



Master Thesis

Carlyst: Used Cars Value Predictor

Master in Data Science

KSchool (2020 - 2021)

Author: Carlos Espejo Peña

Sevilla, July 2021

Table of Contents

1. Introduction	3
2. Tools used on project development	4
3. Raw data description	5
3.1. Data Source and Collection.....	5
3.2. Data Scraping	6
3.3. Definition of Attributes	9
4. Exploratory Data Analysis (EDA)	10
4.1. Data Cleansing	10
4.2. Exploration and definition of outliers	13
4.3. Correlation of fields	17
5. Methodology	19
5.1. MVP: Setting a starting point.....	20
5.2. Feature Engineering.....	22
5.2.1. One-Hot Encode: low cardinality variables	22
5.2.2. Target Encode: high cardinality variables	23
5.2.3. Exploring numerical features	25
5.2.4. Column Transformation: Features Normalization	26
5.3. Data Model	28
5.3.1. Machine Learning Algorithms tested	28
5.3.2. Benchmarking of Regression Models.....	29
5.3.3. Measuring Column Transformers impact	33
5.3.4. Hyperparameter Tuning: CV methods	35
5.4. Building the optimal model.....	38
5.4.1. Evaluation of the model.....	38
5.4.2. Actual vs predicted values	39
5.4.3. SHAP figures: Feature importance.....	41
5.4.4. Model persistence	42
6. Frontend with Streamlit: User manual	44
6.1. How was the backend built?	44
6.2. User Manual.....	44
7. Conclusions and Future steps.....	47
Appendix: Reference links and articles.....	48

1.Introduction

The aim of this project is to provide a tool which can evaluate the value of used cars. The scope of this prediction is based on the Spanish second-hand car market. In this way, both sellers and buyers can find in this tool a solution to estimate the best and most fair price for each vehicle. What is more, this golden price could be obtained by solving this question: how do the characteristics of the vehicle impact on the final price?

Nowadays, there are a wide number of online buying portals to look for second-hand cars. However, the different offers that appear on each of these sites are not regulated but for the market itself. There is an uncertainty on the price to be paid for a second-hand vehicle. This assumption leads to the idea that it is difficult to find a suitable price for cars, especially to sellers when the demand is high or to purchasers when the offer is low.

Here is where the necessity arises and Carlyst ^[1.2] emerges. The name of this app resides on the combination of two terms: “Car” and “Analyst”. It was chosen due to the fact that the algorithm studies the price of similar cars and try to evaluate the fair value, providing a final recommended appraisal. This data-driven solution is based on the technical specifications of 55,326 cars offer on one car web service, coches.com ^[1.1] based on Spain.

The main goal of this project was to design an algorithm which could provide accurate results that can be used by any person in Spain who is looking for more information about a used car. Therefore, this app was focused on the prediction of cars market value up to 100,000€. A higher number of cars below this amount could assure an enhance on the performance of the model, without leading to a confusion on the final price.

To reach this objective, a machine learning algorithm was used to perform these automatic predictions. Behind the scenes of the tool, a regression model is running to make the forecasted values possible, CatBoost ^[1.3]. This open-source gradient boosting algorithm was developed by Yandex ^[1.4], a Russian tech-company. Through this document, I will explain why the final decision was to use this model as the best operator for Carlyst.

Along this paper, I will cover the Definition and Collection of the data, followed by the Exploratory Data Analysis (EDA) to define the most suitable data to feed the model. Afterwards, the full Methodology will be presented, covering from the development of the MVP to the explanation of the process to find the best model for this problem. Finally, the Summary of the most remarkable solutions will be exposed and how to interact with the Front-End to make our own decisions by using this tool.

2.Tools used on project development

To produce this project, several technical tools were used to make it possible:

- Linux ^[2.1]: Operative system (OS) used under Xubuntu ^[2.2] flavour. Xubuntu is a Linux distribution based on Ubuntu. Shell command-line interpreter has been used to interact with the system through scripting language.
- Oracle VM VirtualBox ^[2.3]: is an open-source hosted hypervisor developed by Oracle Corporation. It allows to launch a virtual machine with a different OS on top of a physical computer.
- Python ^[2.4]: Main programming language used on this project. Nowadays, it is the most extended language used for Data Science, so it is perfect for working with Machine Learning. Main packages used:
 - Data analysis and manipulation: pandas, numpy
 - Web scrapping: requests, selenium, lxml, BeautifulSoup4
 - Visualization: matplotlib, seaborn, streamlit
 - Machine learning: scikit-learn, shap, catboost, xgboost

Version used: python-3.7.10

- Anaconda ^[2.5]: is an open-source python distribution to create controlled environments for different projects. Environments can be later shared by using yaml export files.
- JupyterLab ^[2.6]: Web-based interactive notebook interface developed by Project Jupyter team. It allows to use several jupyter notebooks under the same environment. It also helps on working with python, markdown and other programming languages, offering an easy-to-use IDE.
- Heroku ^[2.7]: platform as a service (PaaS) offered by Salesforce, which enables developers to build, run and operate applications entirely in the cloud.
- Markdown ^[2.8]: lightweight markup language for creating formatted text using a plain-text editor. It is widely used for readme files. In this project, it will be also included on jupyter notebooks.
- HTML ^[2.9]: HyperText Markup Language is used to create web pages. It has been implemented on several elements of the front-end of this project as well as on notebooks to create more complex and flexible layout formats.

3.Raw Data Description

In this chapter it will be explained how the data was obtained from scratch. Starting by how the original source contained the information and how it ended up in a consumable dataset. Moreover, I will explain every obtained attribute that conforms the original conformed file.

3.1 Data Source and Collection

The data source of this project is based on one of the most visited second-hand car sale portals in Spain, coches.com ^[3.1.1]. On this site, sellers, both individuals and dealerships, can easily publish an ad and buyers can search for their future car.

Inside this web portal, it is possible to find a second-hand section. This was the starting point from which this project was built. From this site, we can see several adverts from which we can take all the technical and qualitative specifications from the vehicle. Moreover, the html tags for each of the characteristics are opened on each site, so they can be consulted using web scraping. Therefore, this car portal was the best choice in terms of obtaining a rich dataset since it was accessible using known tools.

Another point to consider is that adverts get refreshed on a daily basis, so it requires a big computational effort to create a refresh on a daily basis. Thereupon, I decided to obtain a static dataset from the 9th of March of 2021. In this way the final algorithm will not require any maintenance. Although it is true that the scalability of the model is lost, it is also important to take into account that the whole project could be replicated using a local machine only. Through this method, I could create a feasible tool for academic purposes. In case this application would evolve to a commercial solution, the scalation to a cloud platform should be considered, i.e., AWS ^[3.1.2].

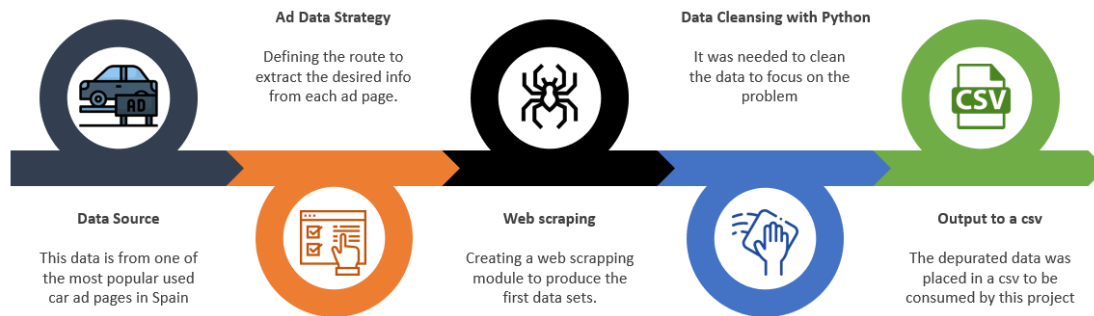
Before creating the final dataset, the data offered on each page was explored to see which fields were standardized. By defining this route to extract the desired information from each ad page, the final data was produced to be used along this project. The ultimate attribute list used will be explored on the 3.3 point of this chapter (Attributes Definition).

Once the fields were selected, a web scraping module was created to deal with the data collection process. This module made possible to generate the final dataset following an automatic extraction. The details of this process will be covered on 3.2 point (Data Scraping).

Although, a bulk download made possible to have a dataset, it was needed to be cleaned before starting to apply any machine learning model. Data cleansing helps on focusing on the problem, once the raw source is prepared.

An initial csv file ^[3.1.3] was produced to hold the extracted raw data. However, after the exploratory data analysis (EDA) was performed, a final output was created with the final dataset to start applying the machine learning methodology.

The pipeline below summarizes the data collection process:



3.2 Data Scraping

As mentioned on the data collection process, the final dataset used for this project consists on an extraction produced on the 9th of March of 2021. To this aim, web scraping ^[3.2.1] was the method used in order to extract each needed attribute from coches.com.

Initially, the idea was to produce a price prediction for both second-hand and renting market. However, due to the lack of renting data, this project focused on generating a price predictor for used cars.

The initial number of cars extracted was 66,533. Nevertheless, there is a limitation on the portal to produce no more than 10,000 results in one search. Therefore, it was needed to deep dive on the maximum number of items per page. Eventually, it was easy to determine that there are always 20 ads per page and that the maximum number of pages per search was of 500. Even that each search showed a major number of cars in stock, only the top 10,000 were searchable on the site.

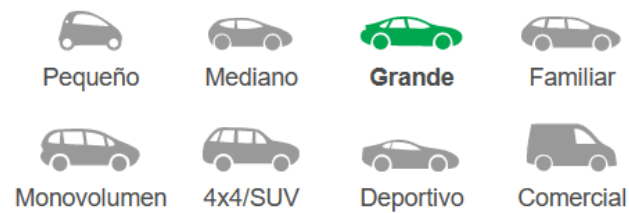
Once known this limitation on the web portal, it was needed to create a different approach to the project to increase the size of the final dataset. A sample of 10,000 cars could be enough to build a regression machine learning model. Nonetheless, the initial aim was to create a robust and consistent model so as to reduce the final number of outliers that could deviate the resulting prediction. Thereupon, it was decided to add a granularity level to the search, increasing the number of samples. To achieve this, the new exploration on the page was based on the type of vehicle:

Tipo



Ver 91.903 resultados

Tipo



Ver 8.570 resultados

Through this method, it was possible to increase the number of samples in the dataset from 10,000 to 66,533. The extraction was performed based on the following 8 types of cars: small, medium, big, familiar, minivan, off-road, sports and van. Nevertheless, it is important to consider that some cars are categorized under more than 1 type. Moreover, not all the types included up to 10,000 car samples (see image above for 'big' category searches). This is why after merging all the data, the final number was not 80,000 as initially expected.

The global extraction was ready to obtain cars data. However, it was needed to decide which features should be considered for the machine learning model. By exploring each ad, it was possible to determine the general compulsory fields for each car in this portal. This is how it was decided to take the price, year, power, kilometres and the rest of attributes which conformed the final dataset. The description of each of them will be provided in the next point, 3.3 (Definition of Attributes).

Tech approach

Now that the web scraping use case is clear, it is time to have a look at the technical details. Inside the folder modules, we can find all the ones created for this project. For web scraping, it was used the *car_scraping.py* module.

Inside this module, we can find several functions to obtain the final data:

- *webpages_generator*: generates a list with all the links of the limit 500 pages. This list is created based on an initial link.

```
[3]: # Extract cars from all the types of cars
small_cars_url = 'https://www.coches.com/coches-segunda-mano/coches-pequenos-segunda-mano.htm?page='
medium_cars_url = 'https://www.coches.com/coches-segunda-mano/coches-medianos-segunda-mano.htm?page='
big_cars_url = 'https://www.coches.com/coches-segunda-mano/coches-grandes-segunda-mano.htm?page='
familiar_cars_url = 'https://www.coches.com/coches-segunda-mano/coches-familiares-segunda-mano.htm?page='
minivan_cars_url = 'https://www.coches.com/coches-segunda-mano/coches-monovolumen-segunda-mano.htm?page='
offroad_cars_url = 'https://www.coches.com/coches-segunda-mano/coches-4x4-todoterreno.htm?page='
sports_cars_url = 'https://www.coches.com/coches-segunda-mano/coches-deportivos-segunda-mano.htm?page='
van_cars_url = 'https://www.coches.com/coches-segunda-mano/furgonetas-segunda-mano.htm?page='
```

...

```
[5]: # Types of cars: webpages generators
small_cars_pages = webpages_generator(small_cars_url, init_pages=0, n_pages=141)
medium_cars_pages = webpages_generator(medium_cars_url, init_pages=0, n_pages=500)
big_cars_pages = webpages_generator(big_cars_url, init_pages=0, n_pages=500)
familiar_cars_pages = webpages_generator(familiar_cars_url, init_pages=0, n_pages=153)
minivan_cars_pages = webpages_generator(minivan_cars_url, init_pages=0, n_pages=425)
offroad_cars_pages = webpages_generator(offroad_cars_url, init_pages=0, n_pages=500)
sports_cars_pages = webpages_generator(sports_cars_url, init_pages=0, n_pages=286)
van_cars_pages = webpages_generator(van_cars_url, init_pages=0, n_pages=285)
```

- *cars_links_generator*: gets all the links previously generated and obtains all the links for each of the 20 cars contained in each page. It is based on regular expressions (regex^[3.2.2]).

```
[6]: # Initialize urls components:
used_cars_reg_exp = "https://.*coches-segunda-mano/.?*id=|https://.*km0/.?*id="
km0_cars_reg_exp = "https://.*km0/.?*id="

Time to obtain all the cars when reading each webpage

[7]: used_cars_urls = cars_links_generator(used_cars_pages, reg_exp=used_cars_reg_exp)
used_cars_urls[-5:]

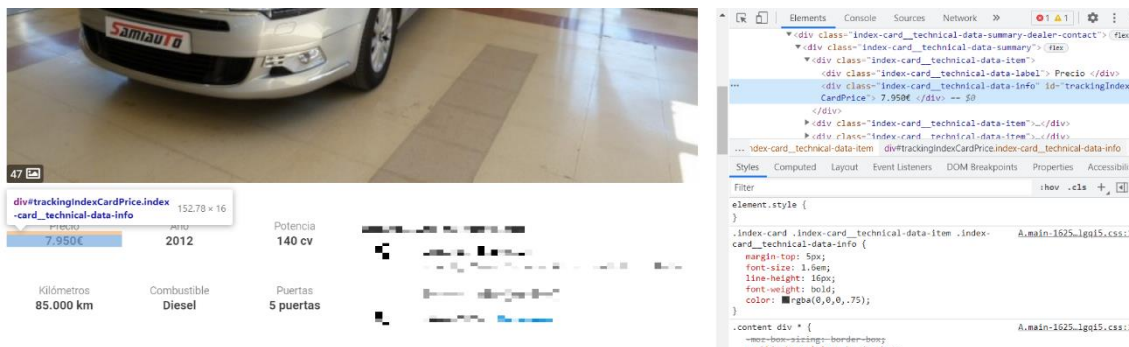
[7]: ['https://www.coches.com/km0/seminuevo-kia-sportage-16-mhev-drive-4x2-136-en-burgos.htm?id=5505020',
'https://www.coches.com/km0/seminuevo-subaru-forester-20i-hybrid-sport-plus-cvt-en-burgos.htm?id=5505080',
'https://www.coches.com/km0/seminuevo-fiat-500x-10-firefly-s_s-120th-aniversario-en-burgos.htm?id=5505094',
'https://www.coches.com/km0/seminuevo-kia-sportage-16-mhev-drive-4x2-136-en-burgos.htm?id=5505208',
'https://www.coches.com/km0/seminuevo-hyundai-ioniq-hev-16-gdi-klass-en-burgos.htm?id=5505236']
```

- *scrape_used_cars_data*: this final function scrapes each of the characteristics of each car by interpreting the XPATH [3.2.3]. The final output is a nested list, which can easily be transformed [3.2.4] into a DataFrame with pandas [3.2.5]. Main libraries used: requests, BeautifulSoup, lxml, re.

```
[31]: df_used = pd.DataFrame(used_cars_data, columns=cols_used).drop(0)
df_used.head()
```

	title	price	year	kms	city	year	doors	seats	power	color	...	length	width
1	Toyota Yaris Hsd 1.5 City	9.500€	2016	40.000 km	100 cv en Madrid	Automática continua, secuencial	5 puertas	5	100 cv	Blanco	...	389 cm	170 cm
2	Mercedes Clase A A 35 Amg 4matic+ 7a-dct	43.890€	2019	24.000 km	306 cv en Salamanca	Automática secuencial	5 puertas	5	306 cv	Blanco	...	444 cm	180 cm

The XPATH were found in each page using Google Chrome inspector (Ctrl + Shift + I):



Using this method, it was feasible to identify each data contained in each website and generate the final dataframe to be saved in a csv/txt file. However, some data cleaning was necessary before launching the bulk download, so, i.e., *strip()* method was used.

```
price = re.sub(r'\s+', ' ', dom.xpath('//*[id="trackingIndexCardPrice"]')[0].text.strip())
year = re.sub(r'\s+', ' ', dom.xpath('//*[id="trackingIndexCardYear"]')[0].text.strip())
kms = re.sub(r'\s+', ' ', dom.xpath('//*[id="trackingIndexCardKms"]')[0].text.strip())
```


3.3 Definition of Attributes

In this section, each attribute contained in the final dataset will be defined:

Attribute	Type	Definition
title	object	Car ad title. Contains a brief description of the vehicle. i.e., brand, model, equipment, city, power.
brand	object	Car brand. i.e., Ford, Seat.
model	object	Car model of the brand. i.e., Focus, Leon.
type	object	Search category associated to the vehicle. i.e., medium, small.
year	int64	Car enrolment year.
kms	int64	Kilometres of travel.
city	object	City in which the car is located.
gearbox	object	Gearbox type. i.e., Automatic, Manual.
doors	int64	Number of doors of the vehicle.
seats	int64	Number of seats of the vehicle.
power	int64	Power of the vehicle [CV].
color	object	Paint colour of car chassis.
co2_emiss	int64	CO ₂ emissions produced by the car [g/m ³].
fuel_type	object	Fuel used to propel the vehicle. i.e., Gasoline, Electric
warranty	object	Warranty of the vehicle provided by the dealer.
dealer	object	Individual or Professional dealer who sell the car.
chassis	object	Car chassis. i.e., Sedan, Coupe, Station-wagon, Van.
height	int64	Height of the vehicle [cm].
length	int64	Length of the vehicle [cm].
width	int64	Width of the vehicle [cm].
trunk_vol	int64	Car trunk volume [L].

