links_limit: #count the number of links processed and check if there are 30
                break
        else:
            print(f"Failed to extract content for link: {link}")
    except requests.exceptions.RequestException as e:
        print(f"Error with link {link}: {e}")
    except Exception as e:
        print(f"Error processing link {link}: {e}")

# Concatenate all DataFrames in df_list
df_11 = pd.concat(df_list, ignore_index=True)

df_11
```

In [ ]:

In [50]:
```
df_11.to_csv('df_11.csv',index=False)
```

In [10]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
pd.options.display.float_format = '{:.2f}'.format
```

In [11]:
```
result = pd.read_csv(r'To_Clean_DF.csv')
```

In [ ]:
```
# result
```

In [13]:
```
#Cleaning up columns and removing unwanted data
result['Acceleration']=result['Acceleration'].str.replace('sec', '',regex=True)
result['Top Speed']=result['Top Speed'].str.replace('km/h', '',regex=True)
result['Efficiency']=result['Efficiency'].str.replace('Wh/km', '',regex=True)
result['Fast Charge Speed']=result['Fast Charge Speed'].str.replace('km/h', '',regex=True)
result['Fast Charge Speed']=result['Fast Charge Speed'].str.replace('-','0',regex=True)
result['Country UK Price']=result['Country UK Price'].str.replace('*','',regex=True)
result['Country NL Price']=result['Country NL Price'].str.replace('*','',regex=True)
result['Country DE Price']=result['Country DE Price'].str.replace('*','',regex=True)
result['Real Range']=result['Real Range'].str.replace('km', '',regex=True)
result['Cargo Volume']=result['Cargo Volume'].str.replace('L', '',regex=True)
result['Turning Radius']=result['Turning Radius'].str.replace('m', '',regex=True)
result['Home Charge Speed']=result['Home Charge Speed'].str.replace('km/h', '',regex=True)
result['HP'] = result['Horsepower'].apply(lambda x: re.findall('\((.*?)PS\)',x)) #keeping only the value
result['HP'] = result['HP'].apply(lambda x: x[0]).str.strip('[]') #removing brackets[]
result = result.drop(['Horsepower'],axis=1)
result = result.replace(r'No Data', np.nan, regex=True)
```

In [14]:
```
#Performing calculations to convert from Euro to Pounds.
def calculate_price(row):
    if pd.isna(row['Country UK Price']):
        if pd.notna(row['Country DE Price']):
            return row['Country DE Price'] / 1.15
        elif pd.notna(row['Country NL Price']):
            return row['Country NL Price'] / 1.15
    return row['Country UK Price']
```

In [15]:
```
result["Country UK Price"] = [float(str(i).replace(",", "")) for i in result["Country UK Price"]]
result["Country DE Price"] = [float(str(i).replace(",", "")) for i in result["Country DE Price"]]
result["Country NL Price"] = [float(str(i).replace(",", "")) for i in result["Country NL Price"]]
```

In [16]:
```
#Setting columns to Int or float. Also calculating the UK price with conversion of NL and DE Euro conversion ra
result[['Acceleration','Turning Radius','Cargo Volume','Home Charge Speed','Country UK Price','Country DE Price
result[['Top Speed','Efficiency','Fast Charge Speed','Real Range','HP']] = result[['Top Speed','Efficiency','Fa
result['Uk Price']=result.apply(calculate_price,axis=1)
result=result.drop(['Country UK Price','Country DE Price','Country NL Price'], axis=1)
```

In [17]:
```
result.dtypes
```

Out[17]:
```
Brand                 object
Model                 object
Battery Size         float64
Towing Capacity      float64
Number of Seats        int64
Acceleration         float64
Top Speed              int32
Efficiency             int32
Fast Charge Speed      int32
Real Range             int32
Drive Type            object
Safety Rating        float64
Cargo Volume         float64
Turning Radius       float64
Home Charge Speed    float64
Vehicle Segment       object
HP                     int32
Uk Price             float64
dtype: object
```

In [18]:
```
result.isna().sum()
```

Out[18]:
```
Brand                  0
Model                  0
Battery Size           0
Towing Capacity      120
Number of Seats        0
Acceleration           0
Top Speed              0
Efficiency             0
Fast Charge Speed      0
Real Range             0
Drive Type             0
Safety Rating        200
Cargo Volume          11
Turning Radius        72
Home Charge Speed      3
Vehicle Segment        0
HP                     0
Uk Price               0
dtype: int64
```

In [44]:
```
#filling and handling NA values. Setting the mean for Home charge speed and Cargo Volume. Setting value 0 to ro
result['Home Charge Speed'].fillna(float(result['Home Charge Speed'].mean()), inplace=True)
result['Cargo Volume'].fillna(float(result['Cargo Volume'].mean()), inplace=True)
result['Safety Rating'] = result['Safety Rating'].fillna(0)
```

In [45]:
```
result.isna().sum()
```

Out[45]:
```
Brand                  0
Model                  0
Battery Size           0
Towing Capacity      120
Number of Seats        0
Acceleration           0
Top Speed              0
Efficiency             0
Fast Charge Speed      0
Real Range             0
Drive Type             0
Safety Rating          0
Cargo Volume           0
Turning Radius        72
Home Charge Speed      0
Vehicle Segment        0
HP                     0
Uk Price               0
dtype: int64
```

In [46]:
```
#Performing One hot encoding for the safety rating column
one_hot = pd.get_dummies(result['Safety Rating'])
one_hot.rename(columns = {0.0:'Safety_Rating_No', 5.0:'Safety_Rating_Yes'}, inplace = True)

#Joining the two column
result=pd.concat([result, one_hot], axis=1)
result=result.drop(['Safety Rating'], axis=1)
```

In [47]:
```
#Performing One hot encoding for the Drive Type column & joining the two columns
one_hot_drive_type = pd.get_dummies(result['Drive Type'])
result=pd.concat([result, one_hot_drive_type], axis=1)
result=result.drop(['Drive Type'], axis=1)
```

In [48]:
```
#Performing One hot encoding for the Vehicle Segment column & joining the two columns
one_hot_vehicle_segment = pd.get_dummies(result['Vehicle Segment'])
result=pd.concat([result, one_hot_vehicle_segment], axis=1)
result=result.drop(['Vehicle Segment'], axis=1)
```

In [49]:
```
result = result[['Brand',
 'Model',
 'Battery Size',
 'Towing Capacity',
 'Number of Seats',
 'Acceleration',
 'Top Speed',
 'Efficiency',
 'Fast Charge Speed',
 'Real Range',
 'All Wheel Drive',
 'Forward Wheel Drive',
 'Rear Wheel Drive',
 'Safety_Rating_No',
 'Safety_Rating_Yes',
 'Cargo Volume',
 'Turning Radius',
 'Home Charge Speed',
 'HP',
 'Cabriolet',
 'Coupe',
 'Hatchback',
```

```
 'Liftback Sedan',
 'SUV',
 'Sedan',
 'Small Passenger Van',
 'Station/Estate',
 'Uk Price'
]]
```

In [1]:
```
# result.dtypes
```

In [51]:
```
result_copy = result.copy()
result_copy_2 = result_copy.drop(result_copy.iloc[:, 19:27],axis = 1)
```

In [ ]:
```
#result_copy = result.copy()
imputation_df=result_copy_2.iloc[:, 2:]
imputation_df
```

In [ ]:
```
imputation_df.isna().sum()
```

Performing Imputations for towing capacity and turning radius

In [54]:
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import LinearRegression
```

In [55]:
```
lr= LinearRegression()
imp = IterativeImputer(estimator=lr,max_iter=10,random_state=0)
```

In [56]:
```
imputed_values = imp.fit_transform(imputation_df)
```

In [57]:
```
#creating a new dataframe with the imputed values and using the previous column names
imputed_df = pd.DataFrame(imputed_values, columns=imputation_df.columns)

#Setting the same dataframe type as before
imputed_df = imputed_df.astype(imputation_df.dtypes)
```

In [2]:
```
# imputed_df
```

In [59]:
```
imputed_df.isna().sum()
```

Out[59]:
```
Battery Size           0
Towing Capacity        0
Number of Seats        0
Acceleration           0
Top Speed              0
Efficiency             0
Fast Charge Speed      0
Real Range             0
All Wheel Drive        0
Forward Wheel Drive    0
Rear Wheel Drive       0
Safety_Rating_No       0
Safety_Rating_Yes      0
Cargo Volume           0
Turning Radius         0
Home Charge Speed      0
HP                     0
Uk Price               0
dtype: int64
```

In [60]:
```
#getting the Segment types from the result dataframe and attaching to the imputed df
extracted_col = result[['Cabriolet',
 'Coupe',
 'Hatchback',
 'Liftback Sedan',
 'SUV',
 'Sedan',
 'Small Passenger Van',
 'Station/Estate']]
imputed_df = imputed_df.join(extracted_col)
```

In [61]:
```
#Renaming the columns
imputed_df.rename(columns = {'Battery Size':'Battery_Size', 'Towing Capacity':'Towing_Capacity',
                             'Number of Seats':'Number_of_Seats','Top Speed':'Top_Speed','Fast Charge Speed':'
```

In [62]:
```
#Moving UK price to the last position
column_to_move = 'UK_Price'
new_position = 25

# Get the list of columns
columns = imputed_df.columns.tolist()

# Remove the column from the list
```

```
columns.remove(column_to_move)

# Insert the column at the new position
columns.insert(new_position, column_to_move)

# Reorder the columns in the DataFrame
imputed_df = imputed_df[columns]
```

In [3]:
```
# imputed_df
```

In [ ]:
```
imputed_df[imputed_df < 0].count()
```

In [ ]:
```
unique_rows = imputed_df.drop_duplicates()
unique_rows
```

In [66]:
```
# unique_rows = result.drop_duplicates()
# unique_rows
# imputation_df.describe()
# imputed_df.isna().sum()
#result[result['Towing Capacity'].isnull()]
# cleaning_1 & cleaning_2 files were shared with the professor for feedback
```

In [67]:
```
imputed_df.to_csv('Final_Clean.csv',index=False)
```

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
pd.options.display.float_format = '{:.2f}'.format
```

In [2]:
```
original_df = pd.read_csv(r'Final_Clean.csv')
visual_df = original_df.copy()
```

In [3]:
```
visual_df.describe()
```

Out[3]:

	Battery_Size	Towing_Capacity	Number_of_Seats	Acceleration	Top_Speed	Efficiency	Fast_Charge_Speed	Real_Range	All_Wheel_Dri
count	367.00	367.00	367.00	367.00	367.00	367.00	367.00	367.00	367.
mean	71.16	1164.60	5.27	7.30	180.22	194.88	548.69	370.10	0.
std	20.10	496.57	0.90	2.96	35.62	31.82	235.41	105.80	0.
min	21.30	0.00	4.00	2.10	125.00	137.00	0.00	135.00	0.
25%	58.00	750.00	5.00	5.00	158.50	171.00	365.00	300.00	0.
50%	71.00	1000.00	5.00	6.90	180.00	187.00	520.00	380.00	0.
75%	83.70	1514.08	5.00	8.85	200.00	208.00	675.00	440.00	1.
max	123.00	2500.00	7.00	19.10	320.00	295.00	1290.00	685.00	1.

