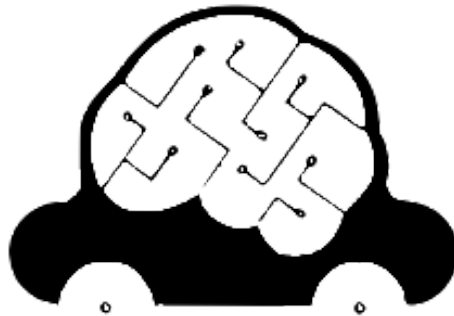


# CAR GENIUS



Master in Data Science  
KSchool

# Index

<b>1.Introduction</b>	<b>3</b>
<b>2.Raw data</b>	<b>4</b>
2.1 Data source	4
2.2 HTML attributes	4
2.3 Data scraper	6
<b>3.Data Cleaning</b>	<b>9</b>
<b>4.Data Analysis</b>	<b>11</b>
4.1 Data visualization	11
4.2 Outliers	13
<b>5.Methodology</b>	<b>16</b>
5.1 Feature engineering	16
5.2 Model	17
5.3 Model Optimization	22
<b>6.Frontend</b>	<b>24</b>
<b>7.Requirements</b>	<b>28</b>
7.1 Dependencies	28
7.2 User Manual	28
<b>8.Conclusions</b>	<b>29</b>
<b>10.References</b>	<b>30</b>

## 1.Introduction

Car Genius is a used-car price predictor developed for my thesis in the Data Science Master from KSchool. The name is the result of combining the words Car and Genius , the last one as a synonym of Guru related to the aim of this project which is to build a machine learning algorithm able to predict the price of any used car in Spain based on the current used cars market and applying the knowledge acquired during this master.

The project will cover all phases related to a Data Science project, involving the data acquisition and transformation, analysis, including the evaluation of the different machine learning models and will end with a frontend or visualization tool that will allow users to interact with the algorithm.

I found this subject very interesting not only for car owners but also for companies, especially lease cars companies that need to sell their cars at the end of each lease contract. In Spain It is not easy to get an accurate estimation for the value of your car. One possibility if you decide to sell your car is to contact a car buyer who will make you an offer, a second option is to visit some of the online portals dedicated to the used cars market where you are able to get a car appraisal when feeding a form with your car details. But these two options are very time consuming apart from not being very accurate as the offer you will get is undervalued as the buyers will save a margin on the offer as later they will need to sell the car to a third party. A different possibility but more tedious one is to dig down on some of the portals mentioned above and search for cars like yours with the same specifications and evaluate your car according to the market situation. In my short work experience in lease car companies I have seen how these companies also use these portals for the pricing of their used cars at the termination of each lease contract.

Briefly, this tool developed during the master could be very useful either if you are a car owner who wants to know the value of your car or you are a lease car company about to post an ad of a used car and want to do a quick market research before setting the price.

## 2.Raw data

This chapter is dedicated to obtaining the whole database of used cars that are currently available on the market. To do so I will not only describe the data source and fields used to build the raw dataset but also all the data science techniques used to do it.

### 2.1 Data source

For this project I have chosen as the datasource the website Autoscout 24, the biggest and most important online portal in Europe dedicated to used cars and one of the most famous in Spain. This portal currently receives 10M of visits a month and it contains 2M of vehicle ads in Europe ("AutoScout24"). In particular, for the Spanish market the portal contains more than 120k ads of used cars only (Pre-registered and second hand cars) Link: <https://www.autoscout24.es/>


For this purpose I have decided to use the technique Web Scraping to obtain the maximum amount of ads for the datasource. The scraper designed takes about 10 hours to obtain all the data and It scraps more than 34k ads. As it is a long process, the project has been done with an extraction produced on the 22nd of November of 2021.

### 2.2 HTML attributes

Once we have selected the data source we will have to decide which fields of the ones available on the portal will be interesting to study in order to include them on the scraper.

If we have a look at the main page with all ads we will see all the fields containing the characteristics available for each vehicle (title, version, price, kilometers, date of first registration, power, transmission, fuel type, fuel consumption, CO2 emissions and Sellers information)

BMW 218 218i Coupé

★ Add to list  Share

€ 31,206.-



9 km

08/2021

100 kW (136 hp)

Used

- (Previous Owners)

Manual

1/4

Gasoline

6 l/100 km (comb.)