Attribute	Type	Definition
max_speed	int64	Max speed that the car can reach [km/h].
urban_cons	float64	Urban fuel consumption of the vehicle [L/100km]. (Field removed from the final dataset) *
xtrurban_cons	float64	Extra-urban fuel consumption of the vehicle [L/100km]. (Field removed from the final dataset) *
mixed_cons	float64	Mixed fuel consumption of the vehicle [L/100km].
weight	int64	Weight of the vehicle [kg].
tank_vol	int64	Car fuel tank volume [L].
acc	float64	Car acceleration from 0-100 km/h [s].
price	int64	Car price [€]. Target variable.

* Check 4.3 Correlation of fields to better understand why these fields were removed from the final dataset.

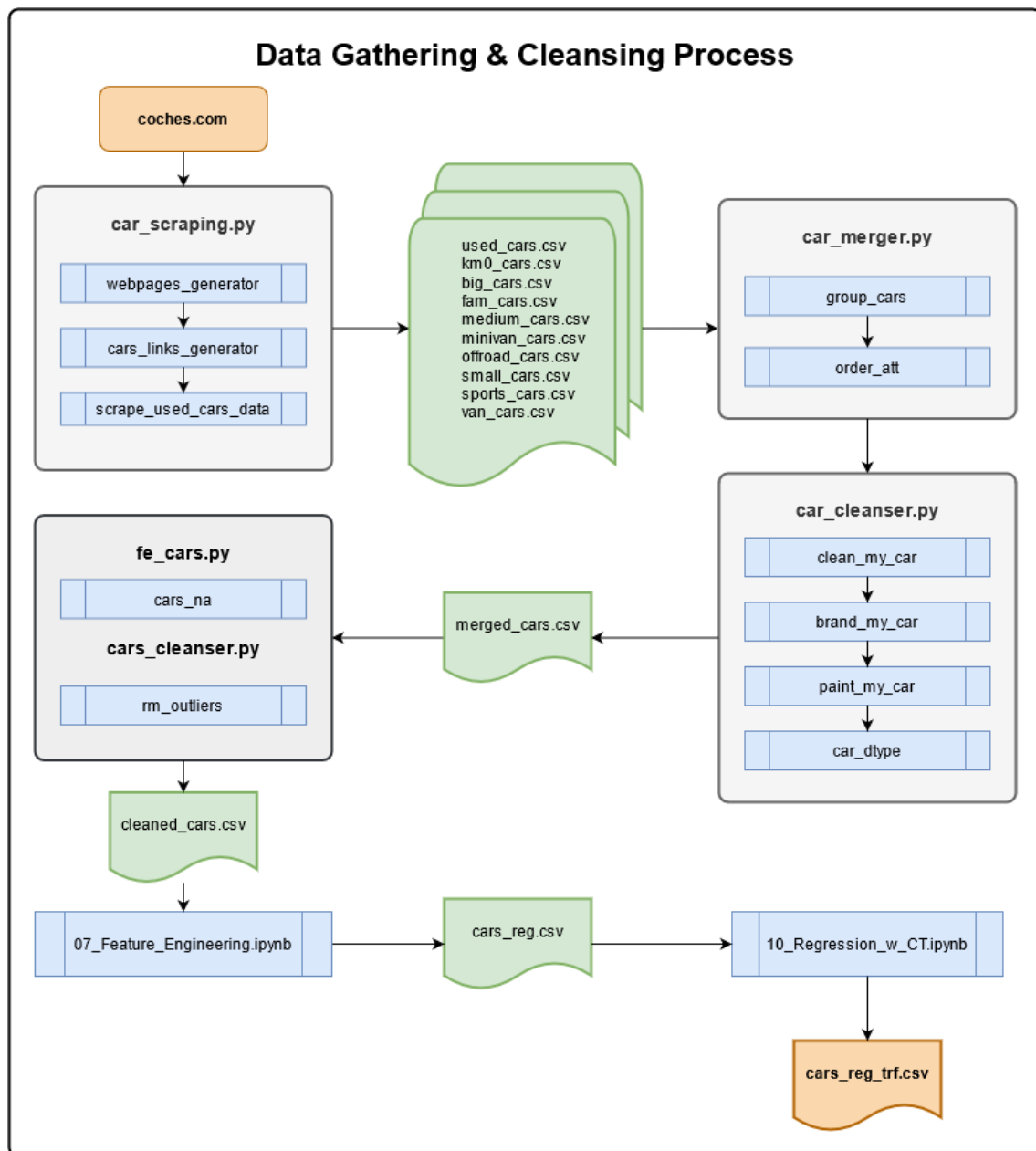
We can notice that almost half of the variables are categorical and non-numerical. At this point, they cannot feed a machine learning model to drive the final regression prediction. To this aim, these features will experiment a feature engineering process in which they will be converted into numbers in order to include new valuable variables into the algorithm.

4. Exploratory Data Analysis (EDA)

During this chapter, we will see the evolution of the dataset. How it was cleaned, detection of outliers and correlation of fields. Moreover, some insights will be exposed about how the dataset was explored to detect samples with an unusual behaviour.

4.1 Data Cleansing

The raw extracted data was not prepared to assume a machine learning algorithm. Therefore, a strategy plan was created to obtain the best possible outcome. You can consult the process below to visualize a summary of the process:



On the web scraping chapter (3.2) we saw the explanation of *car_scraping.py* module and how every csv file was obtained through this method. This step is followed by merging all the cars under the same file (*merged_cars.csv*). However, before creating this final csv, it was necessary to follow a detailed cleaning of the variables.

After applying the *car_merger.py* module, which unified all the csv into a unique dataframe, *car_cleanser.py* took action. These are some of the steps taken in each of the functions:

- *clean_my_car*: cleans all the data types for the entire cars' dataset. It executes the following tasks:
 - Numerical columns: Selecting only the int/float part of each of the columns that contains numbers. Units are also removed

```

16 # Extracting numbers from all the Quantitative columns
17 df['price'] = df.price.str.replace('.', '', regex=False).str.replace('€', '')
18 df.loc[df.year.str.len() > 4, 'year'] = df.year.str[-4:]
19 df['kms'] = df.kms.str.replace('.', '', regex=False).str.replace('km', '').str.replace('NUEVO', '0')
20 df['power'] = df.power.str.replace(' cv', '')
21 df['co2_emiss'] = df.co2_emiss.str.replace(' gr/m', '')

```

- ES>EN Translations: The source datasets are in Spanish. Translates every value to English.

```

64 # Creating a dictionary to translate the chassis
65 chassis_transl = {'Berlina': 'Sedan',
66                  'Todo Terreno': 'Offroad',
67                  'Pick-Up Doble Cabina': 'Pickup',
68                  'Chasis Doble Cabina': 'Van',
69                  'Stationwagon': 'Stationwagon',
70                  'Coupe': 'Coupe',
71                  'Furgon': 'Van',
72                  'Chasis': 'Chassis',
73                  'Monovolumen': 'Minivan',
74                  'Combi': 'Combi',
75                  'Convertible': 'Convertible',
76                  'Roadster': 'Roadster',
77                  'Pick-Up': 'Pickup',
78                  'Bus': 'Bus'
79                  }
80
81 for es, en in chassis_transl.items():
82     df['chassis'] = df.chassis.str.replace(es, en)

```

- Standardization of columns: Setting buckets to allocate all the components into a major group. i.e., for ‘gearbox’ column was simplified to ‘Manual’, ‘Automatic’ and ‘Direct’.

```

55 # Translating gearbox column
56 df = df.rename({'gear': 'gearbox'}, axis=1)
57 df['gearbox'] = df.gearbox.replace('Manual automatizada', 'Manual')\
58     .replace('Automática continua, secuencial', 'Automatic')\
59     .replace('Directo, sin caja de cambios', 'Direct')\
60     .replace('Automática secuencial', 'Automatic')\
61     .replace('Automática continua', 'Automatic')\
62     .replace('Automática', 'Automatic')

```

- *brand_my_car*: creates ‘brand’ and ‘model’ features for the model. the input field is ‘title’. It deals with exceptions to obtain the brand/model for every car. To this aim, regex is used to produce the desired output.

```

109 # Conditional lists
110 long_brands = ['Land Rover', 'Aston Martin', 'Alfa Romeo', 'Mercedes Amg']
111 long_models = ['Coupé', 'Serie', 'Clase', 'Grand', 'Grande', 'Santa', 'Glory', 'Model', 'Xsara', 'Pt',
112               'Is', 'Es', 'Ct', 'Rx', 'Nx', 'Ux', 'Rc', 'Gs', 'Ls'] # Lexus
113
114 df['title'] = df['title'].str.replace(r'\b(\w+)\s+(\w+)\b', r'\1', regex=True)
115
116 # Excep: Volkswagen Up contains '!'
117 df['title'] = df.title.str.replace('!', '')
118
119 # Excep: E Advance
120 df['title'] = df.title.str.replace('E Honda', 'E-Advance')
121
122 # Splitting columns to conform both brands & models
123 df[['brand', 'aux_1', 'aux_2', 'aux_3']] = df.title.str.split(' ', 3, expand=True)
124
125 # Excep: Ds
126 df['brand'] = df.brand.str.replace('Ds', 'Citroen')
127
128 # Aux columns
129 df['brand_long'] = df.brand + ' ' + df.aux_1
130 df['model_long'] = df.aux_1 + ' ' + df.aux_2
131 df['model'] = df['aux_1']

```

- *paint_my_car*: deals with 'color' column to create a paint palette. Cleans the dataset to obtain only general colours. It also translates the names from Spanish to English.

```

162 list_numbers = [str(x) for x in list(range(0, 10))]
163 dict_colors = {'BLANCO': 'WHITE', 'BLANC': 'WHITE', 'BIANCO': 'BLANCO', 'ALPINWEISS': 'WHITE',
164               'GRIS': 'GREY', 'GRAY': 'GREY',
165               'NEGRO': 'BLACK',
166               'AZUL': 'BLUE',
167               'ROJO': 'RED', 'GRANATE': 'RED', 'BURDEOS': 'RED',
168               'PLATA': 'SILVER', 'PLATEADO': 'SILVER',
169               'MARRÓN': 'BROWN', 'MARRON': 'BROWN',
170               'VERDE': 'GREEN',
171               'BEIGE': 'BEIGE',
172               'AZUL MARINO': 'NAVY BLUE',
173               'NARANJA': 'ORANGE',
174               'AMARILLO': 'YELLOW',
175               'BRONCE': 'BRONZE',
176               'VIOLETA': 'PURPLE', 'MORADO': 'PURPLE',
177               'ROSA': 'PINK',
178               'OTRO': 'OTHER'
179             }
180
181 colors = sorted(df.color.dropna().unique())
182 colors_up = [c.upper() for c in colors]
183
184 # Remove numbers from colors
185 df.loc[df.color.isin(list_numbers), 'color'] = 'OTHER'
186
187 # Assigning 'OTHER' to empty colors
188 df.loc[df.color.isna(), 'color'] = 'OTHER'

```

- *car_dtype*: changes the types of attributes of the input dataframe.

Once this process is completed, the merged_cars.csv is obtained. The only remaining step to reach the cleaned dataset is the detection of outliers, which will be exposed in the next point.

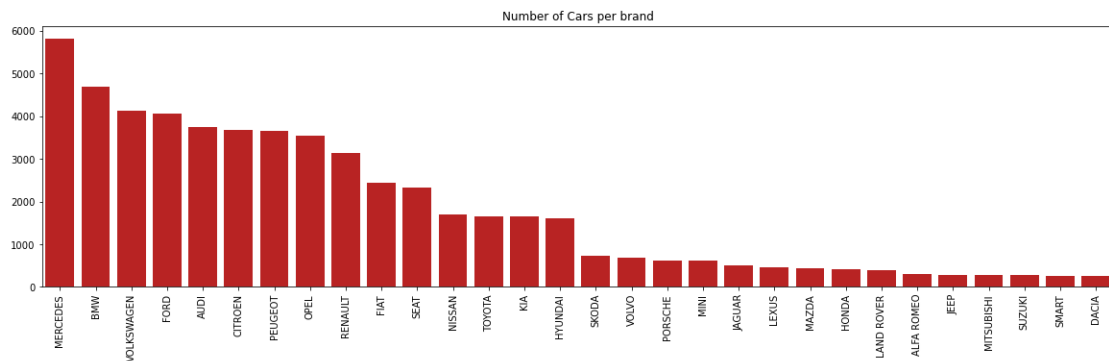
Through this method, the size of the dataset was significantly decreased from 66,533 samples to 55,973. In this way, undesirable values were removed from the initial data extraction, such as duplicated rows.

The rest of the cleaning process will be exposed in the next chapters until we reach to the final dataset used for the CatBoost regression algorithm, cars_reg_trf.csv.

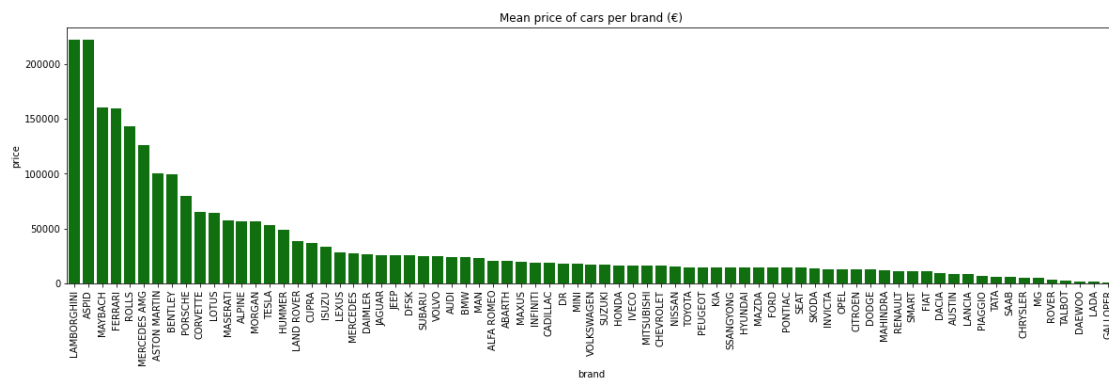
4.2 Exploration and definition of outliers

Now that the data is apparently cleaning and merged, it is time to explore the dataset. Obviously, the machine learning model will not be as well adapted to every car. These algorithms are based on the training data that they use, so the more the amount and diversity of data, the finest the prediction.

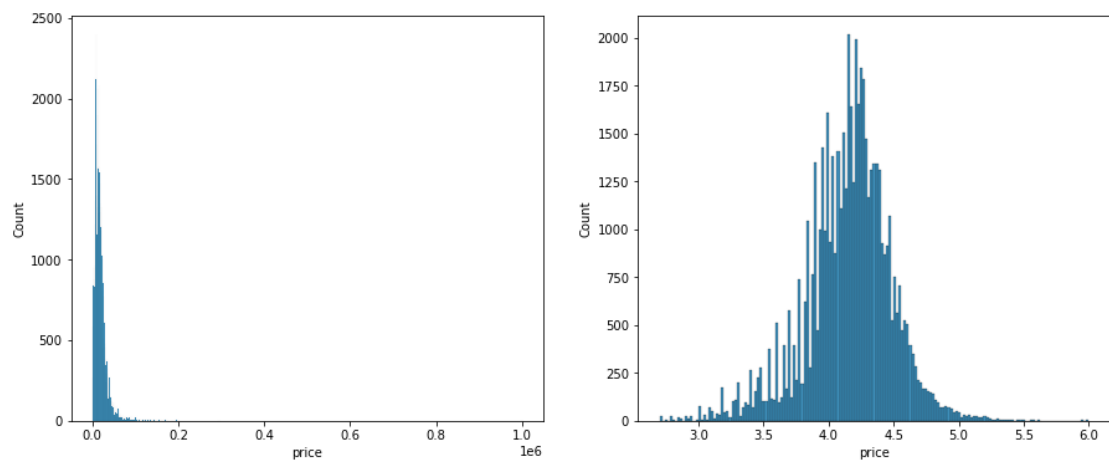
The second-hand car market is full of brands and models, but we can have a look at the predominant ones. Mercedes, BMW and Volkswagen are the top three selling brands in the source car portal. This means that the model will adjust better to the prediction of a Mercedes (~5900 samples) than a Dacia (~200 samples). In any case, it could happen that 200 samples is enough to do a suitable prediction, taking into account that 'brand' is not the only feature that conforms the training data.



The target value of the CatBoost model will be the 'price' of the car. The value of some brands is really high compared to others, producing a range in the initial dataset from less than 1,000€ to more than 200,000€. This could produce a huge bias to high prices for these brands, even if one specific Lamborghini or Ferrari is in bad condition and the price should decrease. In any case, as it was mentioned before, the rest of attributes should compensate the behaviour of the model.

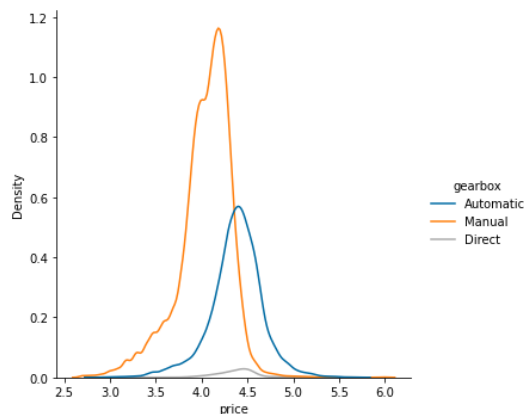


The 'price' target caused more problems related to outliers. Some cars had a wrongly input for price, having values higher than 4,000,000€, which was not possible at all for these models. Therefore, the initial car price was capped to this price. Its distribution was right-skewed, so together with a logarithmic scale, helped to visualize the distribution of prices:



Including a second field to evaluate the price, helped on making a better data-driven decision. Thereupon, 'gearbox' types helped to visualize price distribution:

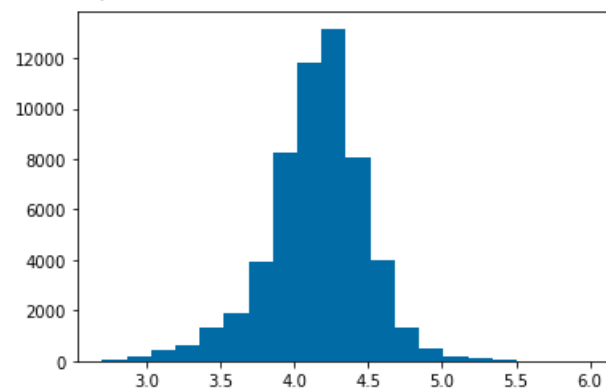
Most repeated value for Automatic gearbox: 4.28
 Most repeated value for Manual gearbox: 4.23
 Most repeated value for Direct gearbox: 4.02



From the previous representations, the following conclusions are obtained from the dataset:

- The most common used cars have a **Manual gearbox**.
- Cars with an **Automatic gearbox** are on average $10^{4.28} - 10^{4.23} \approx \$2,072$ more expensive than cars with **Manual gearbox**.