8 rows × 26 columns

In [4]:
```
visual_df.corr()
# represents total correlation. 0 no correlation. -ve when one goes up the other goes down. +ve when one goes u
```

Out[4]:

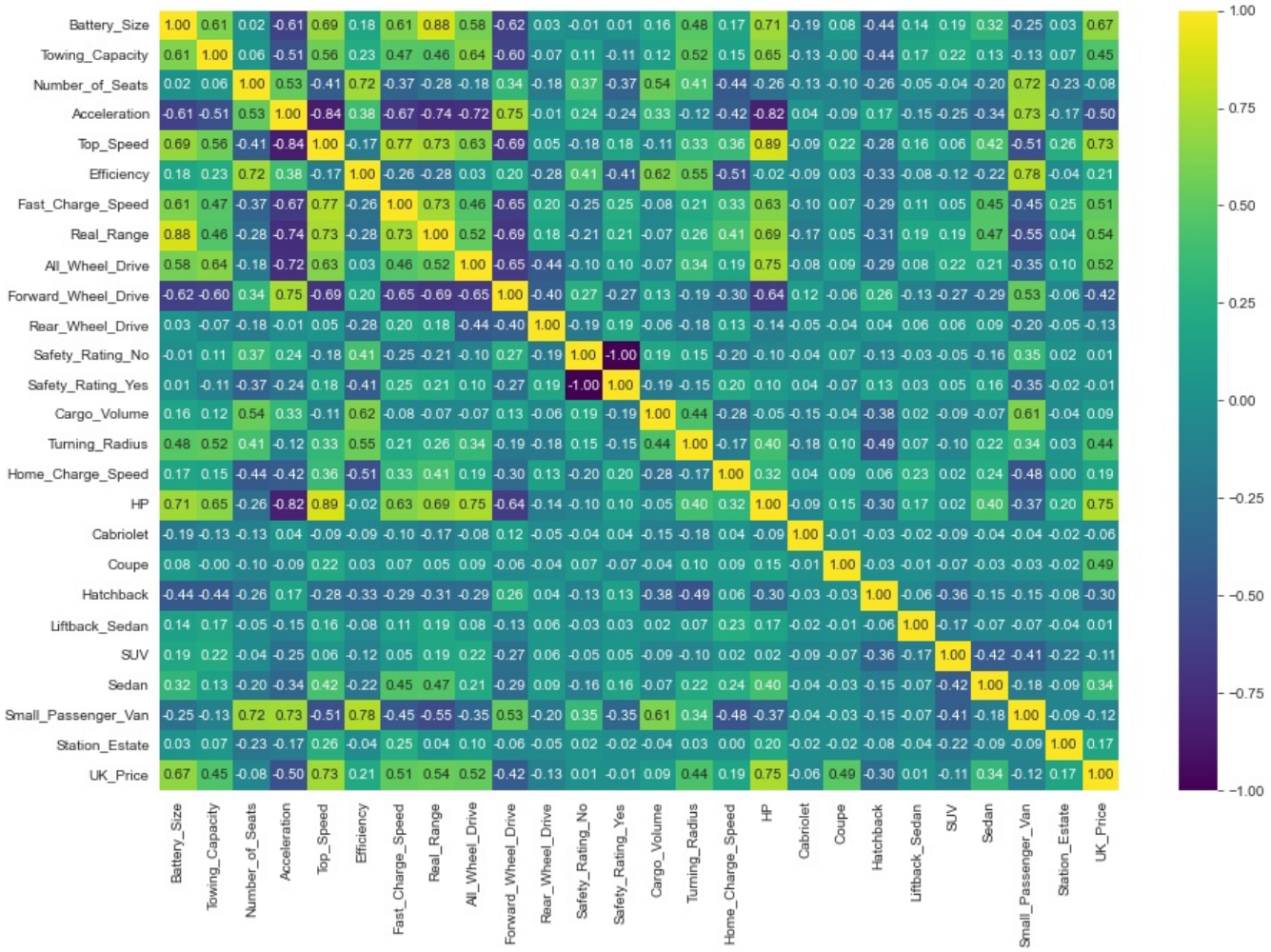
	Battery_Size	Towing_Capacity	Number_of_Seats	Acceleration	Top_Speed	Efficiency	Fast_Charge_Speed	Real_Range
Battery_Size	1.00	0.61	0.02	-0.61	0.69	0.18	0.61	0.88
Towing_Capacity	0.61	1.00	0.06	-0.51	0.56	0.23	0.47	0.46
Number_of_Seats	0.02	0.06	1.00	0.53	-0.41	0.72	-0.37	-0.28
Acceleration	-0.61	-0.51	0.53	1.00	-0.84	0.38	-0.67	-0.74
Top_Speed	0.69	0.56	-0.41	-0.84	1.00	-0.17	0.77	0.73
Efficiency	0.18	0.23	0.72	0.38	-0.17	1.00	-0.26	-0.28
Fast_Charge_Speed	0.61	0.47	-0.37	-0.67	0.77	-0.26	1.00	0.73
Real_Range	0.88	0.46	-0.28	-0.74	0.73	-0.28	0.73	1.00
All_Wheel_Drive	0.58	0.64	-0.18	-0.72	0.63	0.03	0.46	0.52
Forward_Wheel_Drive	-0.62	-0.60	0.34	0.75	-0.69	0.20	-0.65	-0.69
Rear_Wheel_Drive	0.03	-0.07	-0.18	-0.01	0.05	-0.28	0.20	0.18
Safety_Rating_No	-0.01	0.11	0.37	0.24	-0.18	0.41	-0.25	-0.2
Safety_Rating_Yes	0.01	-0.11	-0.37	-0.24	0.18	-0.41	0.25	0.2
Cargo_Volume	0.16	0.12	0.54	0.33	-0.11	0.62	-0.08	-0.07
Turning_Radius	0.48	0.52	0.41	-0.12	0.33	0.55	0.21	0.26
Home_Charge_Speed	0.17	0.15	-0.44	-0.42	0.36	-0.51	0.33	0.4
HP	0.71	0.65	-0.26	-0.82	0.89	-0.02	0.63	0.69
Cabriolet	-0.19	-0.13	-0.13	0.04	-0.09	-0.09	-0.10	-0.17
Coupe	0.08	-0.00	-0.10	-0.09	0.22	0.03	0.07	0.05
Hatchback	-0.44	-0.44	-0.26	0.17	-0.28	-0.33	-0.29	-0.3
Liftback_Sedan	0.14	0.17	-0.05	-0.15	0.16	-0.08	0.11	0.19
SUV	0.19	0.22	-0.04	-0.25	0.06	-0.12	0.05	0.19
Sedan	0.32	0.13	-0.20	-0.34	0.42	-0.22	0.45	0.47
Small_Passenger_Van	-0.25	-0.13	0.72	0.73	-0.51	0.78	-0.45	-0.55
Station_Estate	0.03	0.07	-0.23	-0.17	0.26	-0.04	0.25	0.04
UK_Price	0.67	0.45	-0.08	-0.50	0.73	0.21	0.51	0.54

26 rows × 26 columns

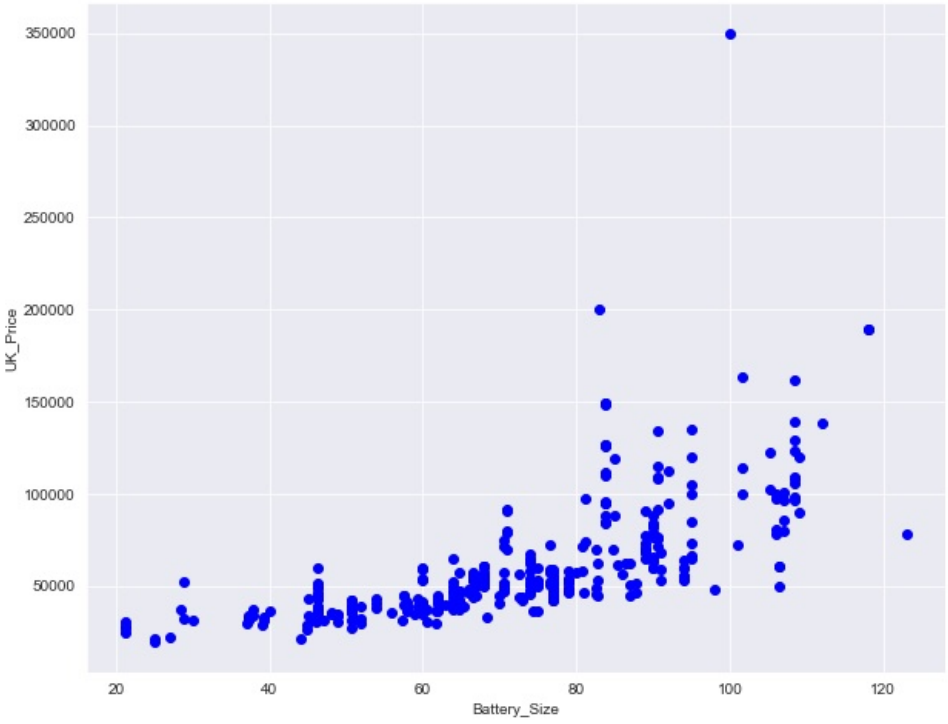
In [5]:
```
# plt.figure(figsize=(10,8))
# sb.heatmap(visual_df.corr())
# #top speed and battery size are strongly correlated.
```

In [6]:
```
plt.figure(figsize=(15,10))
sb.heatmap(visual_df.corr(),annot=True,cmap="viridis",fmt="0.2f")
# corr goes from-1 to 1. 1 is perfectly correlated. -1 is one column goes up the other goes down
# check the column of the price
# check dark colors. the strongest correlations. These strong correlations can indicate variables that have a s
#top speed, battery size,fast charge,real range,all wheel drive,acceleration,fwd are of interest - These might
```

Out[6]:
```
<AxesSubplot:>
```



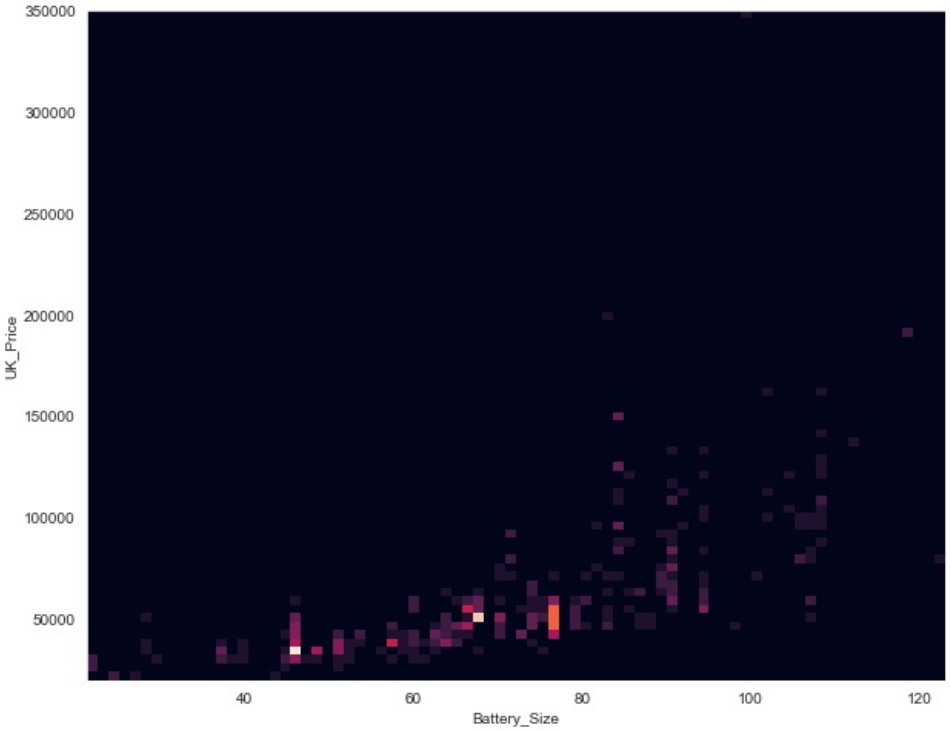
In [7]:
```
plt.figure(figsize=(10,8))
plt.scatter(x="Battery_Size",y="UK_Price",data=visual_df,c='blue')
plt.xlabel("Battery_Size")
plt.ylabel("UK_Price")
plt.show();
```



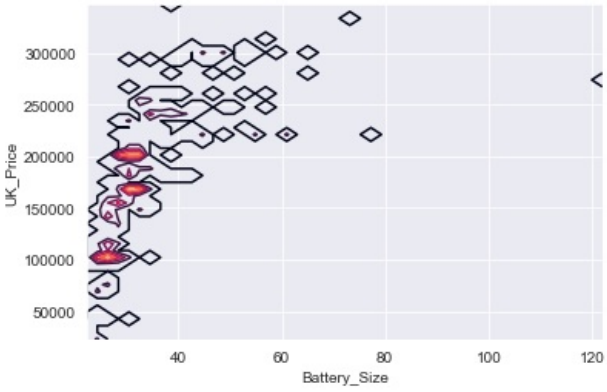
In [8]:
```
plt.figure(figsize=(10,8))
```

```
plt.hist2d(visual_df["Battery_Size"],visual_df["UK_Price"],bins=80)
plt.xlabel("Battery_Size")
plt.ylabel("UK_Price");
#positive correlation between battery size and real range
```