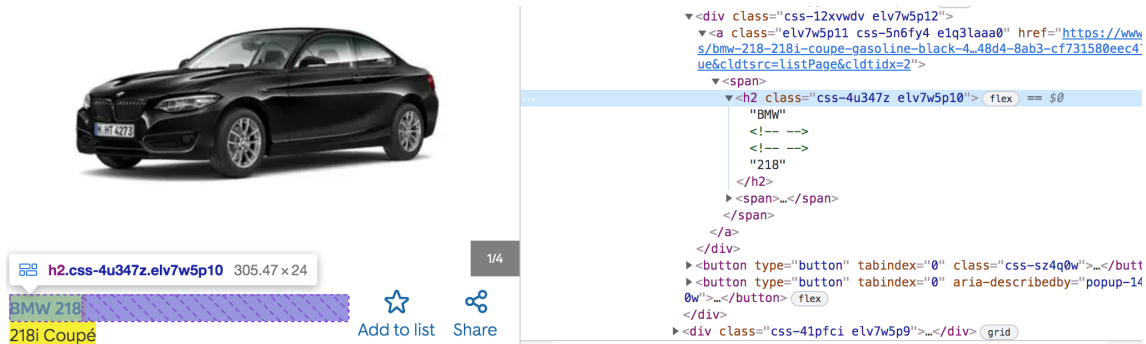
- (g/km)



MOTOR MUNICH, concesionario oficial BMW y MINI  
Contáctanos en: • ES-8223 terrassa

+ Show more vehicles

These different characteristics of each car are identified by the scrapper by interpreting the HTML code of the website. This code is available by using the Google chrome inspector for each desire field:



On the below table you can find all the characteristics scraped from the website as well as its html code:










Characteristics	Definition	Code
Title	Car ad title containing the cars make and model	<code>attrs={"class":'cldt-summary-makemodel sc-font-bold sc-ellipsis'}}.text</code>
Versión	cars version	<code>attrs={"class":'cldt-summary-version sc-ellipsis'}}.text</code>
Price	Ads price in EUR	<code>attrs={"class":'cldt-price sc-font-xl sc-font-bold'}}.text</code>
Kms	Current cars mileage	<code>attrs={"data-type":'mileage'}}.text</code>
Year	Date of registration of the car	<code>attrs={'data-type':'first-registration'}}.text</code>
Hp	Cars power in HorsePower	<code>text=re.compile(r'CV')).text</code>
Prev. Owners	Cars previous owners	<code>attrs={"data-type":'previous-owners'}}.text</code>
Gear type	Type of transmission, i.e. manual, automatic	<code>attrs={"data-type":'transmission-type'}}.text</code>
Fuel type	Fuel type of the vehicle, i.e. petrol, diesel..	<code>attrs={"class":'summary_item_no_bottom_line'}}.text</code>
Fuel consumption	Amount of fuel used by the car in 100 kms (in Liters)	<code>attrs={"data-type":'combined-consumption'}}.text</code>
CO2 emissions	CO2 emissions	<code>attrs={"data-type":'co2-emission'}}.text</code>
Location	Sellers location	<code>attrs={"class":'cldf-summary-seller-contact-city'}}.text</code>

## 2.3 Data scraper

Now we have established the characteristics that we are going to include in the scraper it is time to build a web scraper able to extract the maximum amount of ads.

If we have a look again at the website with all ads we will see there is a maximum of 20 pages to visualize and each page contains 20 ads which means that the total amount of ads to be scrapped is 400. In order to increase the amount of ads but also to obtain new characteristics that are not on the preview fields we are going to play with the filters so we will be able to extract 20 pages and 20 ads per page for each combination of the filters below:

- Car Types: from the basic specifications & Location filters we can filter by the different body types from a list of 9 different types of cars and include this characteristic to the dataset

- ☐  Compact
- ☐  Convertible
- ☐  Coupe
- ☐  Off-Road/Pick-...
- ☐  Station wagon
- ☐  Sedans
- ☐  Van
- ☐  Transporter
- ☐  Other

- Seller Type: also from the basic specifications & Location we can filter by the seller type, Dealer or Private and it will be also included in the dataset.
- Exterior Colour: included in the dataset as well, from exterior filters we can filter colours one by one and obtain all cars from each colour and combinations from above.

- |                                 |                                 |
|---------------------------------|---------------------------------|
| <input type="checkbox"/> Beige  | <input type="checkbox"/> Blue   |
| <input type="checkbox"/> Brown  | <input type="checkbox"/> Bronze |
| <input type="checkbox"/> Yellow | <input type="checkbox"/> Grey   |
| <input type="checkbox"/> Green  | <input type="checkbox"/> Red    |
| <input type="checkbox"/> Black  | <input type="checkbox"/> Silver |
| <input type="checkbox"/> Violet | <input type="checkbox"/> White  |
| <input type="checkbox"/> Orange | <input type="checkbox"/> Gold   |

In order to be able to scrap all the characteristics commented above we are going to describe the process of building the scraper that is going to extract all this information.