Most repeated value for: 4.18



Therefore, the most common price for a used car in the dataset is about:

$$10^{4.18} \approx \$15,136$$

Above we can see that the most common price for a car in the dataset was of 15,136€. It was also possible to detect that a car with an automatic gearbox is about 2,072€ more expensive than the ones with a manual gearbox, which makes sense according to the market.

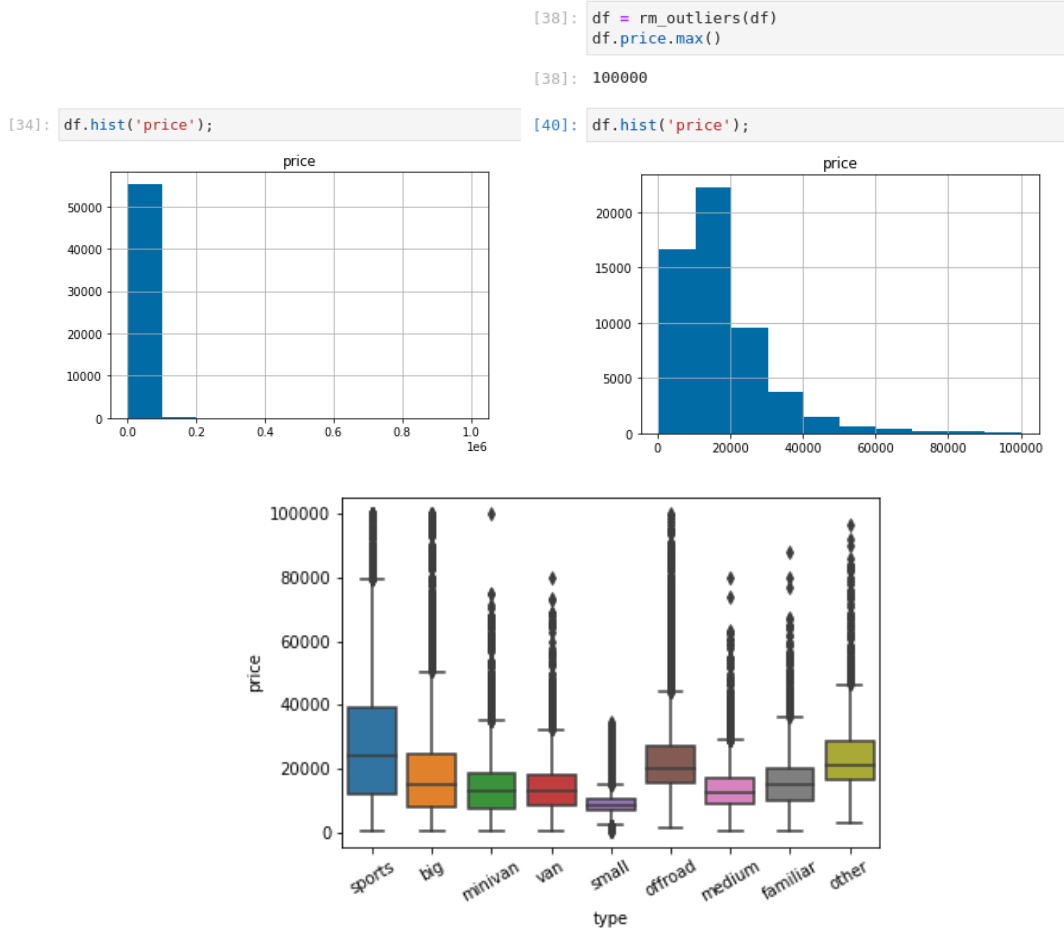
The inclusion of a new variable versus the target price, added more value to detect outliers [4.2.1]. The number of cars with a strange price increased with the enrolment year of the car. In cases in which they were too old, historical vehicles emerged, so the price was either incorrect or too high to be consumed and interpreted by the final algorithm:

```
[37]: df[df.price == df.price.max()]
```

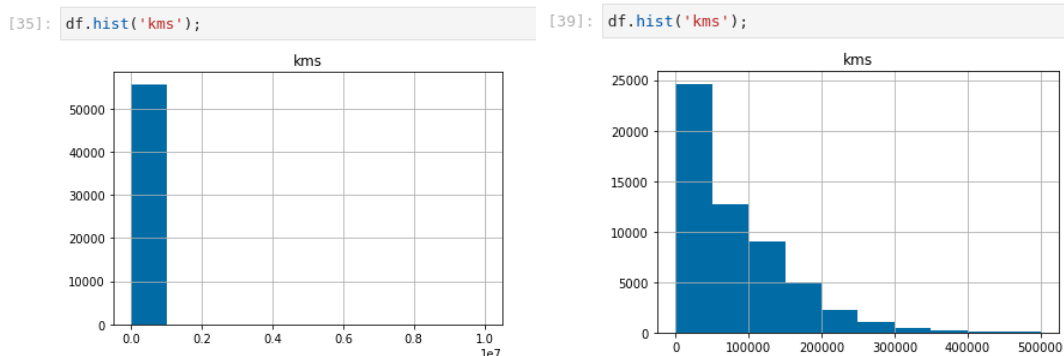
	title	brand	model	type	year	kms	city	gearbox	doors	seats	...	height	length	width	trunk_vol	max_speed	mixed_cons	weight	tank_vol	acc	price
2946	Porsche 911 Carrera Rs	PORSCHE	911	sports	1973	50000	Barcelona	Manual	2	4	...	127	425	174	126	299	10.6	1270	65	5.0	999999
3202	Ferrari Testarossa	FERRARI	TESTAROSSA	sports	1986	59000	Barcelona	Manual	2	2	...	125	458	193	247	324	15.2	1505	88	3.7	999999
3993	Porsche 911 Carrera Rs	PORSCHE	911	sports	1973	50000	Barcelona	Manual	2	4	...	127	425	174	126	299	10.6	1270	65	5.0	999999
5070	Porsche 911 Carrera Rs	PORSCHE	911	sports	1995	136000	Barcelona	Manual	2	4	...	127	425	174	126	299	10.6	1270	65	5.0	999999

In this screenshot, we can see some Porsches from 1973 or 1986, which are undesirable to be included in the final model.

As a final decision, it was decided to exclude historical and luxury cars by capping the price to 100,000€. This was included in the `rm_outliers()` function inside the `car_cleanser.py` module. Once this is applied, it is possible to visualize prices without applying a log scale:



Outliers were not only causing issues on prices, but also on the total travel kilometres of cars. Some samples have an exorbitant number of kilometres, so this value was also capped to 500,000 km. This maximum number was based on the maximum lifespan of a car engine ^[4.2.2]. In this article it is mentioned that they could last up to 300,000 miles on electric cars, which are about 482,803 kms.



Removing outliers helped on narrowing down the scope of the machine learning model. This helped to reduce some bias to wrong conclusions and to improve the final R^2 score metric of the algorithm. For further details, please check 5.3 chapter (Data Model).

Finally, the *fe_cars.py* module (*cars_na*) was created to deal with some missing (NaN - Not a Number) or zero values. The objective was to drop missing values and replace zeros with some business sense. After exploring the variables, the main focus was placed on 'kms' and 'co2_emiss' attributes.

It is well-known that some second-hand cars in the market are sold as 0-km. This significantly reduces the price of a new car. Therefore, *NaN* values under this field were replaced by zero.

According to CO₂ emissions, electric cars recently entered to the Spanish market. These cars do not produce any emissions, so the dataset also accepted zero values under this field and *NaN* values were replaced.

For the rest of columns, 'max_speed', 'height', 'length', 'width', 'trunk_vol', 'urban_cons', 'xtrurban_cons', 'mixed_cons', 'weight', 'tank_vol', 'acc', the missing values were replaced by the mean of each brand/model combination in the dataset. Only for those cars with an enrolment year previous than 2018 and CO₂ emissions equal to zero, the mean was used.

Zeros are only valid on 'co2_emissions' & 'kms' columns

```
[21]: # Defining columns with NaN value
dropna_subset = df.columns[df.isna().any()].drop(['kms', 'co2_emiss'])
dropna_subset

[21]: Index(['height', 'length', 'width', 'trunk_vol', 'max_speed', 'urban_cons',
          'xtrurban_cons', 'mixed_cons', 'tank_vol', 'acc'],
          dtype='object')

[22]: df.shape

[22]: (55972, 29)

[23]: df = df.dropna(subset=dropna_subset)
df.shape

[23]: (55722, 29)
```

The remaining samples with missing values, in columns different from 'kms' and 'co2_emiss', were dropped. This produced a reduction of 250 samples in the final dataset. This process was applied just right before dropping the outliers from 'kms' and 'price' columns, so the final number of samples was reduced to 55,326 used cars.

What is more, the analysis of *NaN* values helped to not drop useful fields since 'trunk_vol' column contained 8,986 *NaN* rows. In case it was decided to just drop *NaN* values for every field and column, this information would have been lost. Therefore, this allowed to save 8,736 samples to enrich the model.

4.3 Correlation of fields

From the described variables on point 3.3, it is important to detect if there is any strong linear correlation ^[4.3.1] between any of them. This can help to detect if a group of variables is just adding redundant information to the model. In this case, we can remove these ones since they will not add too much value to the final outcome. Therefore, the

aim of this analysis is to reduce the dimensionality of the dataset, which eventually will enhance the performance and speed of the machine learning algorithm.

Firstly, high correlation values will be defined so as to not cause any confusion. The Pearson's Correlation Coefficient (r)^[4.3.2] was the metric used to determine the strength of the correlation, which is the standard method of pandas `corr()`^[4.3.3]. There is a rule of thumb^[4.3.4] for interpreting the strength of a relationship based on its absolute r value:

Absolute Value of r

$r < 0.3$

$0.3 < r < 0.5$

$0.5 < r < 0.7$

$r > 0.7$

Strength of Relationship

None or very weak

Weak

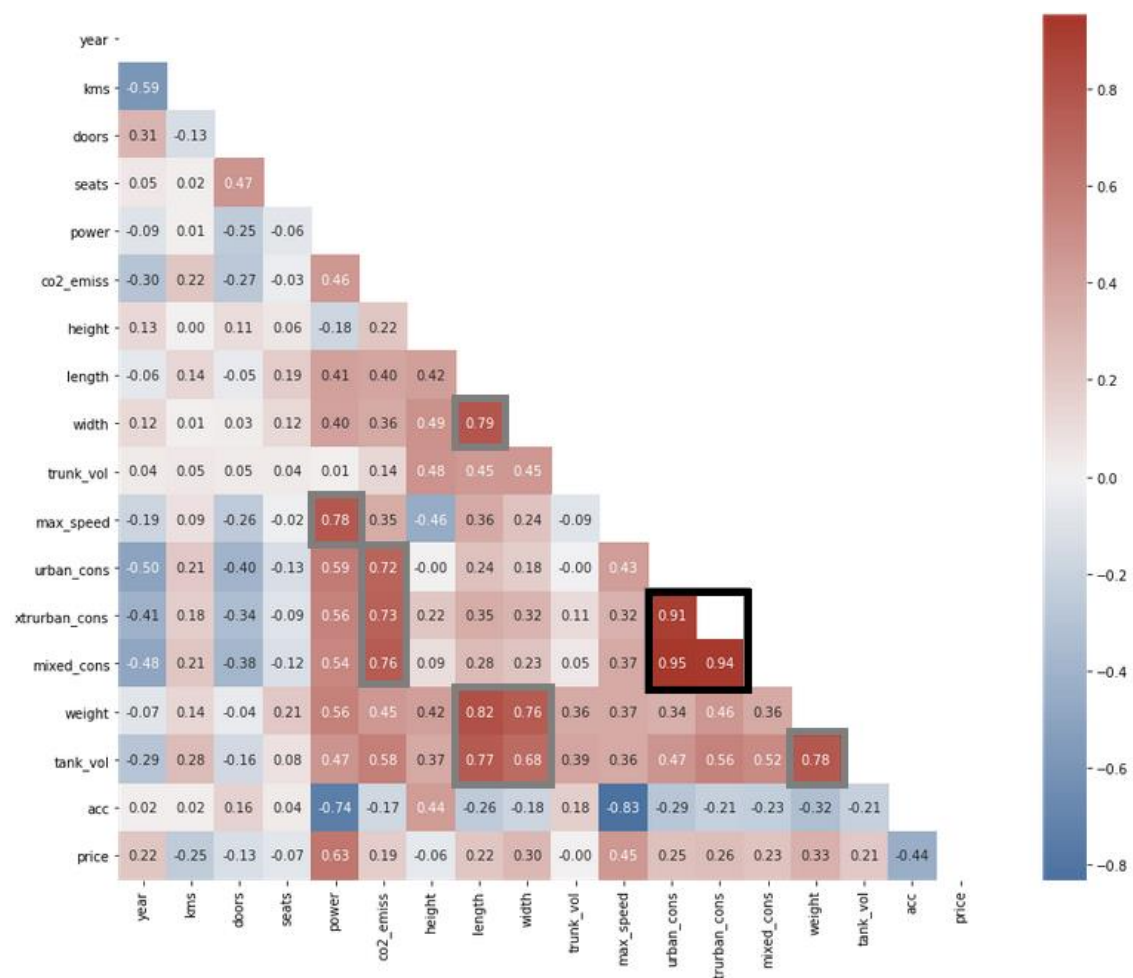
Moderate

Strong

The perfect linear relationship will only occur when $r = \pm 1$. We can include an additional "Very Strong" category for those with a correlation higher than 0.7.

Very Strong : $r \in [0.9, 1]$

To visualize these relationships, `low_corr_matrix()` was created inside `plotter.py` module. It plots the lower half of a correlation matrix in a nice format, based on a heatmap:



From this representation, we can take the following assumptions:

- There is a very strong positive correlation ($r > 0.9$) between 'mixed_cons', 'xtrurban_cons' and 'urban_cons'.
- There is a strong positive correlation in the next pairs of variables:
 - *power* - *max_speed*: it is reasonable that the higher the power of the vehicle, the higher the maximum speed of the car.
 - *co2_emiss* - *mixed_cons*, *urban_cons*, *xtrurban_cons*: it makes sense that a higher consumption of a car can lead to more CO² emissions.
 - *length* - *width*: the size of a big car relies on both its length and width.
 - *weight* - *tank_vol*: if the car fuel tank is bigger, it leads to a higher car weight.
 - *weight*, *tank_vol* - *length*, *width*: the dimensions of a car are strongly correlated with the weight and fuel volume, which is not a surprise.

These insights provided by the correlation matrix are really useful to better understand the relation between the different variables that conform the dataset. Moreover, a very strong positive correlation, leads to conclude that these variables are adding the same value to the model, so we can take just one of them to reduce the dimensionality.

These consumption variables, 'mixed_cons', 'xtrurban_cons' and 'urban_cons', define the same aspect of the vehicle. Therefore, it was decided to take only the one that will potentially add more value, which is 'mixed_cons'. The reason is that this feature includes the other two, so it will be a simplification of this technical specification. The deletion of these variables helped on the stability of the model since a high correlation leads to a model very sensible to this data [4.3.5].

Finally, this dataset is ready to be used as an input of a machine learning model. This analysis made possible to reduce the number of columns in the model from 29 to 27.

[4]: df

		title	brand	model	type	year	kms	city	gearbox	doors	seats	...	height	length	width	trunk_vol	max_speed	mixed_cons	weight	tank_vol	acc	price
0		Smart Fortwo Cabrio 52 Mhd Pure Aut.	SMART	FORTWO	sports	2014	37125	Alicante	Automatic	2	2	...	157	270	156	203	145	4.3	780	33	13.7	5500
1		Volkswagen Scirocco 2.0 Tsi	VOLKSWAGEN	SCIROCCO	sports	2009	0	Barcelona	Manual	3	4	...	140	426	181	292	235	7.6	1373	55	7.2	10900
2		Bmw Serie 2 218da Gran Coupé	BMW	SERIE 2	sports	2021	0	Cantabria	Automatic	4	5	...	142	453	180	430	222	4.2	1545	42	8.5	36100
3		Bmw X4 Xdrive 20d	BMW	X4	sports	2017	85000	Girona	Automatic	5	5	...	162	467	188	500	212	5.4	1740	67	8.0	28000
4		Dodge Viper Srt-10	DODGE	VIPER	sports	2005	95017	Unknown	Automatic	2	2	...	123	446	191	344	189	21.0	1546	70	3.9	27990
...	
55321		Opel Corsa 1.3cdti Selective 75	OPEL	CORSA	other	2016	60000	Madrid	Manual	5	5	...	148	402	175	285	164	3.8	1237	45	14.8	7500
55322		Mercedes Clase V 250d Largo Avantgarde 7g Tronic	MERCEDES	CLASE V	other	2016	70261	Unknown	Automatic	5	6	...	188	514	193	1030	206	6.0	2145	67	9.1	48500
55323		Jeep Cherokee 2.0d Longitude 4x2 103kw	JEEP	CHEROKEE	other	2015	182069	Ciudad Real	Manual	5	5	...	163	462	186	412	187	5.3	1828	60	10.9	15999
55324		Bmw X3 Xdrive 30da	BMW	X3	other	2011	159000	Barcelona	Automatic	5	5	...	186	465	188	550	230	6.0	1800	67	6.2	18300
55325		Citroen C4 Aircross 1.6hdi S&S Exclusive Plus ...	CITROEN	C4	other	2016	78000	Sevilla	Manual	5	5	...	163	434	180	419	180	5.0	1500	60	11.6	15500

55326 rows x 27 columns

5. Methodology

Along this chapter, it will be covered the whole methodology applied to obtain a model to be used in production from the initial cleaned dataset. The initial section will start with the basic creation of an MVP. Later we will see the feature engineering techniques

needed to extract numbers from our categorical features. Several algorithms will be tested to evaluate their performance using this data. Over and underfitting will be treated by applying CV methods as well as to find the best hyperparameters. Finally, the optimal model will be evaluated and saved using the pickle library.

5.1 MVP: Setting a starting point

The objective of this Minimum Viable Product (MVP) is to reproduce the project in just one notebook to set a starting point, the first iteration. It consists on the following steps: importing the data, data visualization, data cleansing, data modelling and model evaluation.