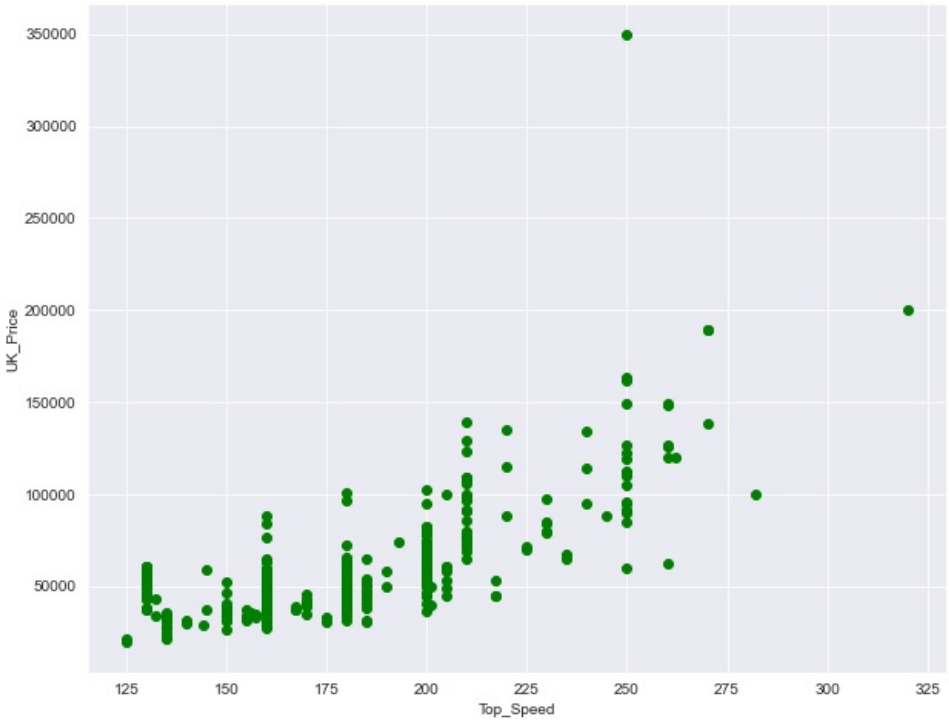
```
C:\Users\reece\AppData\Local\Temp\ipykernel_17356\1880486323.py:2: MatplotlibDeprecationWarning: Auto-removal o
f grids by pcolor() and pcolormesh() are deprecated since 3.5 and will be removed two minor releases later; plea
se call grid(False) first.
  plt.hist2d(visual_df["Battery_Size"],visual_df["UK_Price"],bins=80)
```



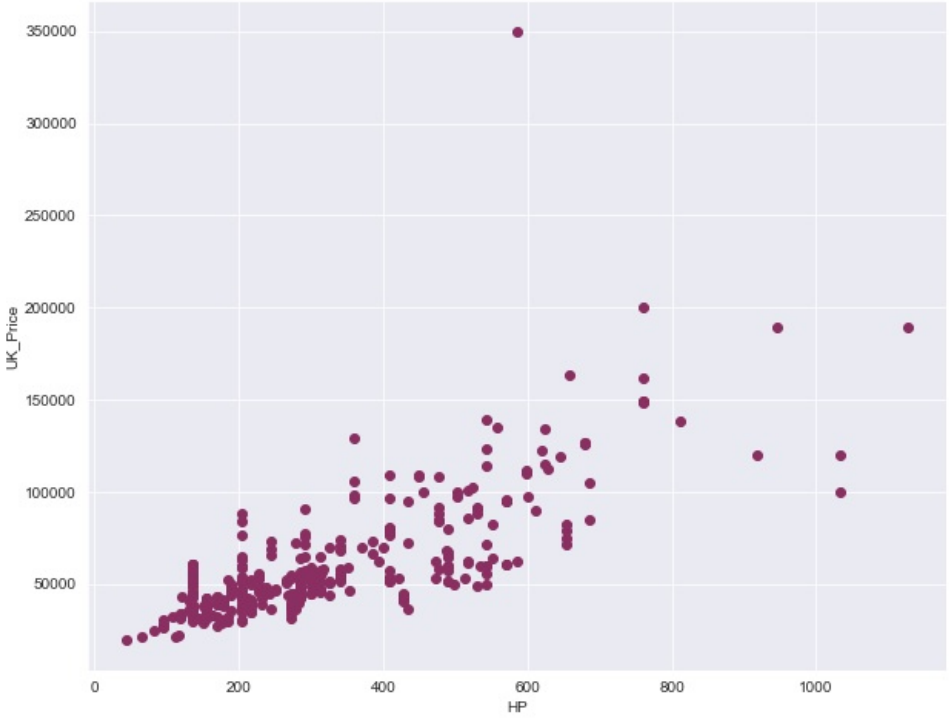
In [9]:
```
#contour plot
hist, x_edge, y_edge=np.histogram2d(visual_df["Battery_Size"],visual_df["UK_Price"],bins=50)
x_center = 0.5 * (x_edge[1:]+x_edge[:-1])
y_center = 0.5 * (y_edge[1:]+y_edge[:-1])

plt.contour(x_center,y_center,hist, levels=5)
plt.xlabel("Battery_Size")
plt.ylabel("UK_Price");
```



In [20]:
```
plt.figure(figsize=(10,8))
plt.scatter(x="Top_Speed",y="UK_Price",data=visual_df,c='green')
plt.xlabel("Top_Speed")
plt.ylabel("UK_Price")
plt.show();
```



In [11]:
```
#Scatter plot
plt.figure(figsize=(10,8))
plt.scatter(x=visual_df.loc[:,"HP"],y=visual_df.loc[:,"UK_Price"],c="#8b2e62")
plt.xlabel("HP")
plt.ylabel("UK_Price");
```



In [12]:
```
# #KDE PLOT
# plt.figure(figsize=(10,8))
# sb.kdeplot(x=visual_df["Battery_Size"],y=visual_df["Real_Range"],shade=True);
# plt.hist2d(x=visual_df["Battery_Size"],y=visual_df["Real_Range"],bins=20,alpha=0.3);
# #peak of our distribution is around battery size = 72 & range of 400.
```

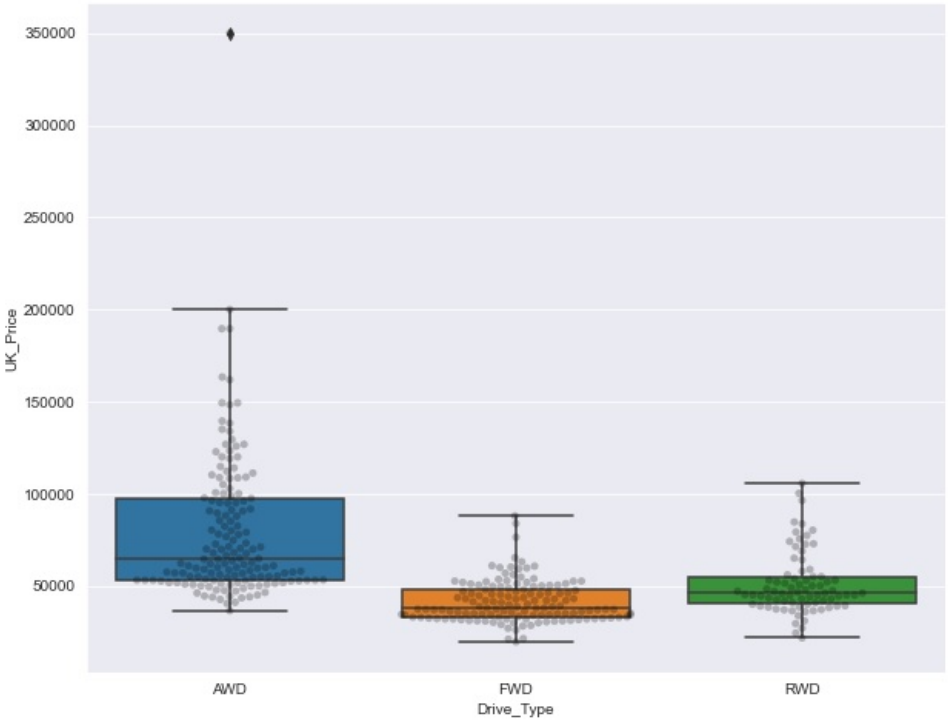
In [13]:
```
def map_drive_type(row):
    if row['All_Wheel_Drive'] == 1:
        return "AWD"
    elif row['Rear_Wheel_Drive'] == 1:
        return "RWD"
    elif row['Forward_Wheel_Drive'] == 1:
        return "FWD"
    else:
        return None

visual_df['Drive_Type'] = visual_df.apply(map_drive_type, axis=1)
```

In [14]:
```
#Bee swarm for categorical data with box plot on it
plt.figure(figsize=(10,8))
```

```
sb.boxplot(x= "Drive_Type", y= "UK_Price", data= visual_df,whis=4.0)
sb.swarmplot(x= "Drive_Type", y= "UK_Price", data= visual_df,color="k",alpha=0.25)
#ALL WHEEL has slightly higher prices then the rest. Looks like all categories show just one peak.
# 25%of the data for each section in box plot. Data is tightly grouped then outliers won't be too far away. Fir
```

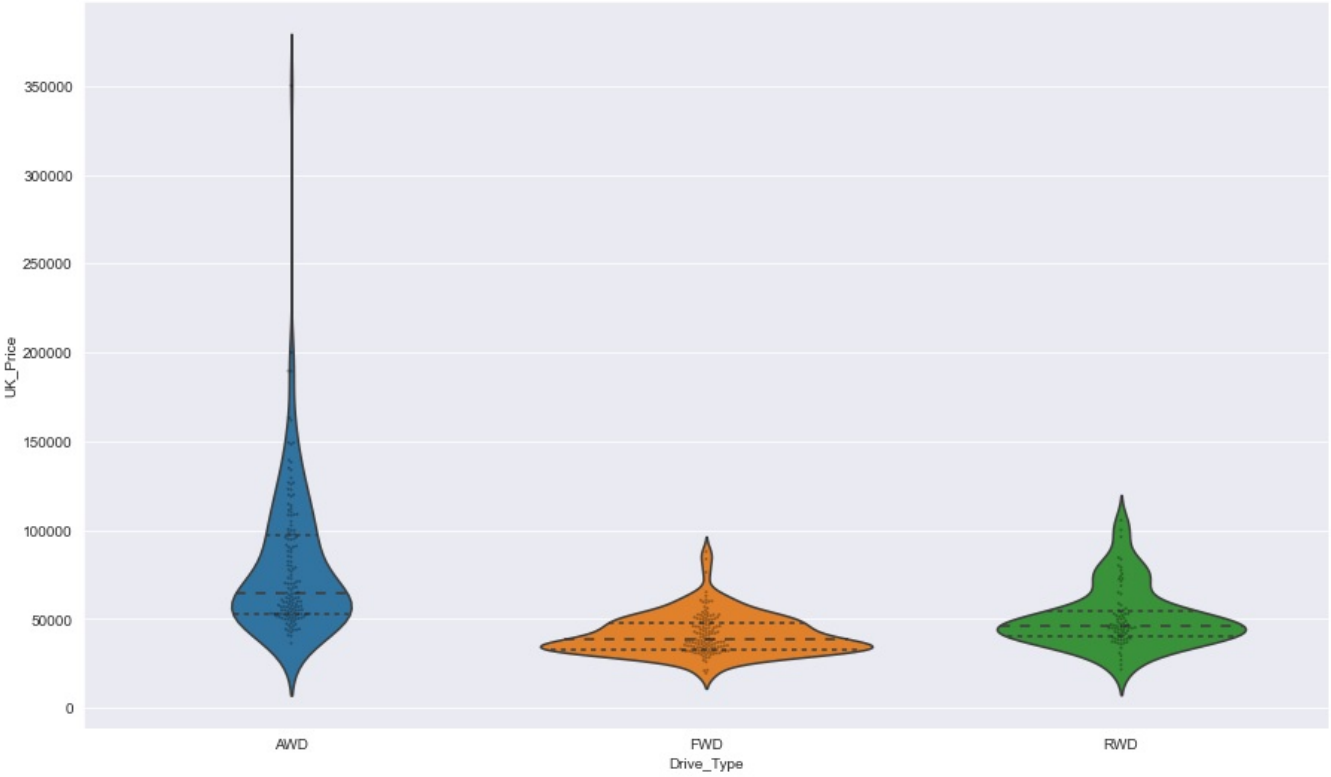
Out[14]:

```
<AxesSubplot:xlabel='Drive_Type', ylabel='UK_Price'>
```



In [15]:

```
#Violin Plot with box plots quartiles. 25,50 and 75%
plt.figure(figsize=(15,9))
sb.violinplot(x= "Drive_Type", y= "UK_Price", data= visual_df,inner="quartile");
sb.swarmplot(x= "Drive_Type", y= "UK_Price", data= visual_df,color="k",alpha=0.3,size=2);
```