On github all this process is in the file 01.Cars\_Scrapper.ipynb. The scraping has been done using BeautifulSoup.

First step is to define the columns of the dataset that we are going to need in order to create the data frame but also where the scraper is going to save the information obtained.

```
#Defining columns for Dealer cars
title = []
version = []
year = []
kms = []
hp = []
gear = []
fuel = []
fuel_cons = []
co2 = []
prev_owners = []
price = []
location = []
colour = []
body_type = []
seller_type = []
```

Having a look to the URL, we see that for every single option in the filter, the URL changes a small part of the link but the rest remains the same, i.e for the body types the code changes only the number after "&body=" being each number a different body type.

```
&cy=E&body=1&atype=C&recommended_sorting_based_id=29e07135-6b02-4c0a-ba49-da018c954028&"
&cy=E&body=6&atype=C&recommended_sorting_based_id=29e07135-6b02-4c0a-ba49-da018c954028&"
cy=E&body=5&atype=C&recommended_sorting_based_id=29e07135-6b02-4c0a-ba49-da018c954028&"
&cy=E&body=12&atype=C&recommended_sorting_based_id=29e07135-6b02-4c0a-ba49-da018c954028&"
&cy=E&body=3&atype=C&recommended_sorting_based_id=29e07135-6b02-4c0a-ba49-da018c954028&"
&cy=E&body=2&atype=C&recommended_sorting_based_id=29e07135-6b02-4c0a-ba49-da018c954028&"
&cy=E&body=4&atype=C&recommended_sorting_based_id=29e07135-6b02-4c0a-ba49-da018c954028&"
&cy=E&body=13&atype=C&recommended_sorting_based_id=29e07135-6b02-4c0a-ba49-da018c954028&"
&cy=E&body=7&atype=C&recommended_sorting_based_id=29e07135-6b02-4c0a-ba49-da018c954028&"
```

The same happens with the other filters, so after defining the different options needed in the scrapper we need to do a numpy array with the different possibilities for each filter, i.e for body types ( As the scraper will add the code for each colour I have created a dictionary in order to replace the codes for the colour description later on):

```
body = np.array([1,6,5,12,3,2,4,13,7])
body

array([ 1,  6,  5, 12,  3,  2,  4, 13,  7])

body_type_dictionary = {'1': 'small',
                        '6': 'sedan',
                        '5': 'familiar',
                        '12': 'minivan',
                        '3': 'coupe',
                        '2': 'cabrio',
                        '4': 'suv',
                        '13': 'van',
                        '7': 'other'}
```

Also, in order to be able to scrap the ads across all the 20 pages we need to input a numpy arange where we tell the scrapper the page to start, the page to stop and the steps, in this case one by one:

# Max page per url = 20 (start: page 1, stop: page 21 -1, step: 1 by 1 as the url changes from page=n to page=n+1 from one page to the following

```
all_pages = np.arange(1,21,1)
```

Finally, we will add all these variables to a for boucle and will apply BeautifulSoup in order to find all the characteristics desired and add them with a “.append” to the columns list previously created. (We perform this extraction two times, one for Dealers and another one for Private sellers)

```
for car_colours in colours:
    for car_types in body:
        for all_cars in all_pages:

            all_cars_page = requests.get("https://www.autoscout24.es/1st/?sort=age&desc=1&car_type=" + car_types + "&car_color=" + car_colours)
            all_cars_soup = BeautifulSoup(all_cars_page.text, "lxml")
            all_cars_ads = all_cars_soup.find_all('div', attrs={"class": "cl-list-element"})

            sleep(randint(2,10))

            for element in all_cars_ads:
                try:
                    Title = element.find('h2', attrs={"class": "cldt-summary-makemodel sc-f"})
                except:
                    Title = ""
                title.append(Title)

                try:
                    Version = element.find('h2', attrs={"class": "cldt-summary-version sc-f"})
                except:
                    Version = ""
                version.append(Version)

                try:
                    Year = element.find('li', attrs={"data-type": "first-registration"}).text
                except:
                    Year = ""
                year.append(Year)
```