Firstly, the previously created cleaned dataset is loaded:

Import Data

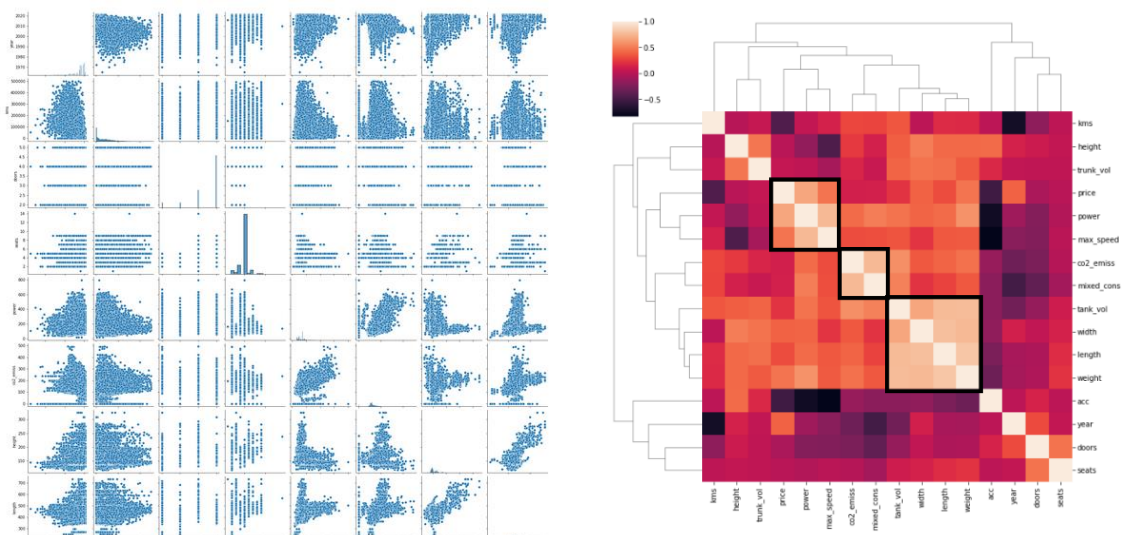
```
[2]: root = '../data/cleaned_cars.csv'
```

```
[3]: df = pd.read_csv(root)
df.head()
```

```
[3]:
```

	title	brand	model	type	year	kms	city	gearbox	doors	seats	...	height	length	width	trunk_vol	max_speed	mixed_cons	weight	tank_vol	acc	price
0	Smart Fortwo Cabrio 52 Mhd Pure Aut.	SMART	FORTWO	sports	2014	37125	Alicante	Automatic	2	2	...	157	270	156	203	145	4.3	780	33	13.7	5500
1	Volkswagen Scirocco 2.0 Tsi	VOLKSWAGEN	SCIROCCO	sports	2009	0	Barcelona	Manual	3	4	...	140	426	181	292	235	7.6	1373	55	7.2	10900
2	Bmw Serie 2 218da Gran Coupé	BMW	SERIE 2	sports	2021	0	Cantabria	Automatic	4	5	...	142	453	180	430	222	4.2	1545	42	8.5	36100
3	Bmw X4 Xdrive 20d	BMW	X4	sports	2017	85000	Girona	Automatic	5	5	...	162	467	188	500	212	5.4	1740	67	8.0	28000
4	Dodge Viper Srt-10	DODGE	VIPER	sports	2005	95017	Unknown	Automatic	2	2	...	123	446	191	344	189	21.0	1546	70	3.9	27990

For the data visualization step, the most worth it approach is to have a quick view at the relation between variables. Thereupon, a pairplot ^[5.1.1] or a clustermap ^[5.1.2] can be the best approach to visualize this interaction among features:



On the one hand, from the pairplot, we can obtain several conclusions about the variables:

- There is a positive linear regression between: power - co2_emiss, mixed_cons - co2_emiss, length - height, width - length, weight - tank_vol. This matches with the strong positive correlations found on the Pearson's Correlation Analysis.

- There is a clear inverse exponential relation ^[5.1.3] between acc - power. The relation is noticeable between: acc - mixed_cons, acc - max_speed.

On the other hand, we can visualize 3 clear clusters under the clustermap:

1. price - power - max_speed: The relation between these variables is strong, but not determinant.
2. co2_emiss - mixed_cons: again, car consumption is strongly related to the CO² emissions.
3. tank_vol - width - length - weight: car dimensions conform a clear cluster. These variables are the ones with the closest relation, which is a preconceived assumption.

During the development of this MVP, kilometres and price outliers, previously analysed, were discovered in the dataset. This step was important to interpret which values make sense within the dataset.

Moving to the data modelling phase, the scikit-learn library ^[5.1.4] enters into the scene. The data is divided into features and target ('price'). Then into train and test data to train the model and make a prediction. The initial regression model used was a Linear Regression. As this is an MVP, I did not apply any pre-processing method to categorical features, so I did a first approximation based on numerical attributes only.

```
[11]: # Dividing the data into target and features
features = df[df.select_dtypes('number').columns[:-1]]
target = df[df.columns[-1]] # Price

[12]: X_train, X_test, y_train, y_test = train_test_split(features, target)

lr = LinearRegression()
lr.fit(X_train, y_train)

[12]: LinearRegression()

[13]: predictions = lr.predict(X_test)
```

Finally, the evaluation of the model was performed on top of a prepared module: *model_evaluator.py*. The *eval_reg()* function takes the test target (*y_test*) and the predictions produced by the model. Then, it generates the evaluation of the model based on several metrics: R² score, RMSE, MSE, MAE, EVS. Our focus will be on the R² score since it was the final evaluation metric used for this project.

```
[20]: from modules.model_evaluator import eval_reg

[21]: eval_reg(y_test, predictions)

### MEASURES OF REGRESSION MODEL ###
-----

R2 = 0.7448

RMSE = 6590.4674

MSE = 43434260.2006

MSLE not possible to be applied. Predicitons contain negative values.

MAE = 4287.8659

EVS = 74.4809%
```

The result was of R² = 0.7448, which is really promising taking into account that is an MVP and that the maximum value that can be reached is 1. This was only by applying a

Linear Regressor, so on this chapter the pre-processing techniques, together with other ML algorithms, will be explain to obtain a final feasible model with a great performance.

5.2 Feature Engineering

Feature engineering ^[5.2.1] is the process of extracting features from raw data by using domain knowledge. This result in improved model accuracy on unseen data. The base features used will directly influence the predictive model and the obtained results. Therefore, the better the features are prepared, the better the results, although this will also rely on the behaviour of the final chosen model.

The feature engineering steps applied to this project are mainly focused on the categorical variables, which are extensive. To this aim, we will start talking about One-Hot and Target Encoding. Afterwards, the numerical features will be explored so as to decide the best Column Transformation for this use case.

5.2.1 One-Hot Encode: Low cardinality variables

The first step in Feature Engineering is to define and separate the 26 variables into numerical and categorical. Once divided, the focus will start on top of the categorical features. Numerical values can be directly input and consumed by an algorithm, but not objects or strings. This is when it is necessary to use One-Hot Encoding ^[5.2.2].

Numerical (14)	width, year, height, seats, co2_emiss, mixed_cons, weight, tank_vol, power, acc, length, trunk_vol, kms, max_speed, seats
Categorical (12)	model, chassis, warranty, city, brand, color, fuel_type, gearbox, dealer, type, doors

One-Hot is used to codify variables with label values rather than numeric values. It is effective for those cases in which there is no ordinal relationship between values. It creates a new binary variable (0, 1) per each unique integer value. As an example, we can see how to apply One-Hot to a categorical feature that consists on colours:

Color		Red	Yellow	Green
Red				
Red	→	1	0	0
Yellow		1	0	0
Green		0	1	0
Yellow		0	0	1

The binary features are also called in statistics “dummy variables” since their only purpose is to act as a number to fit into the model. The library of pandas contains a *get_dummies()* method ^[5.2.3], which is a simpler solution to apply this feature engineering step to a dataset.

```
>>> s = pd.Series(list('abca'))
```

```
>>> pd.get_dummies(s)
   a  b  c
0  1  0  0
1  0  1  0
2  0  0  1
3  1  0  0
```

Since One-Hot Encoding produces one column per value in each feature, it drastically increases the dimensionality of a data structure. This is why it was decided to just apply this feature engineering transformation to low cardinality variables. In this way, the performance of the model will not be significantly impacted and there will not be plenty of variables with a low adding value to the final outcome. The chosen features from the 12 categorical ones in the model were: type, gearbox, fuel_type, warranty, dealer, doors.

As we can notice, not all the categorical variables are nominal, which means that not all of them only contain strings or labels. Categorical features can also be numerical, so they are especially easy to detect when they are referring to cardinality. However, how can we detect which non-ordinal numerical variables are categorical in this dataset? Well, a categorical variable is a variable that assumes only a limited number of discrete values ^[5.2.4]. Therefore, this rule is met for one of our features: 'doors'. It is discrete as it can only assume a range of values. It is not only that they can just be integers, a car cannot have 12 or 30 doors. This is why it was decided to integrate it as categorical feature into the final model.

Finally, we can visualize how the implementation of One-Hot Encoding changed the features in the dataset:

Before

```
[7]: ohe = OneHotEncoder(categories='auto')
```

```
[8]: df.select_dtypes(exclude=["number"])
```

[8]:		title	brand	model	type	city	gearbox	color	fuel_type	warranty	dealer
0		Smart Fortwo Cabrio 52 Mhd Pure Aut.	SMART	FORTWO	sports	Alicante	Automatic	WHITE	Gasoline	YES	Professional
1		Volkswagen Scirocco 2.0 Tsi	VOLKSWAGEN	SCIROCCO	sports	Barcelona	Manual	WHITE	Gasoline	YES	Professional
2		Bmw Serie 2 218da Gran Coupé	BMW	SERIE 2	sports	Cantabria	Automatic	OTHER	Diesel	YES	Professional
3		Bmw X4 Xdrive 20d	BMW	X4	sports	Girona	Automatic	WHITE	Diesel	YES	Professional
4		Dodge Viper Srt-10	DODGE	VIPER	sports	Unknown	Automatic	RED	Gasoline	YES	Professional

After

[11]:		gearbox_Automatic	gearbox_Direct	gearbox_Manual	fuel_type_Diesel	fuel_type_Electric	fuel_type_Gasoline	fuel_type_Hybrid	warranty_NO	warranty_YES
0		1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0
1		0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
2		1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3		1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
4		1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0

5.2.2 Target Encode: high cardinality variables

Some features can have a lot of different values, so applying One-Hot to them will result on creating a ridiculous amount of dummy variables. To transform these columns into numerical values, Target Encoding ^[5.2.5] is the best option.

These variables without a natural numerical representation, need to be transformed, even if they have a high cardinality. Target Encoding presents a solution, which consists on applying an average of the target by each category. In this way, the values that are picked up can explain the target. Having a look at a simple example, it is easier to understand the logic behind this method:

Target Encoding

	Animal	Target	Encoded Animal
0	cat	1	0.40
1	hamster	0	0.50
2	cat	0	0.40
3	cat	1	0.40
4	dog	1	0.67
5	hamster	1	0.50
6	cat	0	0.40
7	dog	1	0.67
8	cat	0	0.40
9	dog	0	0.67

Table 1: Dataframe With Target Encoded Animal Values



	Animal Group	Target 0	Target 1	Probability of 1
0	cat	3	2	0.40
1	dog	1	2	0.67
2	hamster	1	1	0.50

Table 2: Simplified Table to Show how Target Encoding is Calculating the Probability

Consider that X are features (*Animal*), while y is the target. Encoded values are calculated using a probability of occurrence, as detailed on the table above:

1. Group the data by each category and count the number of occurrences of each target (Table 2).
2. Calculate the probability of Target 1 occurring given each specific 'Animal Group.' After doing this, we get the values in Table 2.
3. Finally, add back in the new column, which gives the probability value of each *Animal Group*, which is shown in the first dataframe (Table 1). Now we have a numerical value for 'Animal' feature that can be recognized by machine learning algorithms.

The potential benefit of using Target Encoding vs One-Hot Encoding ^[5.2.6] is that is a simple and quick encoding method, while it does not increase the dimensionality of the dataset. However, this method is dependent on the distribution of the target, so it can be prone to overfitting. The way this problem will be mitigated is through applying cross-validation techniques. Thereupon, the model will not be biased by the effect of this feature engineering transformation.

On this project dataset, the encoded features were: brand, model, city, color, type, chassis. All these variables count on a high cardinality (+14 values), so this is why it was decided to apply a Target Encoder to them. This was the transformation that the dataframe suffered after being exposed to this machine learning mechanism:

```
[17]: from category_encoders import TargetEncoder
```

```
[18]: data.select_dtypes(exclude=["number"])
```

```
[18]:
```

	brand	model	type	city	color	chassis
0	SMART	FORTWO	sports	Alicante	WHITE	Convertible
1	VOLKSWAGEN	SCIROCCO	sports	Barcelona	WHITE	Coupe
2	BMW	SERIE 2	sports	Cantabria	OTHER	Coupe
3	BMW	X4	sports	Girona	WHITE	Coupe
4	DODGE	VIPER	sports	Unknown	RED	Coupe
...
55321	OPEL	CORSA	other	Madrid	WHITE	Sedan
55322	MERCEDES	CLASE V	other	Unknown	BLACK	Minivan
55323	JEEP	CHEROKEE	other	Ciudad Real	GREY	Offroad
55324	BMW	X3	other	Barcelona	WHITE	Offroad
55325	CITROEN	C4	other	Sevilla	GREY	Offroad

```
[22]: data[cols_encoded]
```

```
[22]:
```

	brand_encoded	model_encoded	city_encoded	color_encoded	type_encoded	chassis_encoded
0	11039.980916	10171.125628	15841.032810	17003.852965	28170.085203	20696.065144
1	17058.096993	13160.993789	18052.176748	17003.852965	28170.085203	31261.850521
2	23593.497861	23508.383973	18554.493671	21242.844857	28170.085203	31261.850521
3	23593.497861	36412.589552	16569.441935	17003.852965	28170.085203	31261.850521
4	12506.369072	17783.891805	17303.956434	17436.456262	28170.085203	31261.850521
...
55321	12850.655659	11980.345216	18965.842225	17003.852965	24125.953305	15131.850727
55322	26831.477053	40573.610879	17303.956434	19080.509292	24125.953305	14177.381352
55323	25865.686411	28652.349939	15706.924710	17975.461741	24125.953305	23421.348833
55324	23593.497861	29654.348315	18052.176748	17003.852965	24125.953305	23421.348833
55325	12661.108152	11178.258242	17534.547549	17975.461741	24125.953305	23421.348833

Now that all the variables have been transformed into numbers, it is time to analyse them before proceeding with further standardization processes.

5.2.3 Exploring numerical features

During the Exploratory Data Analysis phase, numerical features were properly cleaned. They do not present any *NaN* values, which was the most critical part to not feed the model with an inconsistent dataset.

After the pre-processing stage, each numerical feature has a continuous distribution, even the categorical ones after applying both One-Hot and Target Encoding. Therefore, they will collaborate to enrich the model in an efficient way.

Once a machine learning algorithm is applied, it will interpret each feature, evaluating the weight of each of them into the final predicted value. The importance of each variable is determined by their influence on changing the target value. This consists on an interrelation between features, which will finally conform the prediction.

```
[34]:
```

	year	kms	doors	seats	power	co2_emiss	height	length	width	trunk_vol	max_speed	mixed_cons	weight	tank_vol	acc
0	2014	37125	2	2	71	99	157	270	156	203	145	4.3	780	33	13.7
1	2009	0	3	4	200	179	140	426	181	292	235	7.6	1373	55	7.2
2	2021	0	4	5	150	109	142	453	180	430	222	4.2	1545	42	8.5
3	2017	85000	5	5	190	142	162	467	188	500	212	5.4	1740	67	8.0
4	2005	95017	2	2	506	488	123	446	191	344	189	21.0	1546	70	3.9

These values will be directly used to produce the first iteration on analysing the best model for this use case. Nevertheless, before exploring these methodologies, we will cover the normalization of these numerical values to enhance the performance of machine learning algorithms during the next section.

The final dataset obtained in this chapter, was saved as *cars_reg.csv*, consult point 4.1 (Data Cleansing) for a better understanding on the data evolution of this project.

5.2.4 Column Transformation: Features Normalization

Many of machine learning models perform better when the distribution of variables is Gaussian ^[5.2.7]. Some algorithms like linear or logistic regression assume that the features are already normally distributed. Others such as Boosting and Ensemble learning algorithms could not have this strong assumption, but it is proved that models perform better when based on Gaussian distributed variables.

To this aim, Power Transformations ^[5.2.8] can be really convenient. They make the probability of a distribution more Gaussian by removing a skew (right or left) in the distribution, stabilizing the variance of the distribution. Although a first approach could be to apply either a log or square root to each variable, they could result to not be the best transformation for this use case.

Power Transformations use a generalized transformation that look for the best parameter (lambda) to adapt the distribution of a feature to a Gaussian probability. The most extended automatic transformation methods are ^[5.2.9]:

- Box-Cox: only works with strictly positive values.
- Yeo-Johnson: works with positive, negative and zero values.

Moreover, in this analysis it was included Quantile Transforms ^[5.2.10]. This transformation method is more powerful since it allows to map a feature's distribution to another probability distribution. In this case, the output distribution chosen was 'normal'. This function ranks and smooth out the relationship between observations to be mapped as a normal distribution.

All these three defined methods can be applied to each numeric input feature in the training set of the machine learning model. Then it will learn a predictive modelling task based on this dataset already Gaussian.

To observe the behaviour of the features of this model to this application, a function (*chang_hug_map*) was created inside *plotter.py* module, based on Eric Chang and Nicolas Hug example ^[5.2.11]. It demonstrates the use of transformations by mapping the results of applying Box-Cox, Yeo-Johnson and Quantile transformed methods.