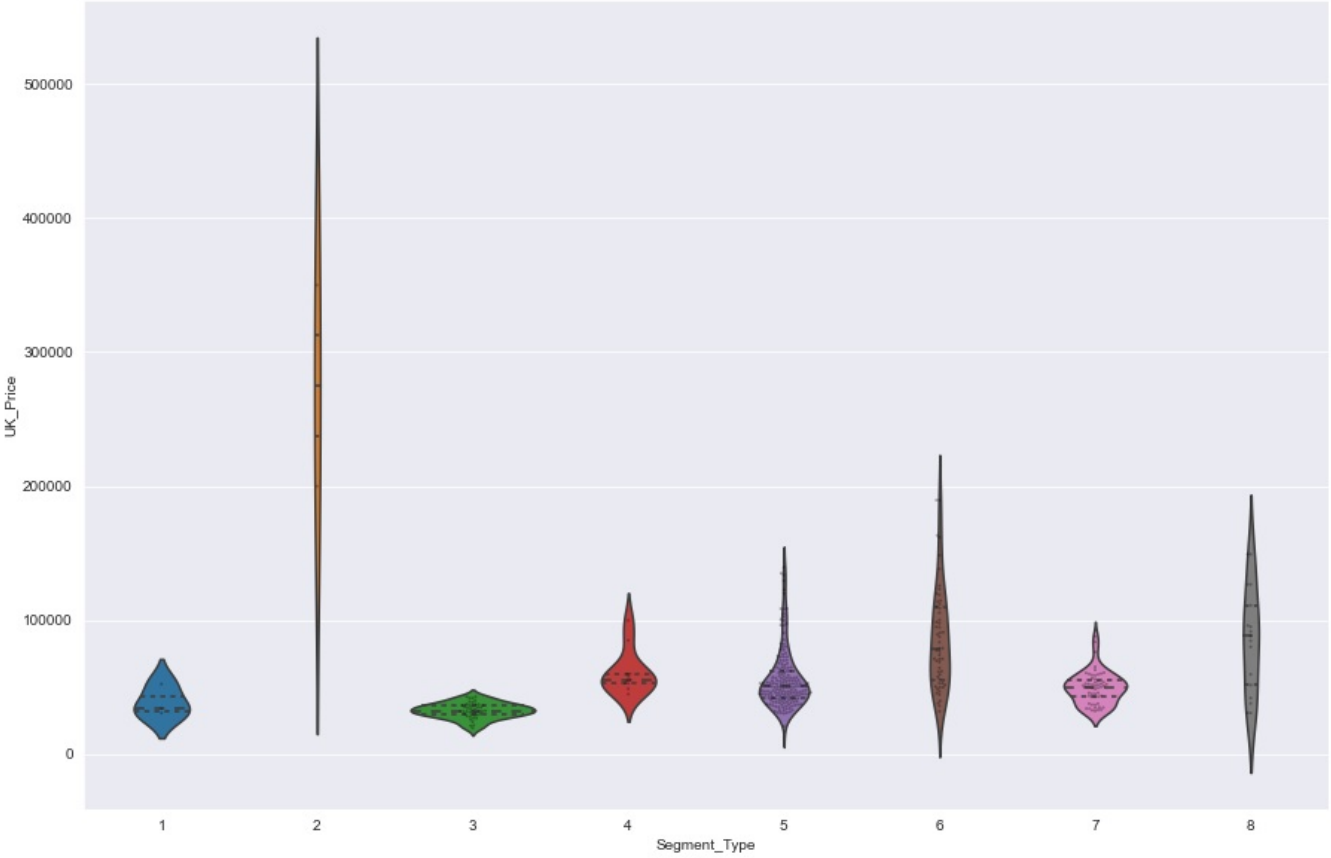


In [16]:

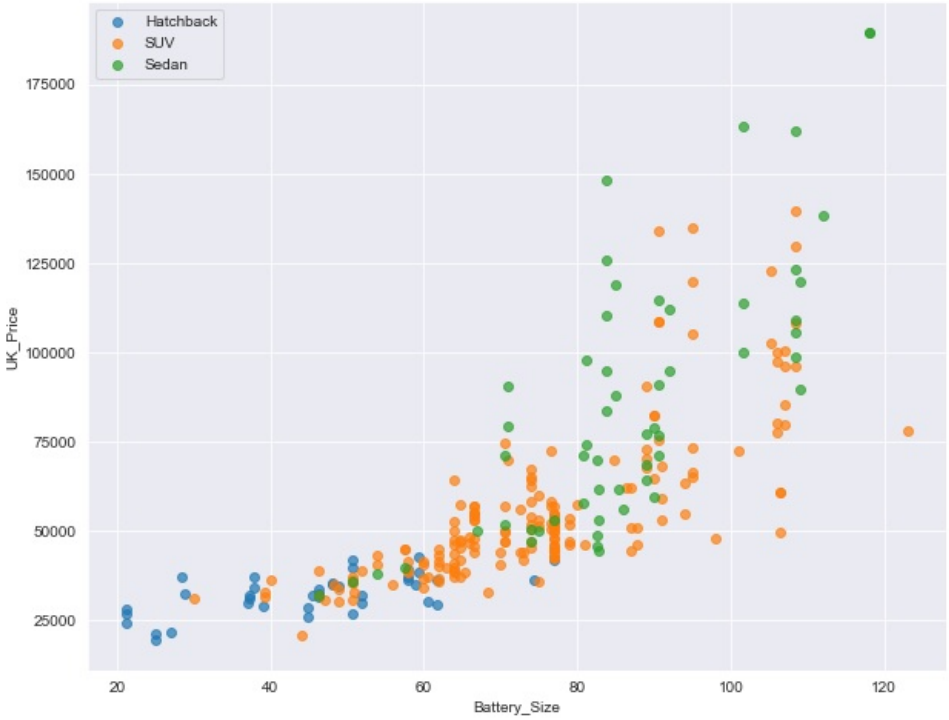
```
def segment_type(row):
    if row['Cabriolet'] == 1:
        return 1
    elif row['Coupe'] == 1:
        return 2
    elif row['Hatchback'] == 1:
        return 3
    elif row['Liftback_Sedan'] == 1:
        return 4
    elif row['SUV'] == 1:
        return 5
    elif row['Sedan'] == 1:
        return 6
```

```
    elif row['Small_Passenger_Van'] == 1:
        return 7
    elif row['Station_Estate'] == 1:
        return 8
    else:
        return None

visual_df['Segment_Type'] = visual_df.apply(segment_type, axis=1)
```

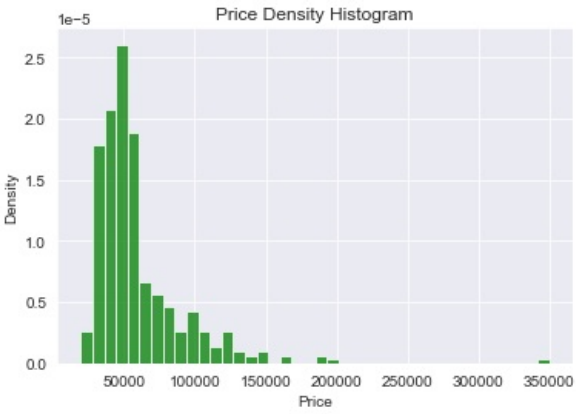
In [17]:
```
#Violin Plot with box plots quartiles. 25,50 and 75%
plt.figure(figsize=(15,10))
sb.violinplot(x= "Segment_Type", y= "UK_Price", data= visual_df,inner="quartile");
sb.swarmplot(x= "Segment_Type", y= "UK_Price", data= visual_df,color="k",alpha=0.3,size=2);
```



In [18]:
```
#Segmenting the vehicle type vs price
plt.figure(figsize=(10,8))
hatchback_segment = visual_df[visual_df["Segment_Type"]==3]
sedan_segment = visual_df[visual_df["Segment_Type"]==6]
suv_segment = visual_df[visual_df["Segment_Type"]==5]
plt.scatter(hatchback_segment['Battery_Size'], hatchback_segment['UK_Price'], label='Hatchback', alpha=0.7)
plt.scatter(suv_segment['Battery_Size'], suv_segment['UK_Price'], label='SUV', alpha=0.7)
plt.scatter(sedan_segment['Battery_Size'], sedan_segment['UK_Price'], label='Sedan', alpha=0.7)
plt.xlabel('Battery_Size')
plt.ylabel('UK_Price')
plt.legend()
plt.show()
```



In [19]:
```
plt.hist(visual_df["UK_Price"], bins=40, density=True, alpha=0.75, color='green')
plt.xlabel('Price')
plt.ylabel('Density')
plt.title('Price Density Histogram')
plt.show()
```



Removing Outliers from Price

In [215…
```
outlier_df = original_df.copy()
```

In [209…
```
# #Maybe Not Use this asit removes roughly around 35 datapoints
# print("Old Shape: ", visual_df.shape)
#
# ''' Detection '''
# # IQR - Removing Outliers using Inter Quartile Range
# # Calculate the upper and lower limits
# Q1 = visual_df['UK_Price'].quantile(0.25)
# Q3 = visual_df['UK_Price'].quantile(0.75)
# IQR = Q3 - Q1
# lower = Q1 - 1.5*IQR
# upper = Q3 + 1.5*IQR
#
# # Create arrays of Boolean values indicating the outlier rows
# upper_array = np.where(visual_df['UK_Price']>=upper)[0]
# lower_array = np.where(visual_df['UK_Price']<=lower)[0]
#
# # Removing the outliers
# visual_df.drop(index=upper_array, inplace=True)
# visual_df.drop(index=lower_array, inplace=True)
#
# # Print the new shape of the DataFrame
# print("New Shape: ", visual_df.shape)
```

In [216…
```
# Using z-score to remove outliers of Price
from scipy import stats
outlier_df["z_score_price"]=stats.zscore(outlier_df['UK_Price'])
```

```
print("Old Shape: ", outlier_df.shape)
positive_limit = 3
negative_limit = -3

upper_array = np.where(outlier_df['z_score_price']>=positive_limit)[0] #gets the index location of values great
lower_array = np.where(outlier_df['z_score_price']<=negative_limit)[0] #gets the index location of values less

outlier_df.drop(index=upper_array, inplace=True)
outlier_df.drop(index=lower_array, inplace=True)

print("New Shape: ", outlier_df.shape)
```

```
Old Shape:  (367, 27)
New Shape:  (361, 27)
```

In [211…
```
#Checking for fast charge values that are 0
# visual_df.loc[visual_df['Fast_Charge_Speed'] <= 5]
```

In [217…
```
outlier_df.var(axis=0, skipna=False,numeric_only=True)
#variance measures how far each number is from the mean of all numbers, thereby providing a ways to identify ho
```

Out[217]:
```
Battery_Size               388.95
Towing_Capacity         246379.67
Number_of_Seats              0.81
Acceleration                 8.64
Top_Speed                 1147.83
Efficiency                1024.78
Fast_Charge_Speed        54981.00
Real_Range               10705.49
All_Wheel_Drive              0.24
Forward_Wheel_Drive          0.24
Rear_Wheel_Drive             0.17
Safety_Rating_No             0.25
Safety_Rating_Yes            0.25
Cargo_Volume             37062.32
Turning_Radius               0.63
Home_Charge_Speed          313.42
HP                       29611.40
Cabriolet                    0.01
Coupe                        0.00
Hatchback                    0.11
Liftback_Sedan               0.03
SUV                          0.25
Sedan                        0.12
Small_Passenger_Van          0.13
Station_Estate               0.04
UK_Price             649725383.03
z_score_price                0.60
dtype: float64
```

In [218…
```
std_dev= outlier_df.std()
std_dev
# square root of varience. about 95% of values will be within 2 standard deviations of the mean. It tells  you
```

Out[218]:
```
Battery_Size             19.72
Towing_Capacity         496.37
Number_of_Seats           0.90
Acceleration              2.94
Top_Speed                33.88
Efficiency               32.01
Fast_Charge_Speed       234.48
Real_Range              103.47
All_Wheel_Drive           0.49
Forward_Wheel_Drive       0.49
Rear_Wheel_Drive          0.41
Safety_Rating_No          0.50
Safety_Rating_Yes         0.50
Cargo_Volume            192.52
Turning_Radius            0.79
Home_Charge_Speed        17.70
HP                      172.08
Cabriolet                 0.09
Coupe                     0.00
Hatchback                 0.32
Liftback_Sedan            0.17
SUV                       0.50
Sedan                     0.35
Small_Passenger_Van       0.36
Station_Estate            0.21
UK_Price              25489.71
z_score_price             0.78
dtype: float64
```

In [220…
```
#Covarience - Deviations of data point a from the mean and multiply the datapoint of column b how far its from
covariance = outlier_df.cov()
covariance
```

Out[220]:

	Battery_Size	Towing_Capacity	Number_of_Seats	Acceleration	Top_Speed	Efficiency	Fast_Charge_Speed	Real_Rang
Battery_Size	388.95	6046.89	0.68	-34.35	457.07	119.23	2799.44	1782.7
Towing_Capacity	6046.89	246379.67	27.67	-744.83	9616.00	3719.03	54699.89	23834.2
Number_of_Seats	0.68	27.67	0.81	1.39	-12.26	20.97	-76.28	-25.3
Acceleration	-34.35	-744.83	1.39	8.64	-83.62	36.71	-456.73	-222.9
Top_Speed	457.07	9616.00	-12.26	-83.62	1147.83	-191.57	6124.34	2565.3
Efficiency	119.23	3719.03	20.97	36.71	-191.57	1024.78	-1978.47	-952.5
Fast_Charge_Speed	2799.44	54699.89	-76.28	-456.73	6124.34	-1978.47	54981.00	17693.1
Real_Range	1782.77	23834.24	-25.37	-222.93	2565.35	-952.59	17693.18	10705.4
All_Wheel_Drive	5.56	156.94	-0.08	-1.04	10.43	0.53	52.44	25.9
Forward_Wheel_Drive	-5.96	-144.18	0.14	1.06	-11.52	3.16	-73.42	-34.5
Rear_Wheel_Drive	0.40	-12.76	-0.07	-0.03	1.08	-3.69	20.98	8.6
Safety_Rating_No	-0.01	29.21	0.17	0.35	-3.16	6.49	-29.28	-10.5
Safety_Rating_Yes	0.01	-29.21	-0.17	-0.35	3.16	-6.49	29.28	10.5
Cargo_Volume	636.97	11885.21	94.31	186.93	-655.14	3816.70	-3514.04	-1471.5
Turning_Radius	7.39	202.24	0.31	-0.23	8.35	14.27	37.32	19.3
Home_Charge_Speed	41.74	1153.56	-7.04	-20.39	177.91	-301.61	1237.99	680.0
HP	2370.42	58123.33	-39.16	-416.35	5138.54	-56.48	25504.95	11877.1
Cabriolet	-0.35	-5.76	-0.01	0.01	-0.28	-0.25	-2.20	-1.5
Coupe	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0
Hatchback	-2.86	-71.87	-0.08	0.16	-3.04	-3.45	-22.07	-10.4
Liftback_Sedan	0.52	15.08	-0.01	-0.08	1.02	-0.45	4.53	3.6
SUV	2.22	57.80	-0.02	-0.41	1.77	-1.93	7.84	11.6
Sedan	2.02	21.05	-0.07	-0.33	4.82	-2.55	36.81	15.9
Small_Passenger_Van	-1.72	-23.59	0.23	0.77	-6.34	8.91	-37.60	-20.3
Station_Estate	0.17	7.30	-0.04	-0.11	2.06	-0.28	12.70	1.0
UK_Price	366586.46	7192377.17	-520.95	-39473.67	643411.76	207981.00	3498436.95	1505379.2
z_score_price	11.17	219.16	-0.02	-1.20	19.61	6.34	106.60	45.8

27 rows × 27 columns

In [232…

```
#checking for skewness of selected columns and taking the log of price
# Taking the log can help to make the distribution more symmetric and easier to work with. Stabalises varience,
skewness_df = outlier_df.copy()
skewness_df['natural_log_price'] = np.log(skewness_df['UK_Price'])
skewness_df=skewness_df.drop(['All_Wheel_Drive','Forward_Wheel_Drive','Rear_Wheel_Drive','z_score_price','Safet
```

In [234…

```
skewness = skewness_df.skew()
skewness
#Use to understand normality of a dataset. is a measure of the asymmetry of the probability distribution of a r
```

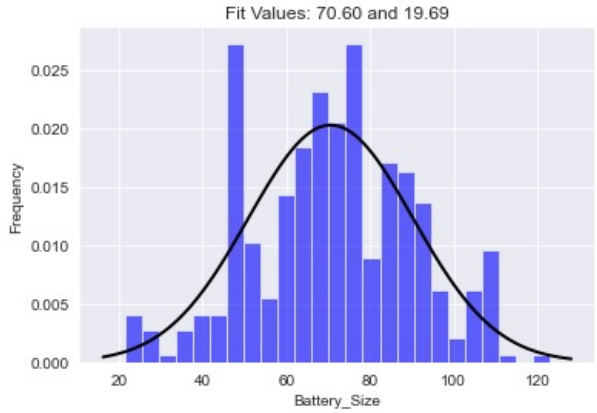
Out[234]:

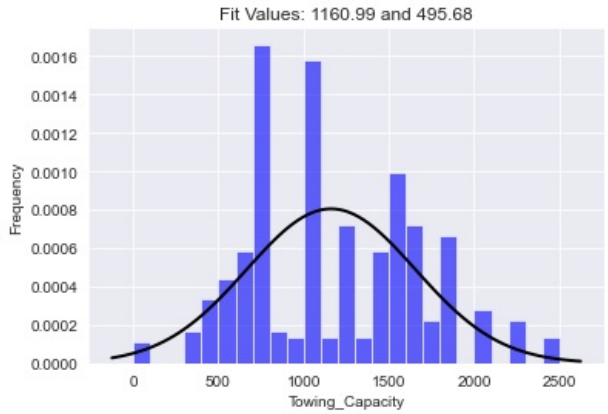
```
Battery_Size        -0.08
Towing_Capacity      0.33
Acceleration         0.72
Top_Speed            0.57
Efficiency           1.09
Fast_Charge_Speed    0.66
Real_Range          -0.17
Cargo_Volume         1.82
Turning_Radius      -0.45
Home_Charge_Speed    1.55
HP                   1.18
natural_log_price    0.52
dtype: float64
```

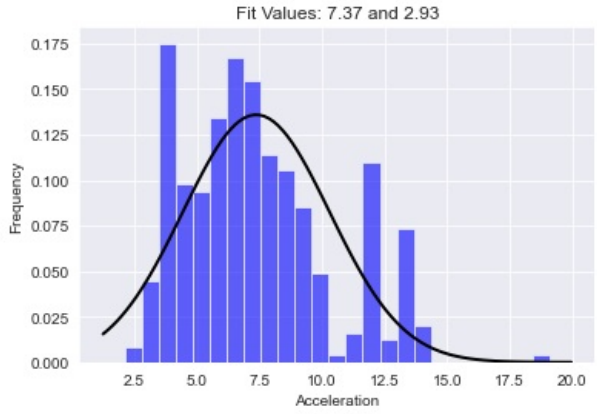
In [235…

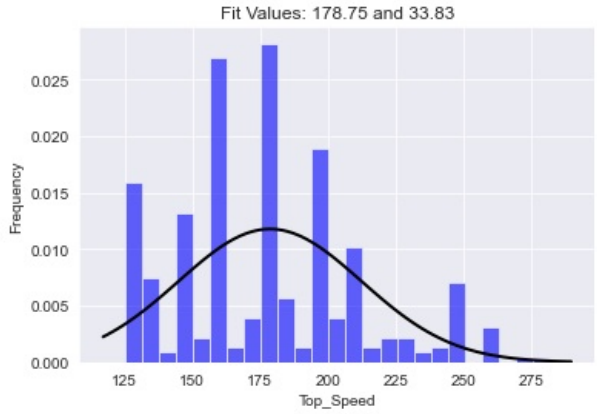
```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

for column in skewness_df.columns:#access each column seperately
    mu, std = norm.fit(skewness_df[column])

    # Plot the histogram.
    plt.hist(skewness_df[column], bins=25, density=True, alpha=0.6, color='b')

    # Plot the PDF.
    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, 100)
    p = norm.pdf(x, mu, std)
```

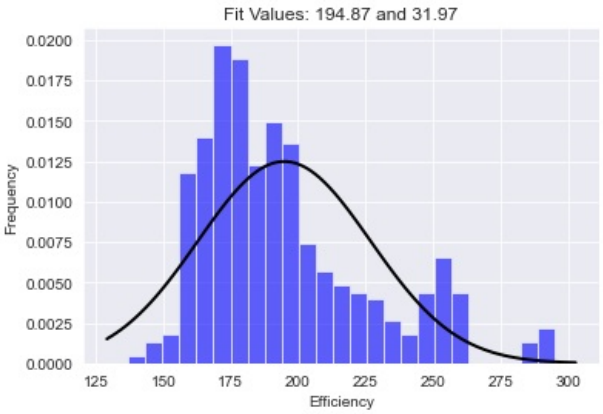
```
    plt.plot(x, p, 'k', linewidth=2)
    title = "Fit Values: {:.2f} and {:.2f}".format(mu, std)
    plt.title(title)
    plt.xlabel(column)
    plt.ylabel('Frequency')
    # plt.legend()

    plt.show()
#mu is the mean
```

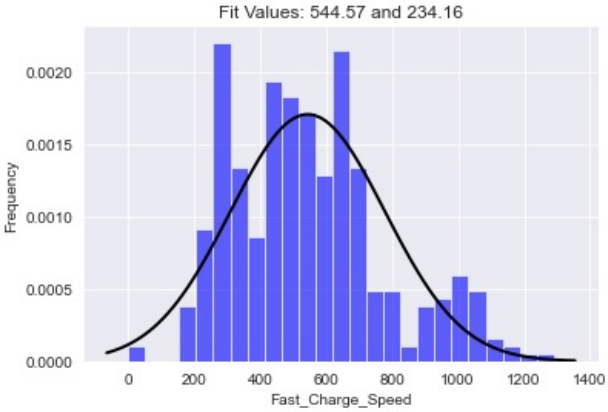


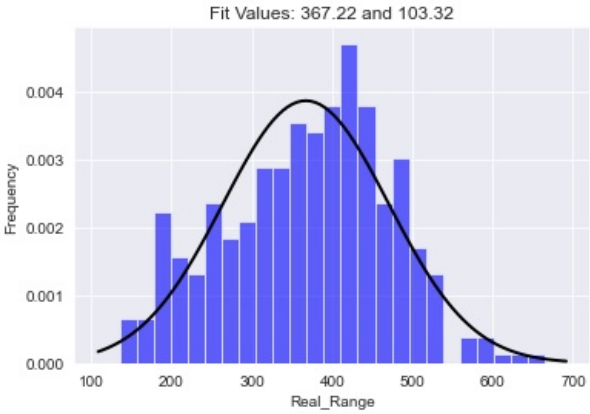


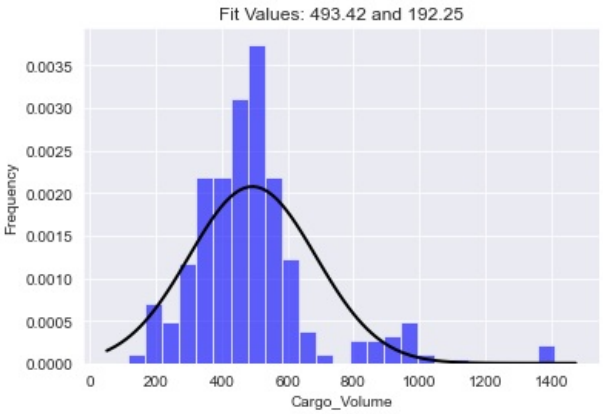


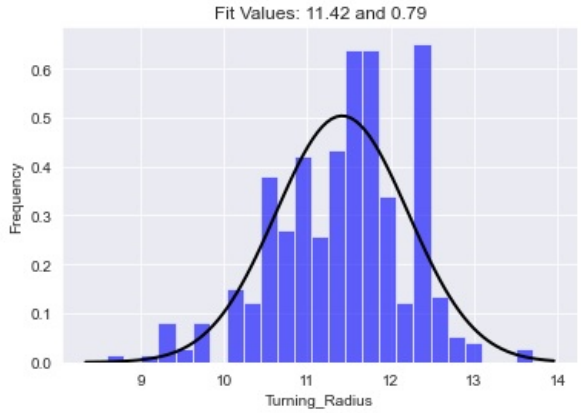


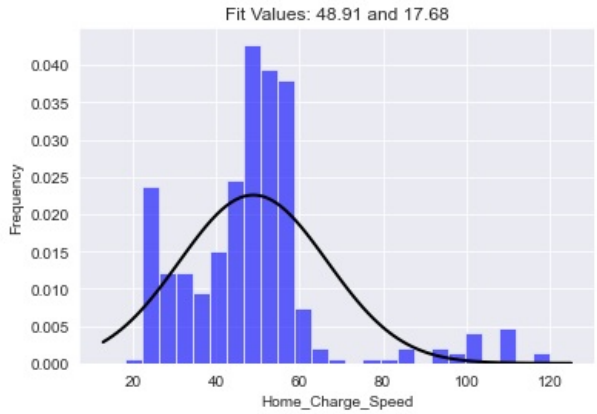

Fit Values: 194.87 and 31.97
Frequency
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
125
150
175
200
225
250
275
300
Efficiency

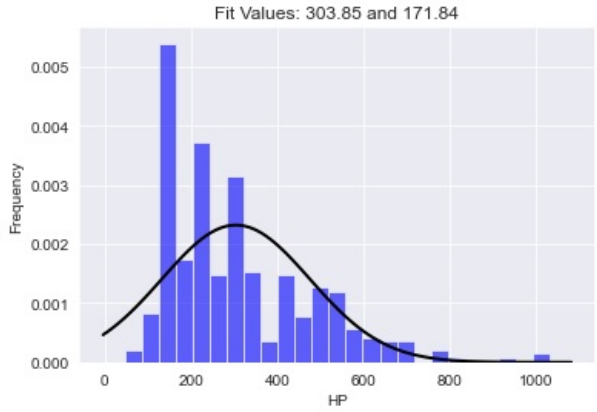

Fit Values: 544.57 and 234.16
Frequency
0.0020
0.0015
0.0010
0.0005
0.0000
0
200
400
600
800
1000
1200
1400
Fast_Charge_Speed