Last step is to create the Data Frames and merge both Dealers and Privates and save them to a csv for a future analysis.





```
#Splitting the title into Brand and Model

#There are some Brands that are built with two words so before splitting the title into
long_brands = ["Land-Rover", "Alfa-Romeo", "Aston-Martin"]
df["Title"] = df["Title"].str.replace("Land Rover", long_brands[0])\
.str.replace("Alfa Romeo", long_brands[1])\
.str.replace("Aston Martin", long_brands[2])

# We also need to transform the Model column to standarize the models across some Brand

#Mercedes -> c 220 ->> clase C

#Bmw -> 320/330 ->> serie 3
#      X5 ->> X5
```

```
df.Model[(df["Brand"] == "BMW") &
(~df["Model"].str.contains("X|Z|i|M|Active", na=False))] = df["Model"].str[:1] +
" Series"
```

- Province: another important column to create is the Province. The field scrapped City\_ZIP contains the information about the City where the ad is published and the ZIP Code. Using the ZIP code and reading the html from wikipedia ("Anexo:Provincias de España por código postal") to a pandas dataframe we can find the relation between the ZIP Code and the Province so merging these data frames on the ZIP Code we can include the Province in our database

```
#Defining the CCAA *****

zip_codes_html = pd.read_html('https://es.wikipedia.org/wiki/Anexo:Provincias_de_Espa%C3%A1%C3%B1a_por_c%C3%B3digo_postal')
zip_codes = zip_codes_html[0]
zip_codes["zip_first_digits"] = zip_codes["Código postal"].apply('{:0>2}'.format)
zip_codes
```

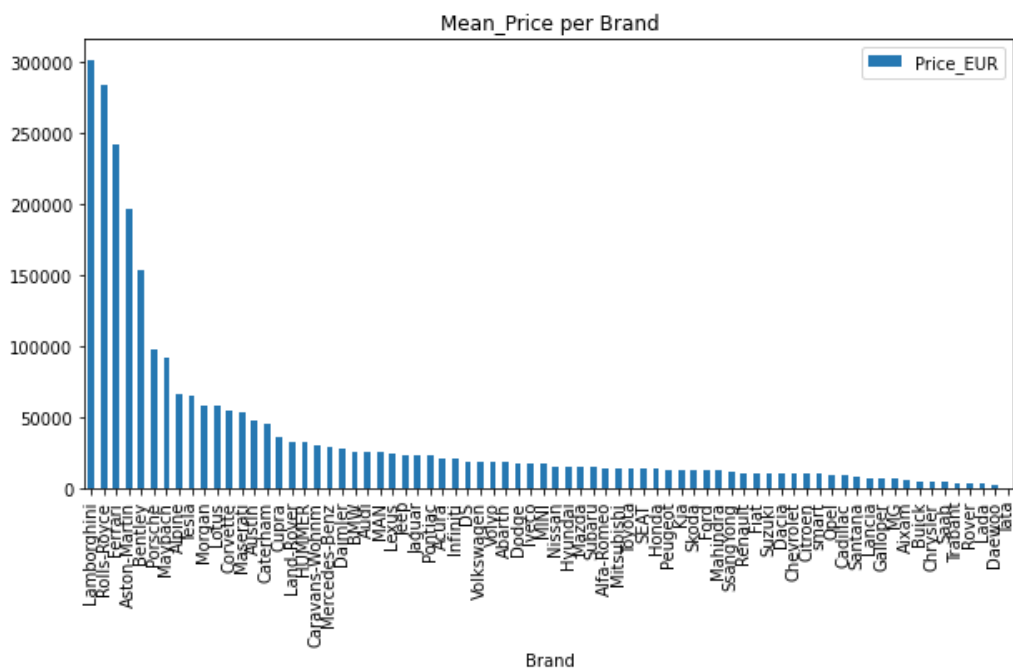
	Código postal	Provincia	Código Ministerio del Interior	zip_first_digits
0	1	Álava	VI	01
1	2	Albacete	AB	02
2	3	Alicante	A	03