To this aim, a pre-processing was needed to adjust some zero values to the Box-Cox method. Therefore, a Pipeline ^[5.2.12] was included to apply a MinMaxScaler ^[5.2.13] before the transformation. A Pipeline is an accumulation of ordered transforms with a final estimator. A MinMaxScaler is a scaling feature engineering method to scale features within a given range:

```
46 scaler = MinMaxScaler(feature_range=(1, 2))
47 boxcox = PowerTransformer(method='box-cox')
48 bc = Pipeline(steps=[('s', scaler), ('bc', boxcox)])
```

The Yeo-Johnson method did not need any further scaling process on data:

```
50 yj = PowerTransformer(method='yeo-johnson')
```

Finally, the Quantile Transformer was defined for `n_quantiles = 500` and an output normal distribution as mentioned before:

```

52 rng = np.random.RandomState(304)
53 qt = QuantileTransformer(n_quantiles=500, output_distribution='normal',
54                          random_state=rng)

```

To continue, transformations were fitted to train data and transformed using test data (`fit_transform` method):

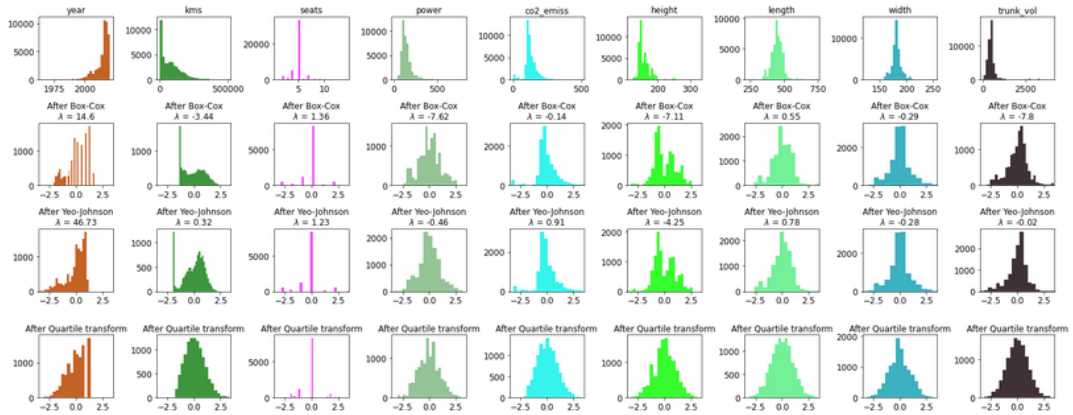
```

85 # perform power and quantile transforms
86 X_trans_bc = bc.fit(X_train).transform(X_test)
87 lambda_bc = round(bc.named_steps['bc'].lambdas_[0], 2)
88 X_trans_yj = yj.fit(X_train).transform(X_test)
89 lambda_yj = round(yj.lambdas_[0], 2)
90 X_trans_qt = qt.fit(X_train).transform(X_test)

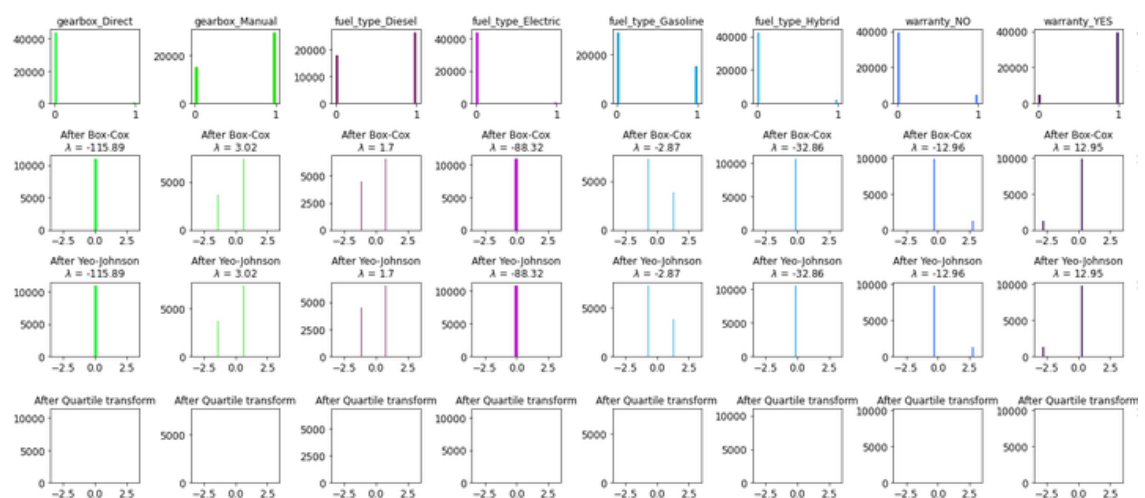
```

To wrap up, each variable was plotted after each transformation. In this way, it was possible to make a visual decision by selecting the method that better adjust features to Gaussian distributions. The best performer for this use case was Quantile Transformer, so we will continue seeing the application and impact of using this pre-processing method on future chapters. The final outcome was saved as `cars_reg_trf.csv`, consult point 4.1 (Data Cleansing), which was the final dataset used to train the model.

[16]: `chang_hug_map(X=X, hex_colors=hex_colors, FONT_SIZE=12, BINS=30)`



It is important to remark that when data is not continuously distributed, it is not possible to perform a column transformation with success. This is the case of the dummy features created on previous steps, which have a binary distribution (0, 1), by using the One-Hot Encoding method.



5.3 Data Model

This chapter is focused on exploring and testing different machine learning algorithms to find the optimal model for this project. It will start exposing the different algorithms chosen to analyse their interpretation of the dataset. Later, they will be a benchmarking process to compare each of the algorithms and their performance. Moreover, the impact after applying Column Transformation (section 5.2.4) will be explained. Finally, we will cover the hyperparameter tuning of the best model, which uses cross-validation methods to allocate the best combination possible to this use case.

5.3.1 Machine Learning Algorithms tested

The machine learning algorithms applied to this project are of supervised regression type. This is because this use case needs to predict a continuous value, which is the price of the second-hand car. Therefore, the testing ML algorithms selected were mainly based on ensemble learning and gradient boosting models. However, some linear regression models were also included to measure the different on performance against other more sophisticated models.

Regression is a method of modelling a target value based on independent predictors. Its used is extended for forecasting aims and finding out cause and effect relationship between variables. Regression techniques differ on how do they base the number of independent variables and the type of relationship between independent and dependent variables.

These are the chosen ML algorithms which will be studied on the dataset of this project:

- Linear Regression: LR ^[5.3.1]. It is based on the correlation between two variables located on the x-axis and y-axis. This means that the number of independent variables is one and that there is a linear relationship between the independent (x) and dependent (y) variable.
- Ridge Regression: RD ^[5.3.2]. It is a technique for analysing multiple regression data that could suffer from multicollinearity. This effect is usually common in models with large numbers of parameters. It adds a degree of bias to the regression estimation, reducing standard errors.
- Support Vector Regression: SVR ^[5.3.3]. These are the equivalent to Support Vector Machines (SVMs) algorithms for classification problems. It adds the flexibility to the model to define how much error is acceptable and will find an appropriate line to fit the data.
- Random Forest Regressor: RF ^[5.3.4]. A Random Forest is a meta estimator that fits a number of classifying decision trees on different sub-samples of the dataset. It uses averaging to improve the predictive accuracy and control over-fitting. It counts on some adjustable parameters to avoid that the whole dataset is used to build each tree.

- Gradient Boosting Regressor: GB ^[5.3.5]. This machine learning algorithm relies on the intuition that the best possible next model, combined with previous models, minimizes the overall prediction error by using loss functions. To this aim, the target outcomes of the next model need to be set.
- XGBoost Regressor: XGB ^[5.3.6]. Extreme Gradient Boosting is part of a class of ensemble ML algorithms, so it is based on decision tree models. It provides an efficient and effective implementation of the previous defined algorithm, Gradient Boosting.

Lately, it has become one of the key components for a wide range of problems in machine learning competitions. It is designed to be computationally efficient, so it is both fast and highly effective.

- CatBoost Regressor: CB ^[5.3.7]. Its name comes from “Category” and “Boosting” since it can handle categorical features automatically. It does not require any pre-processing steps to convert categories into numbers. Moreover, it is based on ordered boosting, which is a permutation-driven alternative to the classic algorithm.

CatBoost is a robust algorithm since it reduces the need for extensive hyper-parameter tuning. It also lowers the chances of overfitting, advocating to conform more generalized models.

Along the next sessions, we will cover which algorithms have a better performance when being parametrised and applied to the second-hand car dataset. What is more, the generalization of the model will be also evaluated since the main objective is to apply the future conformed model to new data.

5.3.2 Benchmarking of Regression Models

During the following sections, scikit-learn will be present since it is the most complete machine learning library. Before starting with the benchmarking of regression models, it is needed to split the data into train and test sets. To this aim, we import the *cars_reg.csv* dataset and use the `train_test_split` function from the `model_selection` module of `sklearn`:

Separating the data into features (X) and target (y) variables

```
[3]: # Train-Test Split
from sklearn.model_selection import train_test_split

[4]: # Removing some outliers
features = df.drop('price', axis=1)
target = df['price']

[5]: X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
```

Moreover, it is imported the module created to evaluate regression models: *model_evaluator*. When using the *eval_reg* function, the model is evaluated using these metrics:

- **R²** ^[5.3.8]: Also known as the coefficient of determination, in statistics, it is the proportion of the variance that is predictable from the independent variables. An article published in the PeerJ Computer Science journal in 2021 ^[5.3.9] states that the coefficient of determination can be more truthful than SMAPE, MAE, MAPE, MSE, and RMSE in regression analysis evaluation.

R² score is the main metric to benchmark these algorithms. The optimal value it can reach is 1. It will be the final decision-maker for the use case presented in this project.

- **RMSE** ^[5.3.10]: Root-mean-square error is a frequently used measure of the quadratic mean of the differences between the predicted and actual values. A perfect fit to the data is defined by a value of 0.
- **MSE** ^[5.3.11]: Mean squared error of an estimator measures the average of the squares of the deviations between the predicted and actual values. A perfect fit to the data is defined by a value of 0.
- **MSLE** ^[5.3.12]: Mean Squared Logarithmic error measures the ratio between the log-transformed actual and predicted values. A perfect fit to the data is defined by a value of 0.
- **MAE** ^[5.3.13]: Mean absolute error is a measure of the arithmetic average of the absolute errors between the actual and predicted values. It is a common measure of forecast error in time series analysis. A perfect fit to the data is defined by a value of 0.
- **EVS** ^[5.3.14]: Explained variation score measures the proportion to which a mathematical model accounts for the variation between actual and predicted values. It is similar to R² score since it accounts for the variance of the model, but it is expressed in percentage. The optimal value it can reach is 100%.

Once the regression metrics are clear, we can start having a look at the behaviour of the algorithms defined in the previous section. After having a quick look at the performance, the SVR was discarded due to its really poor performance (R² = 0.1113):

Support Vector Regression

```
[10]: svr = SVR()
      svr.fit(X_train, y_train)

      predictions = svr.predict(X_test)

      eval_reg(y_test, predictions)

      ### MEASURES OF REGRESSION MODEL ###
      -----

      R2 = 0.1113

      RMSE = 12189.5592

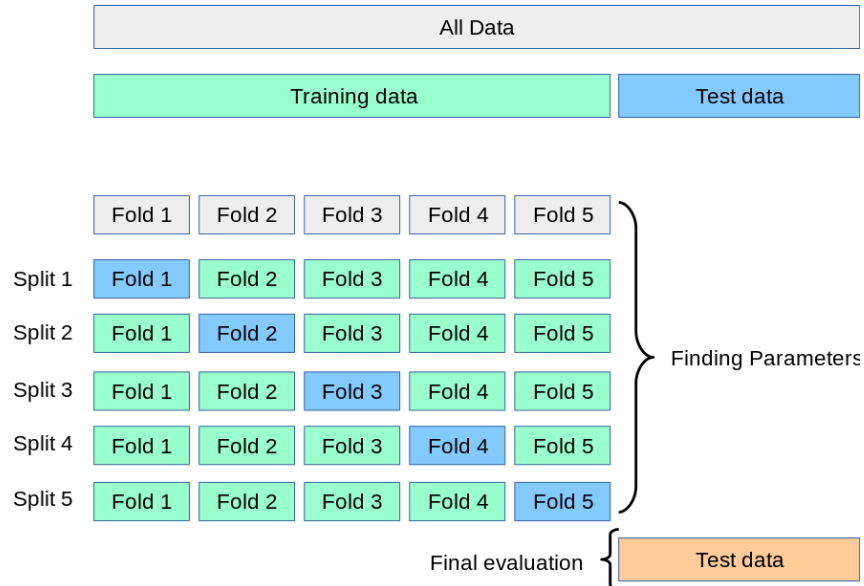
      MSE = 148585352.8881

      MSLE = 0.4265

      MAE = 7666.2253

      EVS = 16.4782%
```

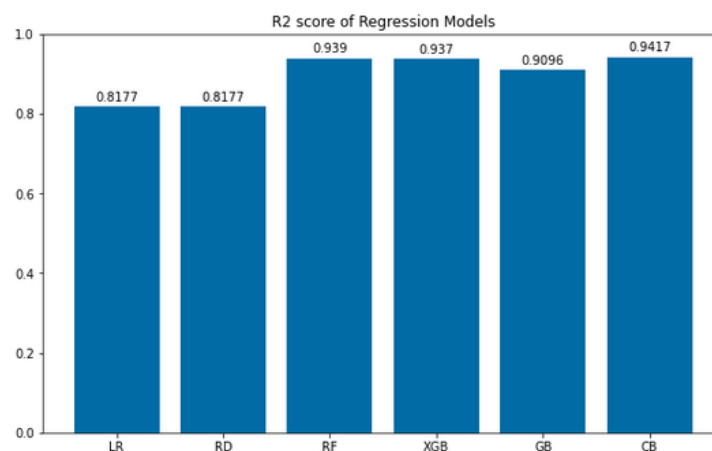
Since one iteration was not enough to decide which ML algorithm was better, it was decided to implement a k-fold cross-validation ^[5.3.15]. This resampling method provides visibility about the generalization of the model by iterating through different sets, i.e., say k=10, so 9 folds for training and 1-fold for testing purpose. This process repeats unless all folds get a chance to be the test set one by one.



To this aim, a new function was created to reproduce this K-Fold resampling for every algorithm: *run_cv_reg*. It takes these parameters: features, target, list of models and evaluator. The output of this method is a table containing the quartiles of the metric (R^2 score) for each model, presenting as well the execution time:

[18]:

	model	min_r2_score	1st_quantile	median_r2_score	mean_r2_score	std_r2_score	3rd_quantile	max_r2_score	exec_time_sec
0	LR	0.8048	0.8122	0.8144	0.8177	0.0091	0.8235	0.8329	0.6676
1	RD	0.8048	0.8121	0.8144	0.8177	0.0091	0.8234	0.8329	0.4044
2	RF	0.9300	0.9374	0.9386	0.9390	0.0045	0.9411	0.9485	664.8461
3	XGB	0.9325	0.9346	0.9369	0.9370	0.0030	0.9396	0.9415	92.2712
4	GB	0.8999	0.9061	0.9094	0.9096	0.0054	0.9143	0.9169	124.5687
5	CB	0.9363	0.9399	0.9408	0.9417	0.0036	0.9428	0.9483	153.2687

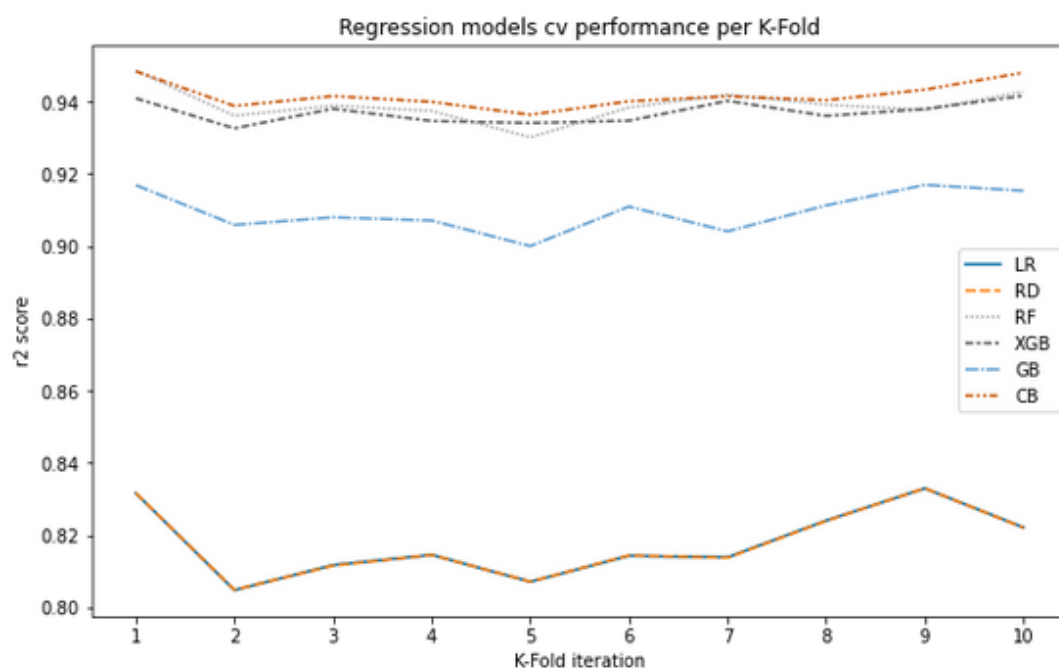


From the output table, it is evident to conclude that CatBoost is the best algorithm (mean_r2_score = 0.9417) to be considered. However, it is better to not disregard

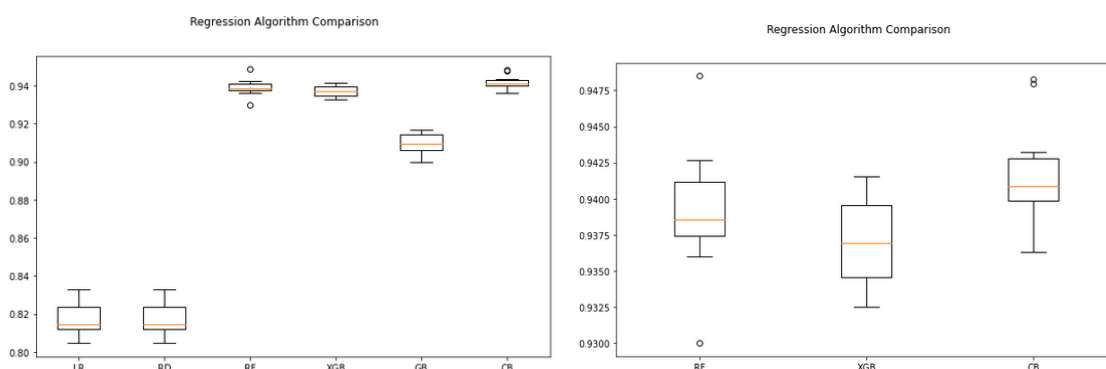
XGBoost (mean_r2_score = 0.9370) & RandomForest (mean_r2_score = 0.9390) algorithms. These results were obtained without adding any parameters to the machine learning process.

Another point to be considered, is that CB was by far the most time consuming after RF, so it is important to be considered for the final performance and speed on evaluations. In terms of execution times, linear models (LR, RD) are the fastest, but the performance it is insufficient. In case that delays in execution could be a problem, boosting algorithms could be the most suitable option. This is another reason to conclude that CB is the best performer, while RF takes really long to be processed.

The created function (*run_cv_reg*) not only produces a dataframe with the main insights, but also a *results* matrix with all the values per K-Fold set iteration. This allows to create a visible line plot with the main results for each of the algorithms:



Finally, boxplots can help to visualize the great difference in performance and variance on R^2 score results between ensemble and boosting algorithms compared to linear ones. This is why the focus will be on top of these 3 models during the following sections.