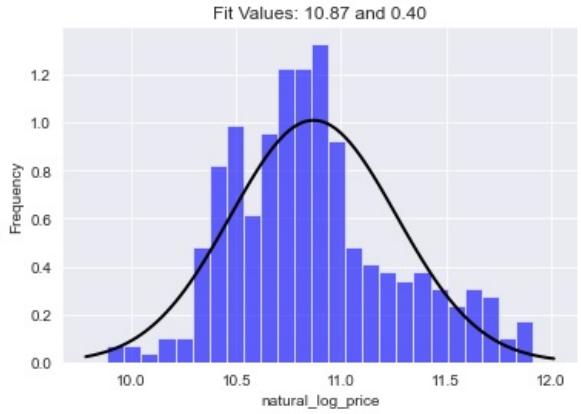

Fit Values: 367.22 and 103.32
Frequency
0.004
0.003
0.002
0.001
0.000
100
200
300
400
500
600
700
Real_Range


Fit Values: 493.42 and 192.25
Frequency
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
200
400
600
800
1000
1200
1400
Cargo_Volume









In [254…
```
machinelearn_df = outlier_df.copy()
machinelearn_df=machinelearn_df.drop(['z_score_price'], axis=1)
```

In [255…
```
machinelearn_df.to_csv('Machine_Learn.csv',index=False)
```

Throwing all variables at the model

In [120…
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import sklearn.metrics as metrics
import statsmodels.api as sm
```

In [121…
```
dataset = pd.read_csv('Machine_Learn.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [122…
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [123…
```
# Training the Multiple Linear Regression model on the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[123]:
```
LinearRegression()
```

In [ ]:
```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

In [125…
```
# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(y_test,y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 8005.60777811615
RMSE: 11022.217965155925
R-Squared: 0.7741465397809325
```

In [126…
```
#Applying K-Fold Cross validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=regressor, X=X_train,y=y_train,cv=10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100)) # Here we take the avg of all 10 accuracy predictions
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))# mean that 10 acuuracies falls more or less
```

```
Accuracy: 75.17 %
Standard Deviation: 8.65 %
```

Backward Elimination in Machine Learning

In [127…
```
df2 = dataset.loc[:,:'Station_Estate']
```

In [128…
```
# list(df2.columns)
```

In [129…
```
X_new=df2.copy()
X_new = sm.add_constant(X_new)
```

In [130…
```
model = sm.OLS(y, X_new).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.815
Model:                            OLS   Adj. R-squared:                  0.804
Method:                 Least Squares   F-statistic:                     71.23
Date:                Thu, 09 Nov 2023   Prob (F-statistic):          1.25e-110
Time:                        20:10:50   Log-Likelihood:                -3869.6
No. Observations:                 361   AIC:                             7783.
Df Residuals:                     339   BIC:                             7869.
Df Model:                          21
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                -3.245e+04   1.35e+04     -2.406      0.017    -5.9e+04   -5917.707
Battery_Size           946.8734    331.966      2.852      0.005     293.900    1599.847
Towing_Capacity         -3.2740      2.338     -1.400      0.162      -7.872       1.324
Number_of_Seats       1598.2451   1228.563      1.301      0.194    -818.321    4014.812
Acceleration          3046.1950    677.455      4.497      0.000    1713.650    4378.740
Top_Speed              280.9864     56.890      4.939      0.000     169.084     392.889
Efficiency              49.6651    137.365      0.362      0.718    -220.529     319.860
Fast_Charge_Speed       15.2583      4.969      3.071      0.002       5.485      25.032
Real_Range            -116.5449     64.585     -1.805      0.072    -243.582      10.492
All_Wheel_Drive      -9739.1179   4682.496     -2.080      0.038   -1.89e+04    -528.712
Forward_Wheel_Drive  -9379.3305   4677.352     -2.005      0.046   -1.86e+04    -179.043
Rear_Wheel_Drive     -1.333e+04   4811.968     -2.771      0.006   -2.28e+04   -3869.791
Safety_Rating_No     -1.698e+04   6770.572     -2.508      0.013   -3.03e+04   -3661.209
Safety_Rating_Yes    -1.547e+04   6795.309     -2.277      0.023   -2.88e+04   -2108.202
Cargo_Volume            -5.1640      4.732     -1.091      0.276     -14.472       4.144
Turning_Radius       -1768.3830   1332.417     -1.327      0.185   -4389.228     852.462
Home_Charge_Speed       95.3621     43.592      2.188      0.029       9.618     181.106
HP                      71.0877     11.167      6.366      0.000      49.123      93.053
Cabriolet             8851.9441   5992.012      1.477      0.141   -2934.262    2.06e+04
Coupe                -5.646e-12   1.61e-12     -3.517      0.000    -8.8e-12   -2.49e-12
Hatchback            -5303.8433   2285.517     -2.321      0.021   -9799.425    -808.262
Liftback_Sedan       -1.983e+04   4150.393     -4.779      0.000    -2.8e+04   -1.17e+04
SUV                  -8343.0218   2150.663     -3.879      0.000   -1.26e+04   -4112.697
Sedan                   45.0896   3105.952      0.015      0.988   -6064.275    6154.455
Small_Passenger_Van  -9363.5290   6760.760     -1.385      0.167   -2.27e+04    3934.794
Station_Estate        1494.0187   3438.252      0.435      0.664   -5268.977    8257.014
==============================================================================
Omnibus:                       33.096   Durbin-Watson:                   1.954
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               75.153
Skew:                           0.470   Prob(JB):                     4.79e-17
Kurtosis:                       5.028   Cond. No.                     1.07e+17
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 7.62e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [131…
```
#Removing Sedan
X_new = X_new[['Battery_Size','Towing_Capacity','Number_of_Seats','Acceleration','Top_Speed','Efficiency', 'Fas
 'Coupe','Hatchback','Liftback_Sedan','SUV','Small_Passenger_Van','Station_Estate']]
```

In [ ]:
```
X_new = sm.add_constant(X_new)
model = sm.OLS(y, X_new).fit()
print(model.summary())
```

In [133…
```
#Removing Efficiency
X_new = X_new[['Battery_Size','Towing_Capacity','Number_of_Seats','Acceleration','Top_Speed', 'Fast_Charge_Spee
 'Coupe','Hatchback','Liftback_Sedan','SUV','Small_Passenger_Van','Station_Estate']]
```

In [ ]:
```
X_new = sm.add_constant(X_new)
model = sm.OLS(y, X_new).fit()
print(model.summary())
```

In [135…
```
#Removing Station Estate
X_new = X_new[['Battery_Size','Towing_Capacity','Number_of_Seats','Acceleration','Top_Speed', 'Fast_Charge_Spee
 'Coupe','Hatchback','Liftback_Sedan','SUV','Small_Passenger_Van']]
```

In [ ]:
```
X_new = sm.add_constant(X_new)
model = sm.OLS(y, X_new).fit()
print(model.summary())
```

In [137…
```
#Cargo Volume
X_new = X_new[['Battery_Size','Towing_Capacity','Number_of_Seats','Acceleration','Top_Speed', 'Fast_Charge_Spee
 'Coupe','Hatchback','Liftback_Sedan','SUV','Small_Passenger_Van']]
```

In [ ]:
```
X_new = sm.add_constant(X_new)
model = sm.OLS(y, X_new).fit()
print(model.summary())
```

In [139…
```
#Cabriolet
```

```
X_new = X_new[['Battery_Size','Towing_Capacity','Number_of_Seats','Acceleration','Top_Speed', 'Fast_Charge_Spee
 'Coupe','Hatchback','Liftback_Sedan','SUV','Small_Passenger_Van']]
```

In [ ]:
```
X_new = sm.add_constant(X_new)
model = sm.OLS(y, X_new).fit()
print(model.summary())
```

In [141…
```
#Seats
X_new = X_new[['Battery_Size','Towing_Capacity','Acceleration','Top_Speed', 'Fast_Charge_Speed','Real_Range','A
 'Coupe','Hatchback','Liftback_Sedan','SUV','Small_Passenger_Van']]
```

In [ ]:
```
X_new = sm.add_constant(X_new)
model = sm.OLS(y, X_new).fit()
print(model.summary())
```

In [143…
```
#TurnRadius
X_new = X_new[['Battery_Size','Towing_Capacity','Acceleration','Top_Speed', 'Fast_Charge_Speed','Real_Range','A
 'Coupe','Hatchback','Liftback_Sedan','SUV','Small_Passenger_Van']]
```

In [ ]:
```
X_new = sm.add_constant(X_new)
model = sm.OLS(y, X_new).fit()
print(model.summary())
```

In [145…
```
#Coupe
X_new = X_new[['Battery_Size','Towing_Capacity','Acceleration','Top_Speed', 'Fast_Charge_Speed','Real_Range','A
```

In [ ]:
```
X_new = sm.add_constant(X_new)
model = sm.OLS(y, X_new).fit()
print(model.summary())
```

In [147…
```
#Tow
X_new = X_new[['Battery_Size','Acceleration','Top_Speed', 'Fast_Charge_Speed','Real_Range','All_Wheel_Drive','F
```

In [ ]:
```
X_new = sm.add_constant(X_new)
model = sm.OLS(y, X_new).fit()
print(model.summary())
```

Linear Regression with selected variables

In [149…
```
#Removes the first constant term.
S = X_new.iloc[:, 1:].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
S_train, S_test, y_train, y_test = train_test_split(S, y, test_size = 0.25, random_state = 0)
```

In [150…
```
y_train= y_train.reshape(len(y_train),1)
```

Applying Standardization & Normalization - Feature scaling

In [151…
```
# Standardization. We dont have to apply for dummy variables. Values can be of any range. Doesnt take care of o
# Always do standardization after splitting into train and test. We should apply the same scaler applied on tra
# from sklearn.preprocessing import StandardScaler
# sc= StandardScaler()
# S_train[:,[0,1,2,3,4,10,11]] = sc.fit_transform(S_train[:,[0,1,2,3,4,10,11]])
# S_test[:,[0,1,2,3,4,10,11]] = sc.transform(S_test[:,[0,1,2,3,4,10,11]])
```

In [152…
```
# Applying Normalization. Sets all values between 0 & 1. For both the dependent and independent variables
from sklearn.preprocessing import MinMaxScaler
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

S_train[:,[0,1,2,3,4,10,11]] = scaler_x.fit_transform(S_train[:,[0,1,2,3,4,10,11]])
S_test[:,[0,1,2,3,4,10,11]] = scaler_x.transform(S_test[:,[0,1,2,3,4,10,11]])

y_train = scaler_y.fit_transform(y_train)

y_train = np.ravel(y_train)#Need to get back to the original format for model fitting
```

In [153…
```
# Training the Multiple Linear Regression model on the Training set
from sklearn.linear_model import LinearRegression
regressor_1 = LinearRegression()
regressor_1.fit(S_train, y_train)
```

Out[153]:
```
LinearRegression()
```

In [154…
```
# Predicting the Test set results
#Transforming the scaled values back to its original form
y_pred = scaler_y.inverse_transform(regressor_1.predict(S_test).reshape(-1,1))
```

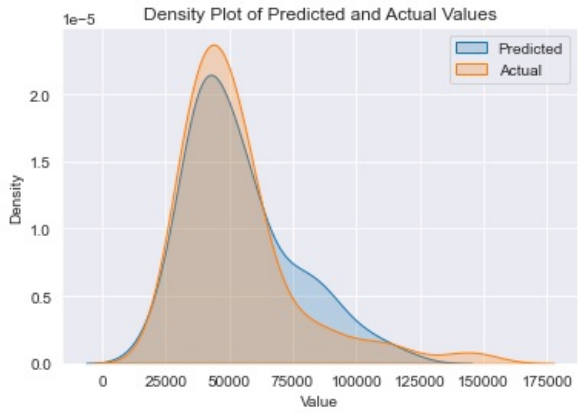
In [155…
```
# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test,y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 7806.981594280928
RMSE: 10929.646700689236
R-Squared: 0.7779243172421941
```

In [156…
```
#Applying K-Fold Cross validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=regressor_1, X=S_train,y=y_train,cv=10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100)) # Here we take the avg of all 10 accuracy predictions
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 75.97 %
Standard Deviation: 8.43 %
```

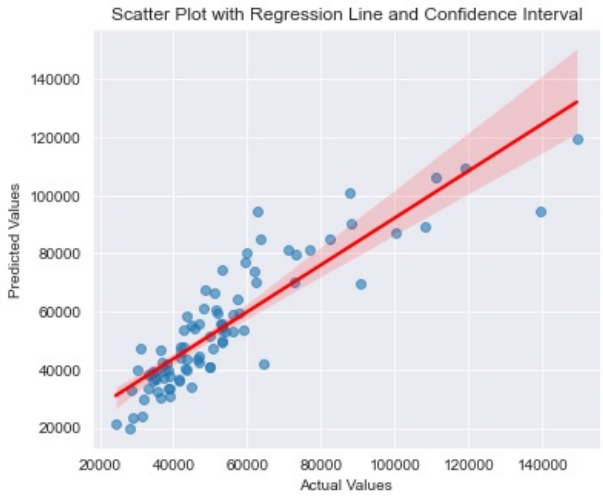
In [157…
```
# Create a KDE plot for y_pred
y_pred = np.ravel(y_pred)

sns.kdeplot(y_pred, label='Predicted', shade=True)