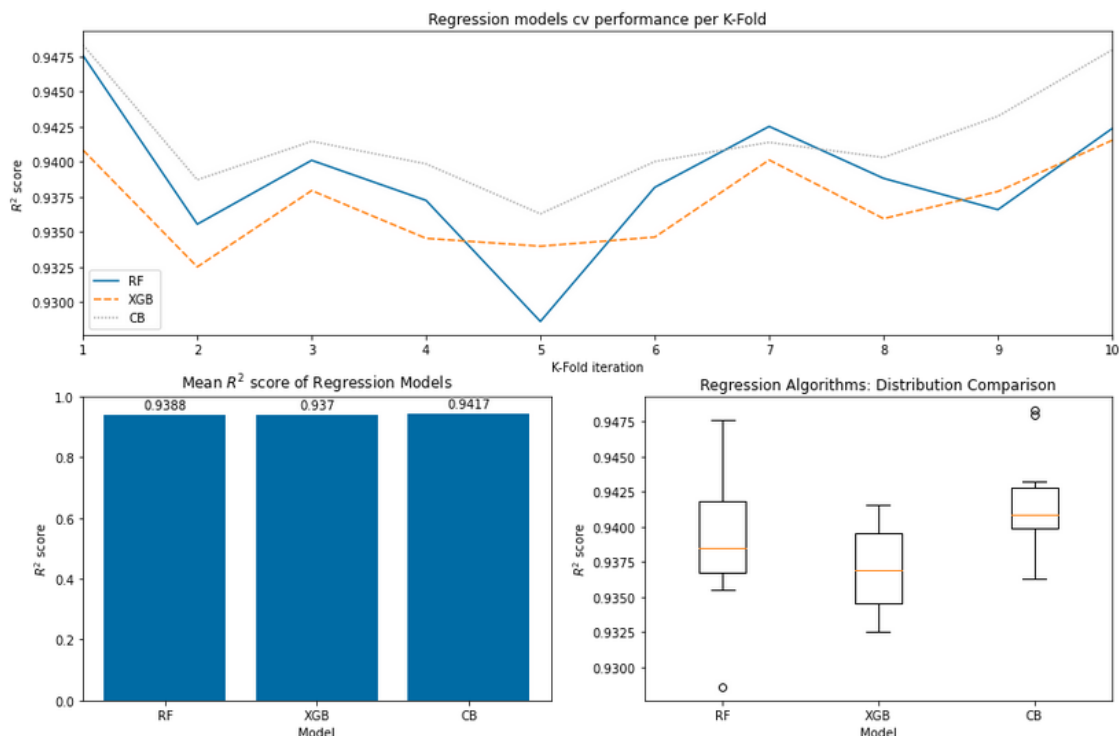
5.3.3 Measuring Column Transformers impact

This section will focus on the top 3 performers algorithms detected on the previous step: RandomForest, XGBoost, CatBoost. As analysed during point 5.2.4, the best column transformer resulted to be the Quantile Transformer, based on a normal distribution output.

Taking these previous considerations into account, we will have a look at how this column transformer can have an impact to the performance of each of the models. To this aim, it was tested how the R^2 score varied after transforming the features that drive the final prediction. To visualize these results, a function was created inside the *plotter* module: *plot_model_eval*. Therefore, the following charts represented:

- Regression models CV performance per K-Fold. Line plot that shows the performance of each input model for each K-Fold step.
- Mean R^2 score of Regression Models. Bar plot that represents the mean of the R^2 score metric on the K-Fold CV process for each algorithm.
- Regression Algorithms: Distribution Comparison. Box plot that shows the quantile distribution of each model performance.

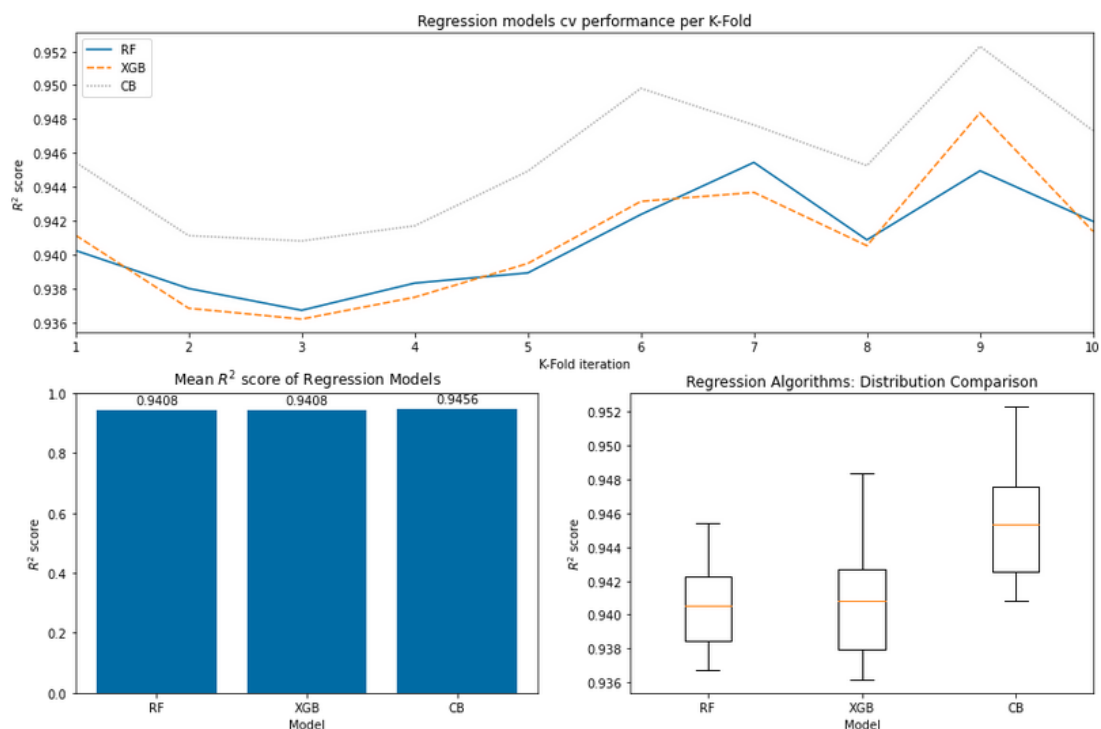
Firstly, we will explore the results for the K-Fold CV without applying any column transformation. This will be the starting point to analyse the variation on performance:



model	min_r2_score	1st_quantile	median_r2_score	mean_r2_score	std_r2_score	3rd_quantile	max_r2_score	exec_time_sec
RF	0.9286	0.9367	0.9385	0.9388	0.0048	0.9418	0.9476	500.2823
XGB	0.9325	0.9346	0.9369	0.9370	0.0030	0.9396	0.9415	26.5178
CB	0.9363	0.9399	0.9408	0.9417	0.0036	0.9428	0.9483	158.2979

It is possible to deduce that the best performer is the CatBoost algorithm (mean R^2 score = 0.9417) as we concluded on the previous point. In any case, XGBoost and RandomForest have a close performance, so after the appliance of the Quantile Transformer these results may change. Another point to consider is that on the K-Fold process, CatBoost maximum score is obtained just in a couple of scenarios. In any case, the minimum score of the winning algorithm is close to the mean of the other two. On the line plot, we can see that CatBoost is only outperformed by RandomForest on the 7-Fold, so the *max_r2_score* is higher for this model. However, XGBoost never beats CatBoost.

Once the Quantile Transformer (output: normal distribution) was applied to the initial regression dataset (*cars_reg.csv*), the performance on each K-Fold split changed for each algorithm:



model	min_r2_score	1st_quantile	median_r2_score	mean_r2_score	std_r2_score	3rd_quantile	max_r2_score	exec_time_sec
RF	0.9367	0.9385	0.9406	0.9408	0.0028	0.9423	0.9454	478.4227
XGB	0.9362	0.9380	0.9408	0.9408	0.0035	0.9427	0.9484	29.2527
CB	0.9408	0.9425	0.9453	0.9456	0.0036	0.9476	0.9523	117.3245

On this second output, after applying the transformation, we can appreciate an enhance on the performance of every algorithm. The mean R^2 score result is the same for both RandomForest and XGBoost algorithms. To CatBoost this change on the dataset produced an increase on the mean R^2 score of 39 bps ^[5.3.16]. What is more, on the line plot is possible to observe that no other model outperformed CatBoost on any K-Fold of the cross-validation process. Finally, the distribution of the winning model became steadier since the 2 whiskers are visible on its box plot and no more outliers are present.

After the Quantile Transformation boosting algorithms show a better behaviour. This is how CatBoost is proclaimed as the clear winner, so it will be the final model adopted to be implemented on the Carlyst application.

5.3.4 Hyperparameter Tuning: CV methods

Along this section, CatBoost is going to undergo different parameter configurations to decide which one is the best one. This process is known as Hyperparameter Tuning ^[5.3.17]. Machine learning models can be optimized by customizing the hyperparameters to the use case dataset. Although it is difficult to know the definition of each of the hyperparameters for each algorithm, it is even more complex to learn the interaction between their combinations to the final prediction.

In any case, it is time and resource consuming to find every possible combination for each algorithm based on its hyperparameters. Therefore, a better approach is to look for different values to try to find the best subset in that search space. This is known as hyperparameter optimization. A hyperparameter is a model configuration argument to guide the learning process for a specific dataset. This means that there is not a golden configuration for each algorithm since they will be applied to different use cases.

To find the best hyperparameter configuration, this process will be based on two consecutive subprocesses. This idea was based on Jason Brownlee article ^[5.3.18] (Hyperparameter Optimization with Random Search and Grid Search):

- Random Search: Define a search space as a bounded domain of hyperparameter values and randomly sample points in that domain. Therefore, not all the possible combinations will be explored and evaluated.
- Grid Search: Define a search space as a grid of hyperparameter values and evaluate every position in the grid.

On the one hand, Random Search is a great method to discover hyperparameters combinations that would have been difficult to be guessed intuitively. However, it usually requires more execution time to be completed. Therefore, it will be the first approach to look for the best subset of hyperparameters.

On the other hand, Grid Search is the most suitable option for spot-checking combinations that are previously known to perform well in general. Once the Random Search method spots the most efficient subsets, the parameters to be explored will be narrowed down to a more concise analysis.

Both of the presented techniques evaluate models for a given hyperparameters set using cross-validation. This leads to a firmer conclusion on the generalization of the final model, so it can be applied to new data, minimizing the risk to have adjusted an overfitted algorithm.

Firstly, the Random Search method will be explored to narrow down the search. This cross-validation method accepts the following arguments:

- *estimator*: Machine learning algorithm to use in the CV Search. In this case it will be a *CatBoostRegressor*.

- *param_distributions*: hyperparameters set of the defined estimator. We will tune the following parameter grid for CatBoost ^[5.3.19]:
 - o *iterations*: the maximum number of trees that can be built when solving machine learning problems.
 - o *learning_rate*: used for reducing the gradient step.
 - o *depth*: depth of the tree. It can be set to any integer up to 16.
 - o *l2_leaf_reg*: coefficient at the L2 regularization term of the cost function. Any positive value is allowed.
- *n_iter*: maximum number of iterations of the Random Search to look for the optimum hyperparameter combination.
- *cv*: cross-validation fold method to be implemented on the regression algorithm. As suggested by Jason Brownlee, *RepeatedKfold* will be the chosen option with 10 folds and 3 repeats.
- *scoring*: scoring metric to evaluate the performance to select the best model. In this case the model will be adapted to the R^2 score.
- *return_train_score*: This will create the *cv_results_* output of the training scores when set to "True".
- *n_jobs*: Number of jobs to be run in parallel. In this case "-1" will point to the use of all processors.
- *random_state*: seed to control the shuffling applied to the data before being split into train/test sets.

```
[4]: # define model
cb = CatBoostRegressor(verbose=False)

# define search space
param_grid = dict()
param_grid['iterations'] = [100, 150, 175]
param_grid['learning_rate'] = [0.03, 0.1]
param_grid['depth'] = [2, 4, 6, 8]
param_grid['l2_leaf_reg'] = [0.2, 0.5, 1, 3]

# define evaluation
cv = RepeatedKfold(n_splits=10, n_repeats=3, random_state=33)

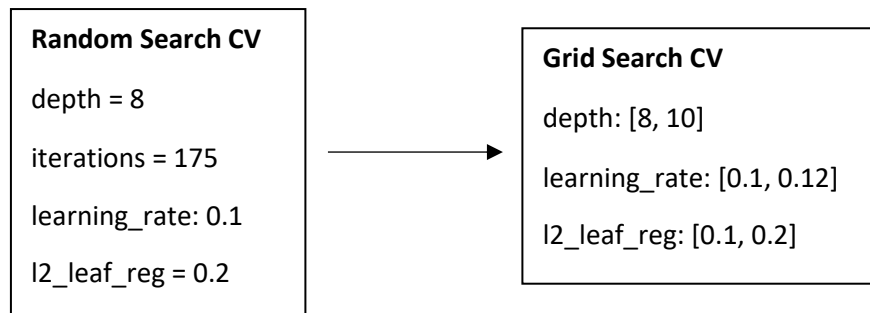
# define scoring
scoring='r2'

# define RandomSearchCV
search_rdm = RandomizedSearchCV(estimator=cb,
                                param_distributions=param_grid,
                                n_iter=100,
                                cv=cv,
                                scoring=scoring,
                                return_train_score=True,
                                n_jobs=-1,
                                random_state=33)
```

```
[7]: # summarize result
print('Best Score: {}'.format(result_rdm.best_score_))
print('Best Hyperparameters: {}'.format(result_rdm.best_params_))
print('Time consumed on the Search: {}'.format(rdm.time))
```

```
Best Score: 0.9374066147204212
Best Hyperparameters: {'learning_rate': 0.1, 'l2_leaf_reg': 0.2, 'iterations': 175, 'depth': 8}
Time consumed on the Search: 5577.818910598755
```

From the Grid Search method, it was possible to obtain R-squared = 0.9374 as best score. Although this approach it is done on random samples, we could still find the best hyperparameters on a last step based on the subset obtained. Therefore, the next step will have a narrowed down the search space to tune:



To apply the Grid Search method, CatBoost count on a built-in function to this aim: *grid_search*. It only needs the parameters grid, features and target to produce the output:

```
[30]: # define model
      cb = CatBoostRegressor(iterations=1000, eval_metric='R2', verbose=False)

      # Using the built-in grid search function
      grid = {'learning_rate': [0.1, 0.12],
              'depth': [8, 10],
              'l2_leaf_reg': [0.1, 0.2]
            }

      search_grd = cb.grid_search(grid,
                                  X=X,
                                  y=y,
                                  verbose=False,
                                  plot=True)
```

```

bestTest = 0.9454863059
bestIteration = 999

bestTest = 0.9460812942
bestIteration = 999

bestTest = 0.9452106616
bestIteration = 999
  
```

Since the number of hyperparameters combination is 8 ($2^3 = 8$), there are 8 outputs related to each combination. Moreover, the best cross-validation iteration is remarked and the best global value can be accessed in the generated Search Grid:

```
[36]: # Best found parameters
      search_grd['params']
```

```
[36]: {'depth': 8, 'learning_rate': 0.12, 'l2_leaf_reg': 0.1}
```

```

CatBoost best model

depth = 8

learning_rate = 0.12

l2_leaf_reg = 0.1

iterations = 1000 (default)
  
```

By obtaining the best CatBoostRegressor through this Grid Search, the performance of the R^2 score was enhanced to 0.9457, 83 bps better than the previous result obtained with Random Search. This result is more aligned to the results obtained after the column transformation process.

To sum up, by reading through the CatBoost documentation ^[5.3.20], it is an algorithm based on applying gradient boosting on decision trees. What is more, it is already prepared for optimizing its hyperparameters to the model. This means that CatBoost sometimes works at its top performance when using the default hyperparameters. In

any case, the results obtained guarantee that applying a hyperparameter tuning process can be the key to find the optimal configuration for ML models.

5.4 Building the optimal model

This chapter will cover how was built the optimal CatBoost model for the use case of this project. The process will start by diving the dataset into train and test values. Right after, the CatBoostRegressor will be trained with the optimal hyperparameter configuration found on the previous section. This trained model will be used to generate a prediction on the test set, so actual vs predicted values will be plotted to visualize the deviations. As an additional step, the impact that each feature has to the final model will be evaluated by using the shap values. Finally, this process will end up with the model persistence, by saving it in a compressed pickle file.

5.4.1 Evaluation of the model

The CatBoost model will be evaluated on *cars_reg_trf.csv* final dataset. The features (X) were extracted from it and separated from the target (y, price). After applying a split into train and test data, it will be used on the fit and prediction of the CB model.

The hyperparameters for the CatBoost have been chosen based on the results of the Grid Search cross-validation process:

- `iterations`: 1000
- `learning_rate`: 0.12
- `depth`: 8
- `l2_leaf_reg`: 0.1

If overfitting occurs, CatBoost can stop the training earlier than the preconfigured options dictate. To achieve this, an additional parameter was including when fitting the model: *early_stopping_rounds* ^[5.4.1]. This overffiting detector takes into account both *od_type* and *od_wait*:

- *od_type*: type of overfitting detector.
- *od_wait*: the number of iterations to continue the training after the iteration with the optimal metric value is found. Together with the “Iter” *od_type* detector, it considers the model overfitted and stop training after the specified number of iterations.

```
[5]: start_time = time()

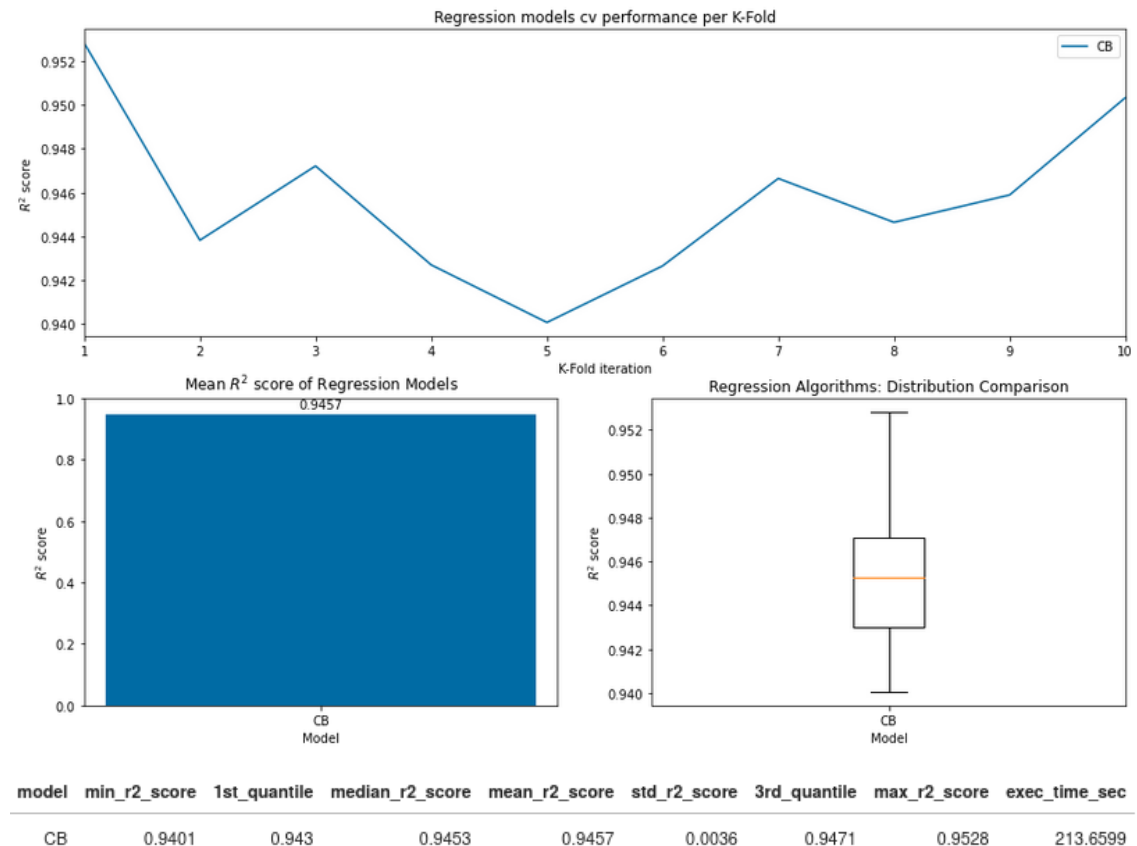
cb = CatBoostRegressor(depth=8,
                       learning_rate=0.12,
                       l2_leaf_reg=0.1,
                       iterations=1000,
                       eval_metric='R2',
                       verbose=False)

cb.fit(X_train, y_train, early_stopping_rounds=10) # od_type="Iter", od_wait=10

exec_time = time() - start_time
print('The CB model was trained in: {:.2f} sec'.format(exec_time))

The CB model was trained in: 22.58 sec
```

For the evaluation of this trained model, a previously defined module will be used: *plotter*. This one contains the *plot_model_eval* function, which can help on the visualization of the performance of the model:



As it was expected, the same mean R^2 score = 0.9457 value is obtained after launching the CV process.

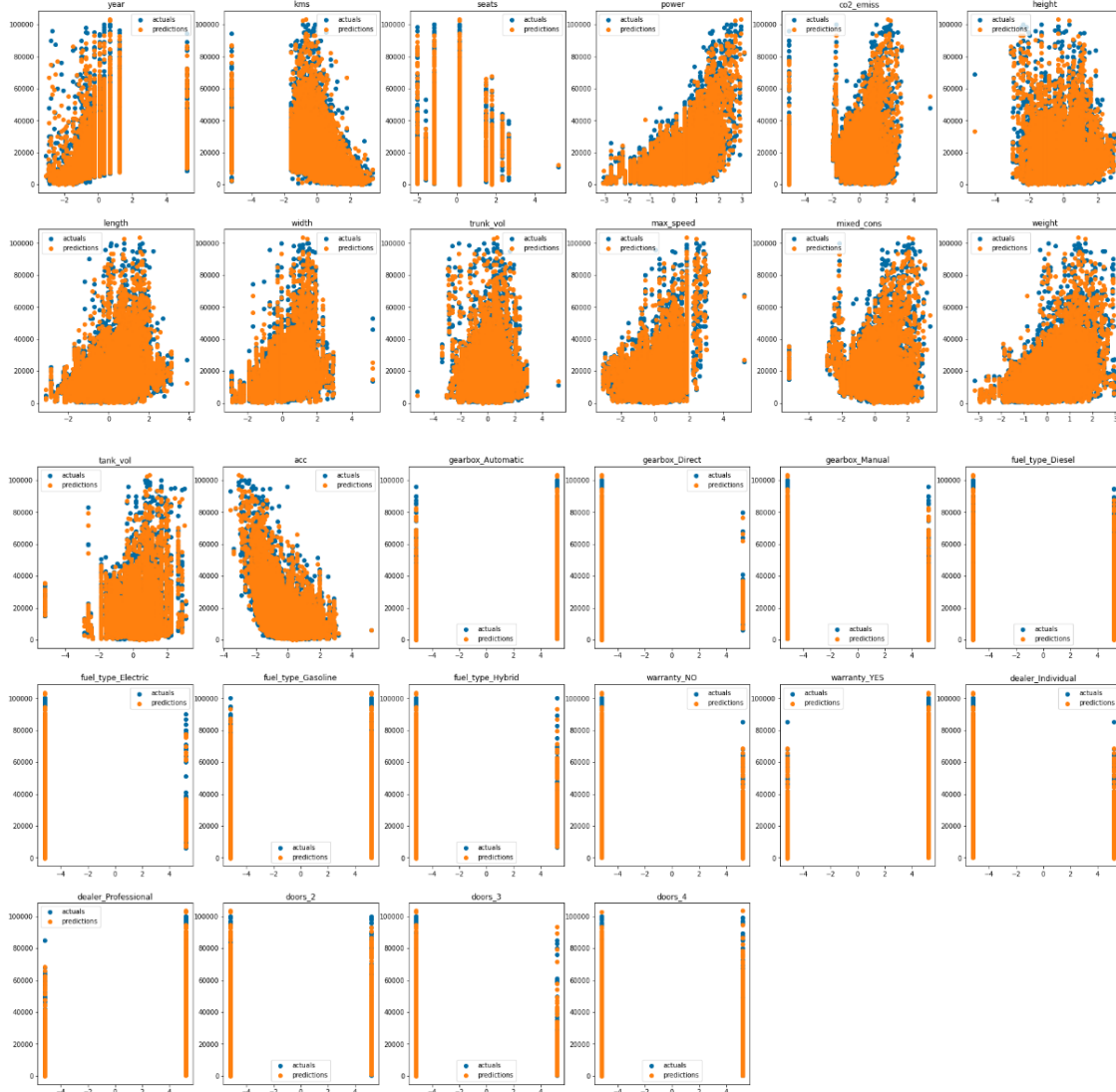
5.4.2 Actual vs predicted values

At this step, we will have a look at how predictions look vs actual values. A visualization analysis will be performed to better understand the scope of each feature and how each of them are related to the target, the second-hand car price.

During this section, firstly, it will be exposed the relation between each feature and the price. Secondly, we will see how the model is adapted to each sample. These scatter plots will help on the decision-making process as an extra step to validate the performance of the CatBoost model.

The following graph is based on the impact of the model on each feature. It is visible that the values are correctly predicted, specially for cheaper cars. Therefore, the higher the price, the less accurate will be the prediction of the estimator. This is due to the fact that the trained dataset has less samples of luxury and historical second-hand cars.

CatBoost: Feature Prediction



Now, we will have a look at the predictions of the whole model to see how they are adapted to actual values. The scatter plot below is a representation of Actual vs Predicted values. From this chart, it is possible to take the following conclusions:

- The closer the scatter points are to the regression line, the better the model.

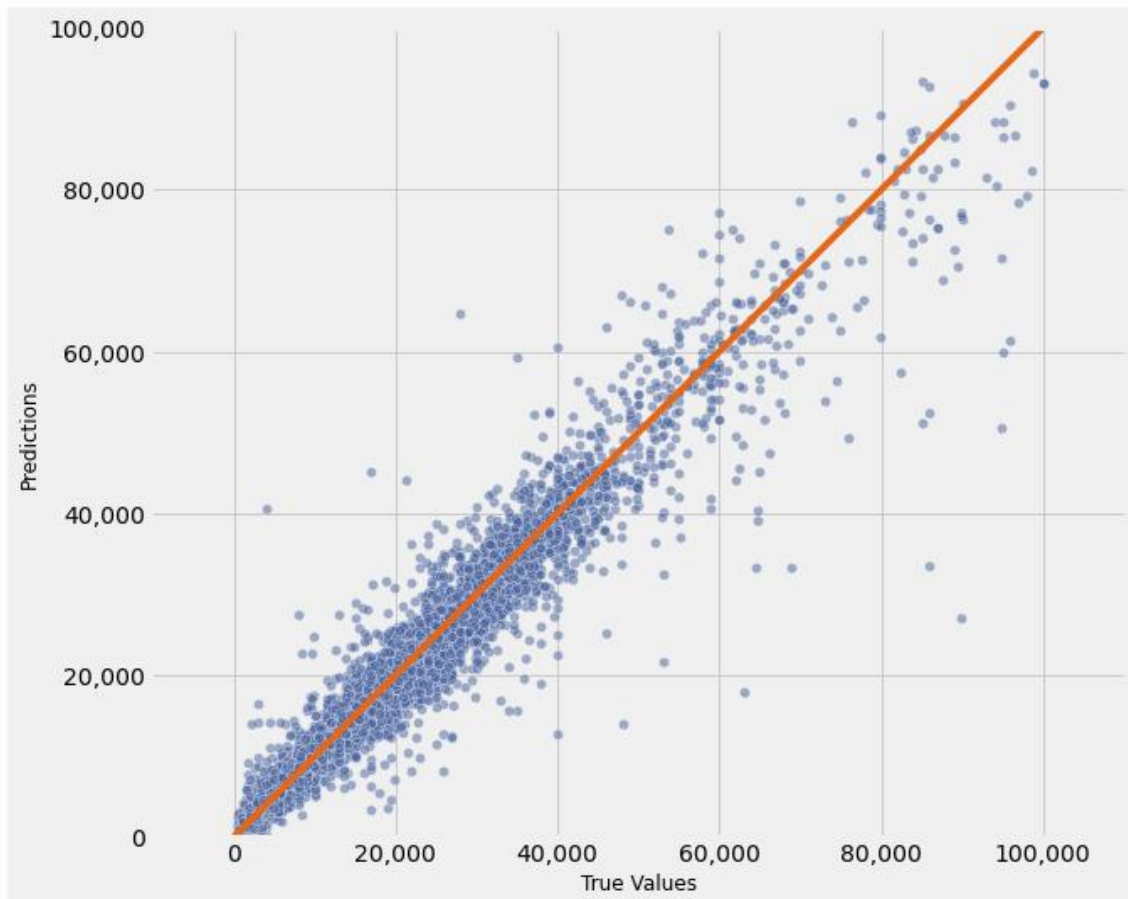
The R^2 score obtained was of 0.9457, which is a high score. This results in all the points being close to the diagonal line. The lower the R^2 score, the weaker the precision on the fit of the model.

- Having a deeper look into the values, we can conclude that the higher the price of the car, the more dispersed is the model.

This affirmation was already visible when analysing the prediction of each feature. The number of second-hand cars with a value higher than 50,000 € are scarce compared to more accessible cars. Therefore, we could expect a better performance of the model when predicting prices of non-luxury cars.

- Residual values are those separated from the regression line. If they are above the line, they are positive, whereas if they are below, they are negative.

Residuals seem to be balanced for lower prices; nevertheless, negative values seem to be predominant with higher prices. Thereupon, there can be underestimations on predicting cars with a higher market value.

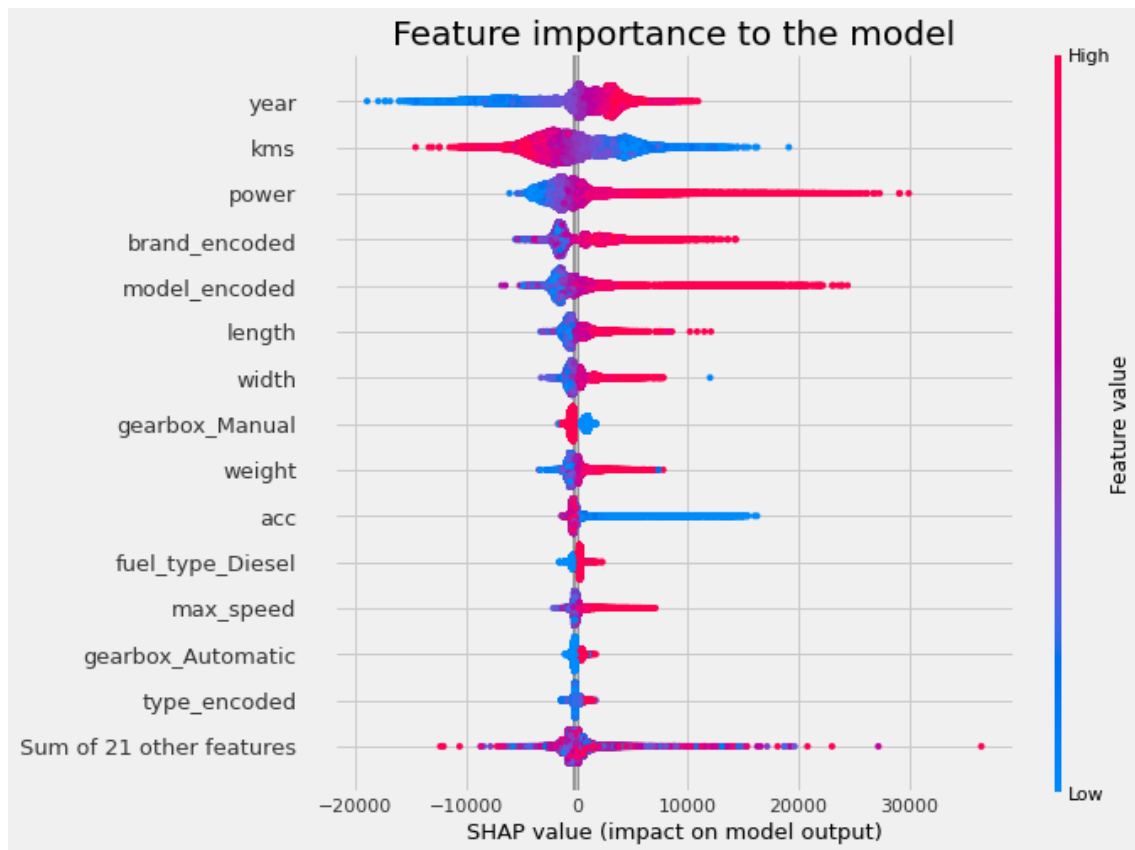


To conclude, the performance of this CatBoost model is more than acceptable since the points are fit to the regression line and the R^2 score value is close to 0.95.

5.4.3 SHAP figures: Feature importance

The SHAP figures exposed on this point are based on the Shapley value ^[5.4.2]. It is based on this theory: A coalition of players cooperates, and obtain a final overall gain for the cooperation. Some players will contribute more to the coalition than other or even a different bargaining power (they could threat to destroy the total surplus). Therefore, the Shapley values answer these questions: how important is each player to the overall cooperation, and what payoff can they reasonably expect?

SHAP stands for Shapley Additive Explanations and it can be used in order to visualize which features have a greater impact to the model. The colour represents feature values: red is high, blue is low as detailed on the legend.



The image above explains the behaviour of the machine learning model. In example, it will consider that a low manufactured year, contributes to a lower final predicted car price. Moreover, we can observe that cars with a high horse power, will result in a higher prediction. Another contra-positional observation is the conclusion obtained from the number of kilometres, the higher they are, the lower the price of the car.

These all affirmations meet the criteria that a car appraiser should have, so the model is aware of the basic reasonable insights. However, it also can find out that a heavier car is usually more expensive than a lighter one through the weight feature. Machine learning models can find patterns on datasets that could not be obvious.

5.4.3 Model persistence

The methodology chapter will be closed with the model persistence ^[5.4.3]. Once a machine learning model is trained, it is important to know how to save a model for a future use without the need to retrain it each time. In this way, the trained algorithm could be used in new data or even to create predictions in an app, which is the final goal of this project: Predict the value of second-hand cars.

The most extended library to maintain models is pickle. However, a compression library will be used to reduce the size of the final saved model: bz2. Thereupon, a new module was written to cover these tasks: *pickle_jar*. It allows to create a normal and compressed pickle, as well as a function to decompress these files and be loaded to predict on new data.

The starting point taken to learn how to produce this pickle file was this article: *Load Data Faster in Python with Compressed Pickles* ^[5.4.4]. To produce the final model, the entire dataset was placed as training data. Through this method, predictions on new data will be richer since they will be more data-driven.

It is possible to benchmark the pickle and bz2 methods to save the model by comparing the final size. The normal size of the model was about 4 MB, while the compress version was around 1 MB. The reduction is significative (~350%), whereas in fact, there should not be a huge impact on performance.

Time to save this model using pickle!

```
[20]: create_pickle("models/cb_model.pkl", cb_model)
[21]: Path("models/cb_model.pkl").stat().st_size
[21]: 4184211
[22]: compressed_pickle("models/cb_model", cb_model)
[23]: Path("models/cb_model.pbz2").stat().st_size
[23]: 1194586
[24]: size_rel = Path("models/cb_model.pkl").stat().st_size / Path("models/cb_model.pbz2").stat().st_size
      print("Pickle wo/ being compressed is {:.2f}% heavier in size.".format(size_rel * 100))
Pickle wo/ being compressed is 350.26% heavier in size.
```

Nevertheless, although CatBoost was the main regression solution to approach this problem, a RandomForest was also built. This is why the two last notebooks were replicated (a, b) to evaluate the difference in performance between these two models:

11a_Hyperparameter_Tuning-RF.ipynb	11b_Hyperparameter_Tuning-CB.ipynb
12a_Model_Builder-RF.ipynb	12b_Model_Builder-CB.ipynb

This will help to visualize the limitations on performance of these algorithms compared to gradient boosting ones. The main difference is driven by the size.

The final application is based on Streamlit, which re-runs the code every time an interactive parameter is changed on the platform. This behaviour can experiment decreases on performance and user experience when the model running in the back-end is too heavy.

The size of the original RandomForest pickle was higher than 600 MB, while the compressed version could reduce a 482% of space, resulting in 125 MB. This produced a significant difference on running times on the service, so CatBoost prediction takes about 3 seconds, while the previous compressed RF model took around 2 minutes.

Finally, the app product was built on top of the CatBoost algorithm. Not only due its better performance, but because it was more suitable to this use case. When time is not concern, using the best performing model according to the metric could be the best approach. However, when times are needed to be shorten up, the most efficient algorithm should be the chosen option. This project was looking for an almost real-time prediction. Therefore, even if the RandomForest had the best R-squared score, CatBoost or XGBoost could have been the winning horse.

6. Frontend with Streamlit: User manual

This chapter will start by giving a general background of what is behind the scenes of the Streamlit application. This explanation will be followed by a user manual to easily handle this web service without any constraints.

6.1 How was the backend built?

Behind the scenes, a CatBoost ML model is running to provide a real-time price prediction of a second-hand car. Parameters can be input both manually and by using a coches.com link.