# Create a KDE plot for z_test
sns.kdeplot(y_test, label='Actual', shade=True)

plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Density Plot of Predicted and Actual Values')
plt.legend()
plt.show()
```



In [158…
```
# sns.residplot(x=y_test,y=y_pred,scatter_kws={'s':10},line_kws={'color':'red'})
# plt.xlabel('Predicted_Values')
# plt.ylabel('Residuals')
# plt.title('Residual Plot')
# plt.legend()
# plt.show()
```

In [159…
```
plt.figure(figsize=(13, 5))

plt.subplot(1,2,1)
sns.regplot(x=y_test, y=y_pred, scatter_kws={'alpha':0.6}, line_kws={'color':'red'})
# Add labels and title
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Scatter Plot with Regression Line and Confidence Interval')

plt.subplot(1,2,2)
sns.regplot(x=np.log(y_test), y=np.log(y_pred), scatter_kws={'alpha':0.6}, line_kws={'color':'yellow'})
# Add labels and title
plt.xlabel('Log Actual Values')
plt.ylabel('Log Predicted Values')
plt.title('Log Scatter Plot with Regression Line and Confidence Interval')
```

Out[159]:
```
Text(0.5, 1.0, 'Log Scatter Plot with Regression Line and Confidence Interval')
```



In [160…
```
# data_vis = pd.DataFrame()
# data_vis['y_pred']= pd.DataFrame(y_pred)
# data_vis
```

In [161…
```
# data_vis['y_test']= pd.DataFrame(y_test)
# data_vis
```

Support Vector Regression Full Dataset - Need feature scaling

In [162…
```
dataset = pd.read_csv('Machine_Learn.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [163…
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [164…
```
#Need to reshape y so that we can apply minmax scaler
y_train= y_train.reshape(len(y_train),1)
```

In [165…
```
from sklearn.preprocessing import MinMaxScaler
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_train[:,[0,1,2,3,4,5,6,7,13,14,15,16]] = scaler_X.fit_transform(X_train[:,[0,1,2,3,4,5,6,7,13,14,15,16]])
X_test[:,[0,1,2,3,4,5,6,7,13,14,15,16]] = scaler_X.transform(X_test[:,[0,1,2,3,4,5,6,7,13,14,15,16]])

y_train = scaler_y.fit_transform(y_train)

y_train = np.ravel(y_train)#the fit method expects y to be a flat array. That's why you have to use ravel, to c
```

In [166…
```
#Non linear kernel-rbf. Can capture non-linear boundary.
from sklearn.svm import SVR
regressor_2 = SVR(kernel='rbf')
regressor_2.fit(X_train,y_train)
```

Out[166]:
```
SVR()
```

In [167…
```
#Transforming the scaled values back to its original form
y_pred = scaler_y.inverse_transform(regressor_2.predict(X_test).reshape(-1,1))
```

In [168…
```
# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test,y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 8523.384731201259
RMSE: 11290.153072859306
R-Squared: 0.7630327006768244
```

In [169…
```
#Applying K-Fold Cross validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=regressor_2, X=X_train,y=y_train,cv=10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100)) # Here we take the avg of all 10 accuracy predictions
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 75.79 %
Standard Deviation: 5.80 %
```

Hyperparameter tuning for SVM - C & Gamma

In [ ]:
```
from sklearn.model_selection import GridSearchCV

c_range = np.arange(1,100,2)
#Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly
gamma_range = np.arange(0.001,1,0.1)
#Kernel coefficient for 'rbf'. It defines how far influences the calculation of plausible line of separation. w

param_grid =dict(C=c_range,gamma=gamma_range)

#param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001]}

grid = GridSearchCV(estimator=regressor_2,param_grid=param_grid,cv=6,scoring="r2",refit=True,verbose=2)
grid.fit(X_train,y_train)
best_accuracy = grid.best_score_
best_parameters = grid.best_params_
print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
print("Best Parameters:",best_parameters)
```

In [171…
```
#Grid Predictions on the test set
grid_pred = scaler_y.inverse_transform(grid.predict(X_test).reshape(-1,1))

# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, grid_pred)
mse = metrics.mean_squared_error(y_test, grid_pred)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test,grid_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 8291.426439613311
RMSE: 11003.475659498516
R-Squared: 0.7749139743690238
```

Support vector with selected variables

In [172…
```
S = X_new.iloc[:, 1:].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
S_train, S_test, y_train, y_test = train_test_split(S, y, test_size = 0.25, random_state = 0)
```

In [173…
```
y_train= y_train.reshape(len(y_train),1)
```

In [174…
```
from sklearn.preprocessing import MinMaxScaler
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

S_train[:,[0,1,2,3,4,10,11]] = scaler_x.fit_transform(S_train[:,[0,1,2,3,4,10,11]])
S_test[:,[0,1,2,3,4,10,11]] = scaler_x.transform(S_test[:,[0,1,2,3,4,10,11]])

y_train = scaler_y.fit_transform(y_train)

y_train = np.ravel(y_train)
```

In [175…
```
#Non linear kernel-rbf. Can capture non-linear boundary.
from sklearn.svm import SVR
regressor_3 = SVR(kernel='rbf')
regressor_3.fit(S_train,y_train)
```

Out[175]:
```
SVR()
```

In [176…
```
#Transforming the scaled values back to its original form
y_pred = scaler_y.inverse_transform(regressor_3.predict(S_test).reshape(-1,1))
```

In [177…
```
# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
```

```
r2 = metrics.r2_score(y_test,y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 8642.940929587307
RMSE: 11510.83943778656
R-Squared: 0.7536782559806408
```

In [178…
```
#Applying K-Fold Cross validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=regressor_3, X=S_train,y=y_train,cv=10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100)) # Here we take the avg of all 10 accuracy predictions
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 73.51 %
Standard Deviation: 6.80 %
```

Decision tree - Whole Dataset

In [179…
```
dataset = pd.read_csv('Machine_Learn.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [180…
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [181…
```
#Need to reshape y so that we can apply minmax scaler
y_train= y_train.reshape(len(y_train),1)
```

In [182…
```
from sklearn.preprocessing import MinMaxScaler
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_train[:,[0,1,2,3,4,5,6,7,13,14,15,16]] = scaler_X.fit_transform(X_train[:,[0,1,2,3,4,5,6,7,13,14,15,16]])
X_test[:,[0,1,2,3,4,5,6,7,13,14,15,16]] = scaler_X.transform(X_test[:,[0,1,2,3,4,5,6,7,13,14,15,16]])

y_train = scaler_y.fit_transform(y_train)

y_train = np.ravel(y_train)
```

In [183…
```
from sklearn.tree import DecisionTreeRegressor
# For pruning the tree change the ccp_alpha values to be slightly greater than 0. anything like 0.1 were teribl
regressor_4 = DecisionTreeRegressor(random_state=0,max_depth=7,ccp_alpha=0.0001)
regressor_4.fit(X_train,y_train)
```

Out[183]:
```
DecisionTreeRegressor(ccp_alpha=0.0001, max_depth=7, random_state=0)
```

In [184…
```
#Transforming the scaled values back to its original form
y_pred = scaler_y.inverse_transform(regressor_4.predict(X_test).reshape(-1,1))
```

In [185…
```
# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test,y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 5500.40847059079
RMSE: 7933.204830464125
R-Squared: 0.8829999374429516
```

In [186…
```
#Applying K-Fold Cross validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=regressor_4, X=X_train,y=y_train,cv=10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100)) # Here we take the avg of all 10 accuracy predictions
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

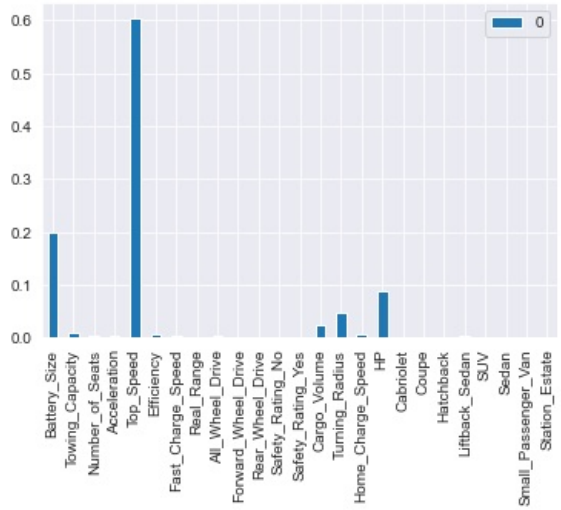
```
Accuracy: 75.49 %
Standard Deviation: 10.18 %
```

In [ ]:
```
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

In [ ]:
```
#Checking which feature is given most importance by the decison tree model
feature_names =dataset.iloc[:, :-1].columns
feature_importance = pd.DataFrame(regressor_4.feature_importances_,index=feature_names)
feature_importance
```

In [189…
```
#Same but in the form of a plot
feature_importance.head(25).plot(kind='bar')
```

Out[189]:
```
<AxesSubplot:>
```



In [190…
```
# from sklearn import tree
#
# feature_names = dataset.columns.tolist()
#
# fig = plt.figure(figsize=(25,20))
# _ = tree.plot_tree(regressor_4, feature_names=feature_names, filled=True)
```

Decision tree - Selected Values

In [191…
```
S = X_new.iloc[:, 1:].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
S_train, S_test, y_train, y_test = train_test_split(S, y, test_size = 0.25, random_state = 0)

y_train= y_train.reshape(len(y_train),1)
```

In [192…
```
from sklearn.preprocessing import MinMaxScaler
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

S_train[:,[0,1,2,3,4,10,11]] = scaler_x.fit_transform(S_train[:,[0,1,2,3,4,10,11]])
S_test[:,[0,1,2,3,4,10,11]] = scaler_x.transform(S_test[:,[0,1,2,3,4,10,11]])

y_train = scaler_y.fit_transform(y_train)

y_train = np.ravel(y_train)
```

In [193…
```
from sklearn.tree import DecisionTreeRegressor
regressor_5 = DecisionTreeRegressor(random_state=0,ccp_alpha=0.00117,max_depth=3)
regressor_5.fit(S_train,y_train)
```

Out[193]:
```
DecisionTreeRegressor(ccp_alpha=0.00117, max_depth=3, random_state=0)
```

In [194…
```
y_pred = scaler_y.inverse_transform(regressor_5.predict(S_test).reshape(-1,1))
```

In [195…
```
# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test,y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 7354.615116553926
RMSE: 11276.315586286177
R-Squared: 0.7636132105017801
```

In [196…
```
#Applying K-Fold Cross validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=regressor_5, X=S_train,y=y_train,cv=10)
```

```
print("Accuracy: {:.2f} %".format(accuracies.mean()*100)) # Here we take the avg of all 10 accuracy predictions
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 67.53 %
Standard Deviation: 21.74 %
```

Random Forest Regression - Whole dataset

In [197…
```
dataset = pd.read_csv('Machine_Learn.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [198…
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [199…
```
#Need to reshape y so that we can apply minmax scaler
y_train= y_train.reshape(len(y_train),1)
```

In [200…
```
from sklearn.preprocessing import MinMaxScaler
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_train[:,[0,1,2,3,4,5,6,7,13,14,15,16]] = scaler_X.fit_transform(X_train[:,[0,1,2,3,4,5,6,7,13,14,15,16]])
X_test[:,[0,1,2,3,4,5,6,7,13,14,15,16]] = scaler_X.transform(X_test[:,[0,1,2,3,4,5,6,7,13,14,15,16]])

y_train = scaler_y.fit_transform(y_train)

y_train = np.ravel(y_train)
```

In [201…
```
from sklearn.ensemble import RandomForestRegressor
#10 trees or 10 estimator
regressor_6 = RandomForestRegressor(n_estimators=35,random_state=0)
regressor_6.fit(X_train,y_train)
```

Out[201]:
```
RandomForestRegressor(n_estimators=35, random_state=0)
```

In [202…
```
#Transforming the scaled values back to its original form
y_pred = scaler_y.inverse_transform(regressor_6.predict(X_test).reshape(-1,1))
```

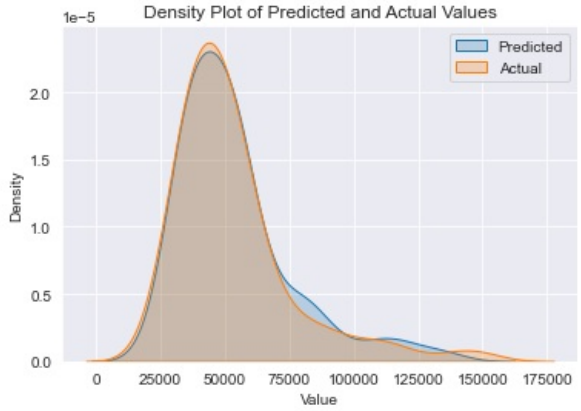
In [203…
```
# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test,y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 5045.11463381339
RMSE: 7520.641548114894
R-Squared: 0.894852600075629
```