To this aim, a frontend module was built: *carlyst_st_app*. The app is maintained by some other modules that are integrated inside this script:

- *fe_cars*: contains the *frontend_preproc* function. It applies the whole feature engineering process to new data to be prepared before predicting the value.
- *pickle_jar*: loads the model saved in a compressed pickle (*cb_model.pbz2*) to be used on new data input by the user.
- *car_link*: utilizes the *used_car_input* function to read a url input by the user. Apply all the previously created web scrapping modules to obtain the new car information from the web direction.
- *frontend_fmt*: it produces the main layout of the frontend. It adds the background colour, fonts for each text, images and the interactive footer.

The interactive control panel was programmed on top of streamlit. This is how sliders and selection boxes are working on the sidebar. These input features are later compiled into a dataframe to be consumed by the ML model.

This dataset is transformed using the feature engineering module and, once the CatBoost model is loaded, it is used to produce the final price prediction in a table.

On the user manual, the interaction with the input widgets will be exposed as well of what should be the expected output of the application.

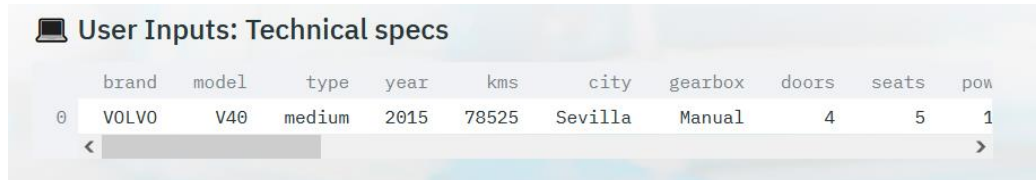
6.2 User Manual

When accessing to <https://carlyst.herokuapp.com/>, this is the initial layout of the app:



These are the sections that we can find on the main page:

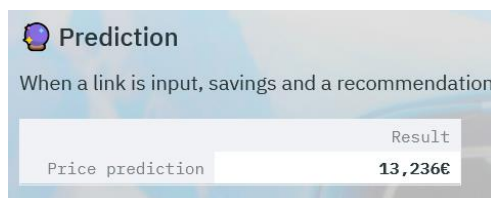
- **Abstract:** Introduction to the application. Contains the use and background about the data that conforms the backend.
- **User Inputs:** Technical specifications of the car for which the model will generate the prediction. It is based on the parameters input by the user.



	brand	model	type	year	kms	city	gearbox	doors	seats	pow
0	VOLVO	V40	medium	2015	78525	Sevilla	Manual	4	5	1

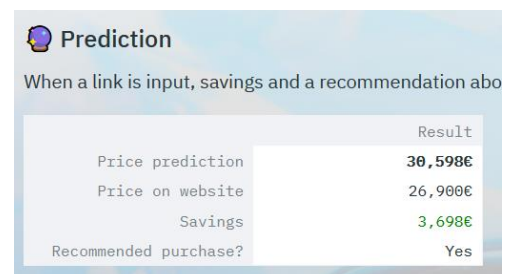
- **Prediction:** This section contains the aim of this project, the final price prediction. When parameters are used directly, it will just retrieve the price. However, when a link from coches.com is pasted, a recommendation will appear.

Direct Prediction



When a link is input, savings and a recommendation	
	Result
Price prediction	13,236€

Link Prediction



When a link is input, savings and a recommendation	
	Result
Price prediction	30,598€
Price on website	26,900€
Savings	3,698€
Recommended purchase?	Yes

- **Behind the scenes:** It contains further details regarding the model used.
 - **Parameters' impact:** SHAP plot with features' contribution to the price.
 - **How different are predictions from actual values:** A deeper insight about the data that helped to conform the final model.
- **References:** link to the Github repository ^[6.2.1] of this project.
- **Footer:** links to LinkedIn and GitHub profiles of the author.

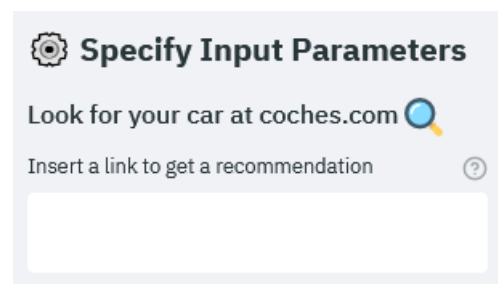
Designed in  by Carlos Espejo Peña





Now that is clear how to interpret the main page, we will have a look at the Input Parameters sidebar. It can be directly accessed by clicking on the upper-left arrow:




It starts with a text box to input a link from coches.com. We can use this feature in case we want to use this platform to directly predict a car price that is already on the real Spanish market:

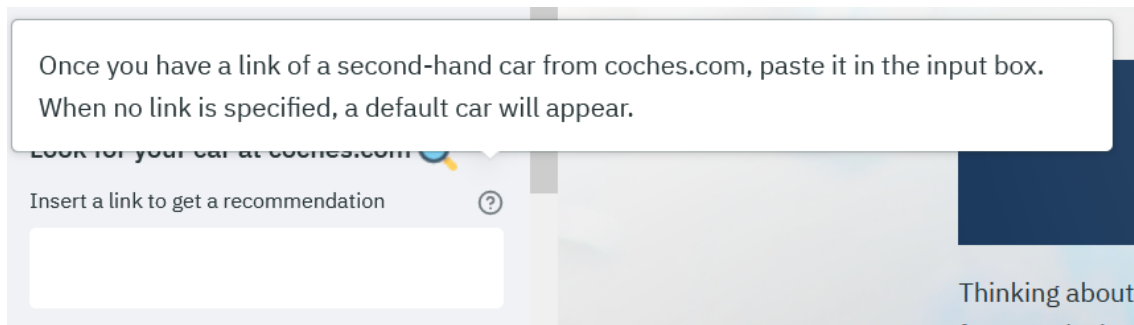


 **Specify Input Parameters**

Look for your car at coches.com 

Insert a link to get a recommendation 

We can obtain further indications about features by hovering over the question marks:



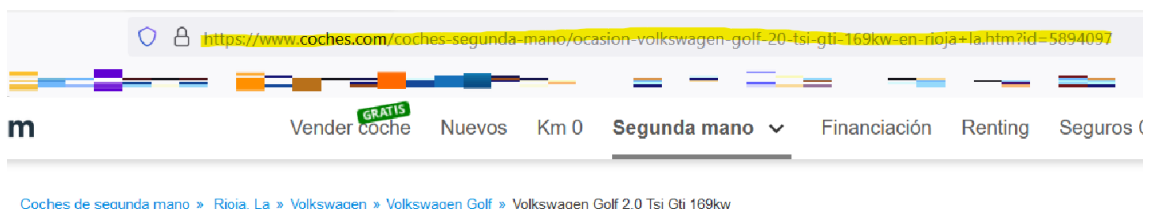
This tooltip suggests to get a coches.com link and paste it in the text box. In case of no link provided, a direct prediction will be made based on the adjusted parameters. By clicking on the magnifying glass, coches.com will be opened on its used cars section:



89.679 - Coches de segunda mano

Tenemos 89.679 coches de segunda mano para esta búsqueda. 11.047 coches son de particulares y 78.632 de concesionario.

You can look for any car in this website and copy its link:



Volkswagen Golf 2.0 Tsi Gti 169kw 230 cv en Rioja, La

Then paste the link in the input box and press enter as suggested to produce the price prediction. Moreover, a recommendation to buy or not the car is provided:

Specify Input Parameters

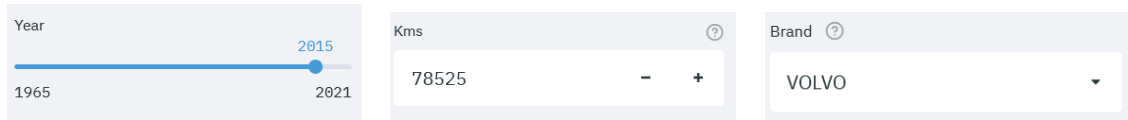
Look for your car at coches.com

Insert a link to get a recommendation

Press Enter to apply

	Result
Price prediction	23,031€
Price on website	25,990€
Savings	-2,959€
Recommended purchase?	No

In this example, the purchase is not recommended since the prediction is lower than the price provided by the link. Therefore, the model is stating that this buy could cause almost 3,000€ of losses.



The image shows a web interface with three input fields for car parameters. The first field, labeled 'Year', is a slider ranging from 1965 to 2021, with a blue dot indicating the selected year of 2015. The second field, labeled 'Kms', is a text box containing the value '78525' with minus and plus buttons for adjustment. The third field, labeled 'Brand', is a dropdown menu with 'VOLVO' selected. Each field has a small question mark icon for help.

To further modify parameters, sliders, text boxes and selection boxes can help on this task. When a link is inserted, it can be deleted, then press enter, to come back to the original configuration of the web service.

7. Conclusions and Future steps

The aim of this project was to find a solution to automatically appraise the value of cars in the Spanish second-hand market. This service will provide both buyers and sellers a global view of the prices that they could establish to make a final accurate decision on their operations.

Before this application was launched, the market already counted on alternatives to solve this problem. On the one hand, we have people that currently work as car appraisers. Without any doubt, experienced professionals could have a better judgement of a specific car as they spend time and effort performing deep analyses: detect imperfections, study specific markets, evaluate historical models. However, while the grade of experience of a professional should be good enough to beat a machine, the key factor is when time is a matter. On the other hand, there are already created algorithms on these online portals. However, I found a gap on the few variables used since they plan to be simple to be handy for users.

This project found in machine learning a perfect ally to produce an automatic prediction. The CatBoost regression model designed for this ad-hoc project was suitable enough to fulfil the demanded application. This algorithm integrated in the designed app allows the user to predict the price of a used car in just 3 seconds, which is a task impossible to be performed by a human in this time. Moreover, up to 25 features

The solution found changed the state of art by offering a tool with a fast performance, which considers up to 25 features of a car at the same time to produce an accurate prediction of its price. Moreover, it offers a new option that I could not see in a similar tool before: insert a link to get a recommendation. This app is able to offer a prediction based on a link that contains the information of the car to be evaluated.

Nevertheless, there is still room for improvement for this tool and the world of used cars automatic appraisers. Due to its speed, I can imagine these models to be integrated on second-hand ads, so they could be attached as a widget to directly make a prediction. While this implementation might be a game changer in terms of having a market with fair price, the model should be trustworthy to not produce any bias to not purchase a vehicle.

Another point to be considered is that this model is limited. It can predict the value of second-hand cars only in the Spanish market and with a value up to 100k€. Training data could be expanded to evaluate other aspects: explore world market, obtain data for luxury cars with a higher market value. In addition to this, new projects could emerge from this one, such as produce a model for historical cars or design an algorithm to appraise renting cars market.

Finally, this model was created based on a dataset extracted on the 9th of March of 2021. This means that predictions are static, so a known limitation of this model is that it would probably be useless or misleading in five years from now. Thereupon, the best solution would be to escalate this application to the cloud, i.e., AWS. From this point, a new approach could be taken to produce the daily dataset and create a final algorithm that will evolve day on day.

To sum up, these three future steps could benefit the used cars market. The direction to a more elaborated solution would be to include more meaningful features so as to enrich the model. Moreover, escalating this solution should help on creating a model that will not be affected by the continuously evolving market.

Appendix: Reference links and articles

Chapter 1

[1.1] coches.com website: <https://www.coches.com/>

[1.2] Carlyst. The name of the app is based on the conjunction of Car and Analyst terms: <https://carlyst.herokuapp.com/>

[1.3] CatBoost. A gradient boosting algorithm developed by Yandex: <https://catboost.ai/>

[1.4] Yandex. Russian tech-company: <https://yandex.com/company/>

Chapter 2

[2.1] Linux organisation: <https://www.linux.org/>

[2.2] Oracle VM VirtualBox: <https://www.oracle.com/uk/virtualization/virtualbox/>

[2.3] Xubuntu: <https://xubuntu.org/>

[2.4] Python: <https://www.python.org/>

[2.5] Anaconda: <https://www.anaconda.com/>

[2.6] Jupyter: <https://jupyter.org/>

[2.7] Heroku: <https://www.heroku.com>

[2.8] Markdown: <https://en.wikipedia.org/wiki/Markdown>

[2.9] HTML: <https://developer.mozilla.org/en-US/docs/Web/HTML>

Chapter 3

- [3.1.1] coches.com: <https://www.coches.com/>
- [3.1.2] AWS stands for Amazon Web Services: <https://aws.amazon.com/>
- [3.1.3] CSV file: https://en.wikipedia.org/wiki/Comma-separated_values
- [3.2.1] Web scraping: https://en.wikipedia.org/wiki/Web_scraping
- [3.2.2] Regular Expression: https://en.wikipedia.org/wiki/Regular_expression
- [3.2.3] XML Path Language: https://www.w3schools.com/xml/xpath_intro.asp
- [3.2.4] Nested list to a pandas DataFrame: [StackOverflow](https://stackoverflow.com/questions/45456942/nested-list-to-a-pandas-dataframe)
- [3.2.5] Pandas DataFrame: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html>

Chapter 4

- [4.2.1] What is an outlier in statistics? <https://en.wikipedia.org/wiki/Outlier>
- [4.2.2] Max lifespan of a car engine:
<https://www.caranddriver.com/research/a32758625/how-many-miles-does-a-car-last/>
- [4.3.1] Linear correlation: <https://en.wikiversity.org/wiki/Correlation>
- [4.3.2] Pearson's Correlation Coefficient:
https://en.wikipedia.org/wiki/Pearson_correlation_coefficient
- [4.3.3] [pandas.DataFrame.corr](https://pandas.pydata.org/pandas-docs/stable/10min/7.html#pearsonr)
- [4.3.4] Pearson r value Article:
https://www.westga.edu/academics/research/vrc/assets/docs/scatterplots_and_correlation_notes.pdf
- [4.3.5] Why exclude highly correlated features when building a regression model?
<https://towardsdatascience.com/why-exclude-highly-correlated-features-when-building-regression-model-34d77a90ea8e>

Chapter 5

- [5.1.1] pairplot: Plot pairwise relationships in a dataset.
<https://seaborn.pydata.org/generated/seaborn.pairplot.html>
- [5.1.2] clustermap: Plot a matrix dataset as a hierarchically-clustered heatmap.
<https://seaborn.pydata.org/generated/seaborn.clustermap.html>
- [5.1.3] Inverse Exponential Function: <http://www.nabla.hr/FU-ExpLogFuEq1.htm>
- [5.1.4] scikit-learn: <https://scikit-learn.org/stable/>
- [5.2.1] Feature Engineering:
<https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/>

[5.2.2] One-Hot Encoding:

<https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/>

<https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd>

[5.2.3] [pandas.get_dummies](#)

[5.2.4] Detecting categorical variables:

<https://support.sas.com/documentation/onlinedoc/stat/141/introcat.pdf>

[5.2.5] Target Encoding: <https://maxhalford.github.io/blog/target-encoding/>

[5.2.6] Target Encoding vs One-Hot Encoding:

<https://medium.com/analytics-vidhya/target-encoding-vs-one-hot-encoding-with-simple-examples-276a7e7b3e64>

[5.2.7] Normal Distribution: https://en.wikipedia.org/wiki/Normal_distribution

<https://machinelearningmastery.com/continuous-probability-distributions-for-machine-learning/>

[5.2.8] Power Transformations: <https://machinelearningmastery.com/power-transforms-with-scikit-learn/>

[5.2.9] Box-Cox and Yeo-Johnson: https://en.wikipedia.org/wiki/Power_transform

[5.2.10] Quantile Transforms: <https://machinelearningmastery.com/quantile-transforms-for-machine-learning/>

[5.2.11] Map data to a normal distribution (Eric Chang & Nicolas Hug):

https://scikit-learn.org/stable/auto_examples/preprocessing/plot_map_data_to_normal.html#sphx-glr-auto-examples-preprocessing-plot-map-data-to-normal-py

[5.2.12] Machine Learning [Pipelines](#)

[5.2.13] [MinMaxScaler](#)

[5.3.1] Linear Regression: <https://in.springboard.com/blog/what-is-linear-regression/>

[5.3.2] Ridge Regression:

https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf

[5.3.3] Support Vector Regressor: <https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2>

[5.3.4] Random Forest Regressor:

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>

[5.3.5] Gradient Boosting Regressor:

<https://www.displayr.com/gradient-boosting-the-coolest-kid-on-the-machine-learning-block/>

[5.3.6] XGBoost for Regression: <https://machinelearningmastery.com/xgboost-for-regression/>

[5.3.7] CatBoost Regressor:

<https://www.kdnuggets.com/2018/11/mastering-new-generation-gradient-boosting.html>

[5.3.8] R^2 score: https://en.wikipedia.org/wiki/Coefficient_of_determination

[5.3.9] PeerJ Computer Science Article, 2021. R-squared is more informative than SMAPE, MAE, MAPE, MESE, RMSE: <https://peerj.com/articles/cs-623/>

[5.3.10] RMSE: https://en.wikipedia.org/wiki/Root-mean-square_deviation

[5.3.11] MSE: https://en.wikipedia.org/wiki/Mean_squared_error

- [5.3.12] MSLE: <https://insideaiml.com/blog/MeanSquared-Logarithmic-Error-Loss-1035>
- [5.3.13] MAE: https://en.wikipedia.org/wiki/Mean_absolute_error
- [5.3.14] EVS: https://en.wikipedia.org/wiki/Explained_variation
- [5.3.15] K-Fold Cross-Validation: <https://machinelearningmastery.com/k-fold-cross-validation/>
- [5.3.16] BPS. Basis Points: <https://www.investopedia.com/ask/answers/what-basis-point-bps/>
- [5.3.17] Hyperparameter Tuning:
<https://neptune.ai/blog/hyperparameter-tuning-in-python-a-complete-guide-2020>
- [5.3.18] Hyperparameter Optimization with Random Search and Grid Search
- Jason Brownlee, 2020:
<https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/>
- [5.3.19] CatBoost Hyperparameter Tuning: <https://catboost.ai/docs/concepts/parameter-tuning.html>
- [5.3.20] CatBoost Documentation: <https://catboost.ai/docs>
- [5.4.1] CatBoost Overfitting detector: <https://catboost.ai/docs/features/overfitting-detector-desc.html>
- [5.4.2] Shapley value: https://en.wikipedia.org/wiki/Shapley_value
- [5.4.3] Model persistence: https://scikit-learn.org/stable/modules/model_persistence.html
- [5.4.4] Load Data Faster in Python with Compressed Pickles - Carlos Valcarcel, 2020:
<https://betterprogramming.pub/load-fast-load-big-with-compressed-pickles-5f311584507e>

Chapter 6

- [6.2.1] Github repository of the project: <https://github.com/caresppen/UsedCarsAppraiser>