In [204…
```
#Applying K-Fold Cross validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=regressor_6, X=X_train,y=y_train,cv=10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100)) # Here we take the avg of all 10 accuracy predictions
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

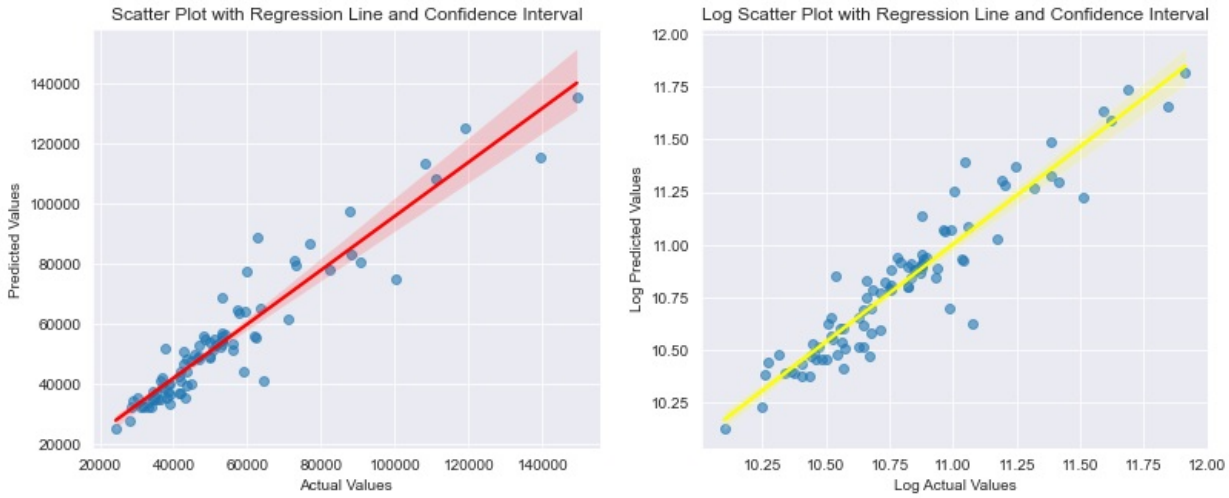
```
Accuracy: 85.41 %
Standard Deviation: 5.98 %
```

In [205…
```
# Create a KDE plot for y_pred
y_pred = np.ravel(y_pred)

sns.kdeplot(y_pred, label='Predicted', shade=True)

# Create a KDE plot for z_test
sns.kdeplot(y_test, label='Actual', shade=True)

plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Density Plot of Predicted and Actual Values')
plt.legend()
plt.show()
```



In [206…
```
plt.figure(figsize=(13, 5))

plt.subplot(1,2,1)
sns.regplot(x=y_test, y=y_pred, scatter_kws={'alpha':0.6}, line_kws={'color':'red'})
# Add labels and title
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Scatter Plot with Regression Line and Confidence Interval')

plt.subplot(1,2,2)
sns.regplot(x=np.log(y_test), y=np.log(y_pred), scatter_kws={'alpha':0.6}, line_kws={'color':'yellow'})
# Add labels and title
plt.xlabel('Log Actual Values')
plt.ylabel('Log Predicted Values')
plt.title('Log Scatter Plot with Regression Line and Confidence Interval')
```

Out[206]:
```
Text(0.5, 1.0, 'Log Scatter Plot with Regression Line and Confidence Interval')
```



Hyperparameter Tuning - Random Forest

In [207…
```
# Tuning n-estimators & max features
from sklearn.model_selection import GridSearchCV
import numpy as np

max_features_range = np.arange(1,15,1)#goes from 1 to 14
n_estimators_range = np.arange(10,280,4)
param_grid =dict(max_features = max_features_range,n_estimators = n_estimators_range)

rf = RandomForestRegressor()

grid = GridSearchCV(estimator=rf,param_grid=param_grid,cv=5,scoring="r2")
```

In [208…
```
grid.fit(X_train,y_train)
best_accuracy = grid.best_score_
best_parameters = grid.best_params_
print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
print("Best Parameters:",best_parameters)
```

```
Best Accuracy: 87.66 %
Best Parameters: {'max_features': 12, 'n_estimators': 114}
```

In [209…
```
#Grid Predictions on the test set
grid_pred = scaler_y.inverse_transform(grid.predict(X_test).reshape(-1,1))

# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, grid_pred)
mse = metrics.mean_squared_error(y_test, grid_pred)
rmse = np.sqrt(mse)
```

```
r2 = metrics.r2_score(y_test,grid_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 5028.622582096263
RMSE: 7830.152671736669
R-Squared: 0.8860198514578783
```

In [ ]:
```
grid_results = pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.cv_results_["mean_test_sco
grid_results #mean test score of the 4 folds
```

In [ ]:
```
#groupby used to segment the data into groups based on the 2 hyperparameters: max_features and n_estimators.
grid_contour = grid_results.groupby(['max_features','n_estimators']).mean()
grid_contour
```

In [ ]:
```
#Pivoting the data to make it look proper
grid_reset = grid_contour.reset_index()
grid_reset.columns = ['max_features', 'n_estimators', 'Accuracy']
grid_pivot = grid_reset.pivot('max_features', 'n_estimators')
grid_pivot
```

In [213…
```
#Assigning the pivoted data into the respective *x, y* and *z* variable
x = grid_pivot.columns.levels[1].values #are the n_estimators. OR column names
y = grid_pivot.index.values #are the features
z = grid_pivot.values# are the mean values(scores)
```

In [97]:
```
#!pip install plotly
```

In [214…
```
import plotly.graph_objects as go

# X and Y axes labels
layout = go.Layout(xaxis=go.layout.XAxis(title=go.layout.xaxis.Title(text='n_estimators')),yaxis=go.layout.YAxi

fig = go.Figure(data=[go.Contour(z=z, x=x, y=y)],layout=layout) #creates a figure object using go.Figure().Take

fig.update_layout(title='Hyperparameter tuning', autosize=False,width=700, height=700,margin=dict(l=65, r=50, b

fig.show()
```

Random Forest - Selected Variables

In [99]:
```
S = X_new.iloc[:, 1:].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
S_train, S_test, y_train, y_test = train_test_split(S, y, test_size = 0.25, random_state = 0)

y_train= y_train.reshape(len(y_train),1)
```

In [100…
```
from sklearn.preprocessing import MinMaxScaler
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

S_train[:,[0,1,2,3,4,10,11]] = scaler_x.fit_transform(S_train[:,[0,1,2,3,4,10,11]])
S_test[:,[0,1,2,3,4,10,11]] = scaler_x.transform(S_test[:,[0,1,2,3,4,10,11]])

y_train = scaler_y.fit_transform(y_train)

y_train = np.ravel(y_train)
```

In [101…
```
from sklearn.ensemble import RandomForestRegressor
#10 trees or 10 estimator
regressor_7 = RandomForestRegressor(n_estimators=26,random_state=0)
regressor_7.fit(S_train,y_train)
```

Out[101]:
```
RandomForestRegressor(n_estimators=26, random_state=0)
```

In [102…
```
y_pred = scaler_y.inverse_transform(regressor_7.predict(S_test).reshape(-1,1))
```

In [103…
```
# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test,y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 5146.183162470459
RMSE: 8004.040993638541
R-Squared: 0.8809012050784588
```

In [104…
```
#Applying K-Fold Cross validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=regressor_7, X=S_train,y=y_train,cv=10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100)) # Here we take the avg of all 10 accuracy predictions
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 83.59 %
Standard Deviation: 6.81 %
```

XGBoost Model with All Variables

In [105…
```
#!pip install xgboost
```

In [215…
```
dataset = pd.read_csv('Machine_Learn.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [216…
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [217…
```
#Need to reshape y so that we can apply minmax scaler
y_train= y_train.reshape(len(y_train),1)
```

In [218…
```
from sklearn.preprocessing import MinMaxScaler
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_train[:,[0,1,2,3,4,5,6,7,13,14,15,16]] = scaler_X.fit_transform(X_train[:,[0,1,2,3,4,5,6,7,13,14,15,16]])
X_test[:,[0,1,2,3,4,5,6,7,13,14,15,16]] = scaler_X.transform(X_test[:,[0,1,2,3,4,5,6,7,13,14,15,16]])

y_train = scaler_y.fit_transform(y_train)

y_train = np.ravel(y_train)
```

In [219…
```
from xgboost import XGBRegressor
#10 trees or 10 estimator
regressor_8 = XGBRegressor(random_state =0)
regressor_8.fit(X_train,y_train)
```

Out[219]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=0, ...)
```

In [220…
```
#Transforming the scaled values back to its original form
y_pred = scaler_y.inverse_transform(regressor_8.predict(X_test).reshape(-1,1))
```

In [221…
```
# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test,y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

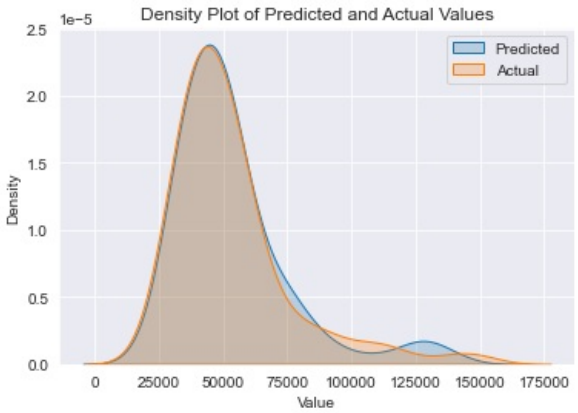
```
Results of sklearn.metrics:
MAE: 5105.078252844302
RMSE: 7200.305584025607
R-Squared: 0.9036191802961426
```

In [ ]:
```
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

In [223…
```
#Applying K-Fold Cross validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=regressor_8, X=X_train,y=y_train,cv=10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100)) # Here we take the avg of all 10 accuracy predictions
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 84.91 %
Standard Deviation: 5.19 %
```

In [224…
```
# Create a KDE plot for y_pred
y_pred = np.ravel(y_pred)

sns.kdeplot(y_pred, label='Predicted', shade=True)

# Create a KDE plot for y_test
sns.kdeplot(y_test, label='Actual', shade=True)

plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Density Plot of Predicted and Actual Values')
plt.legend()
plt.show()
```



Hyper Parameter tuning XGBoost Regression

In [117…
```
# Tuning n-estimators, max depth, gamma, learning_rate, min_child_weight
from sklearn.model_selection import GridSearchCV

n_estimators_range = np.arange(10,200,5)
max_depth_range = np.arange(1,24,2)
gamma_range = np.arange(0.001,0.02,0.005)
learning_rate_range = np.arange(0.01,0.1,0.03)
min_child_range = np.arange(1,20,2)

param_grid =dict(n_estimators = n_estimators_range,max_depth = max_depth_range,gamma=gamma_range,learning_rate=

grid = GridSearchCV(estimator=regressor_8,param_grid=param_grid,cv=5,scoring="r2")
```

In [118…
```
grid.fit(X_train,y_train)
best_accuracy = grid.best_score_
best_parameters = grid.best_params_
print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
print("Best Parameters:",best_parameters)
```

```
Best Accuracy: 87.14 %
Best Parameters: {'gamma': 0.001, 'learning_rate': 0.09999999999999999, 'max_depth': 5, 'min_child_weight': 5,
'n_estimators': 60}
```

In [119…
```
#Grid Predictions on the test set
grid_pred = scaler_y.inverse_transform(grid.predict(X_test).reshape(-1,1))

# Evaluating the Model Performance
mae = metrics.mean_absolute_error(y_test, grid_pred)
mse = metrics.mean_squared_error(y_test, grid_pred)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test,grid_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 4839.957249611801
RMSE: 7683.599038247038
R-Squared: 0.8902465588703369
```

In [ ]:
```
# # Dictionary with hyperparameter tuning
# search_space = {
#  "n_estimators": [75,80,85,95,100,110,120],
#  "max_depth": [3,4,5,6],
#  "gamma": [0.003,0.0035,0.004,0.0045],
#  "learning_rate":[0.060,0.052,0.049,0.047,0.045,0.04],
#  #"lambda": [2,3,4,5], #l2 regularization term on weights
#  "min_child_weight": [5,6,7,8]
# }
```

In [ ]:
```
# #Performing Grid Search for XGboost with CV of 5
# from sklearn.model_selection import GridSearchCV
# GS = GridSearchCV(estimator=regressor_8,
#                   param_grid=search_space,
#                   scoring="r2",
#                   cv=5,
#                   verbose=4)
```

In [ ]:
```
# #You perform Grid search only on the training set. This is not done on the test set.
# GS.fit(X_train,y_train)
# best_accuracy = GS.best_score_
# best_parameters = GS.best_params_
# print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
# print("Best Parameters:",best_parameters)
```