## 4.Data Analysis

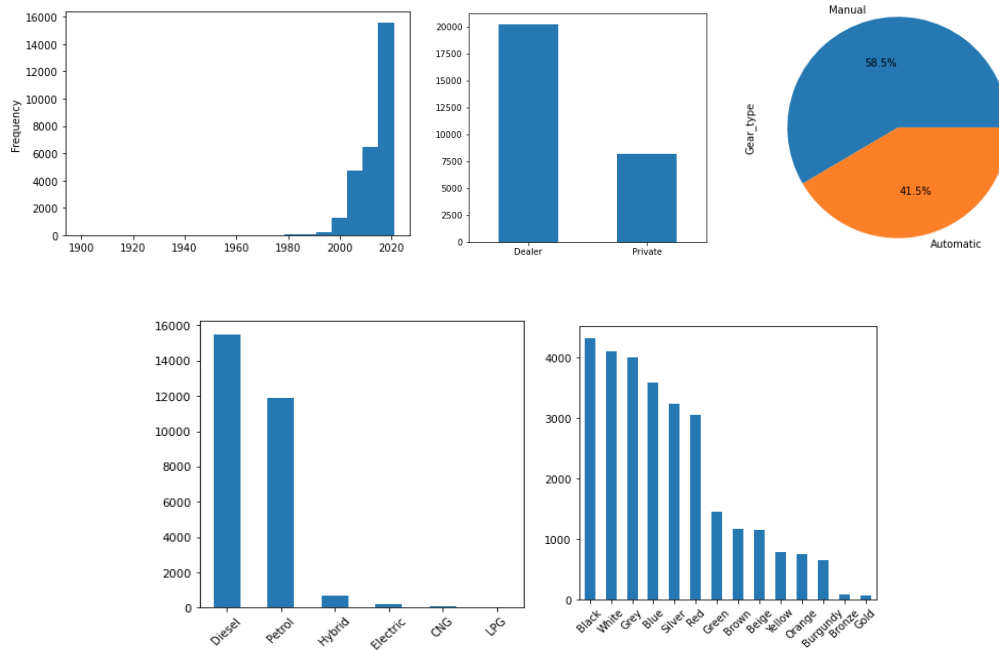
The project will continue with an exploratory analysis of each feature. It is important to analyze all the data extracted to see the data distribution along with the different information obtained. In order to have better results when predicting it is important to train the model with many examples but also is important the quality of the data , not only the quantity so at the end of this point we will check for outliers in order to avoid inconsistencies and errors.

## 4.1 Data visualization

The first visualizations we will perform will be the distribution of the data across the different Brands and also the analysis of the Target Feature, which is the Price. For this last analysis we will compare the mean price along all Brands. We can see how the Luxury Brands like Lamborghinis, Rolls-Royces or Ferraris have higher mean Prices.

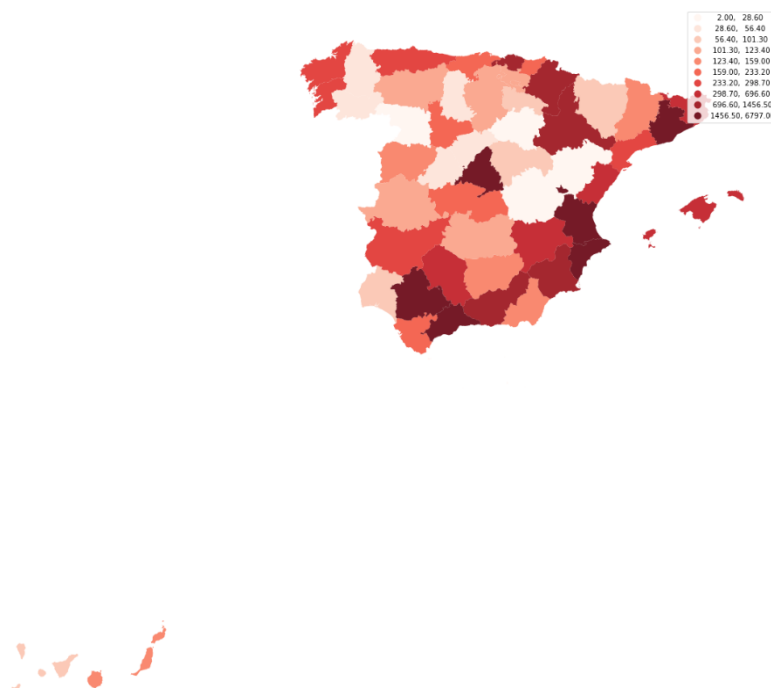


After this first analysis we will continue with an analysis of the most representative features to display in the model as for example could be the Year, Gear Type, Colour, Seller Type or even the Province.



In order to be able to better visualize the data along the different Provinces in Spain a new module from python is needed, geopandas. Geopandas will help us to show using a colour map the distribution of the dataset by Province.

I created a new data frame “df\_province” with just the name of the Province and the two first digits of the ZIP Code and applied a value count for each ZIP Code. After that we will merge this new data frame with a .shp file that will provide the designated coordinates for each province and will allow us to plot our map.



## 4.2 Outliers

Using the `df.describe().T` we can get statistical data including the mean, standard deviation and percentile of the numerical values of the Series or DataFrame.Methodology

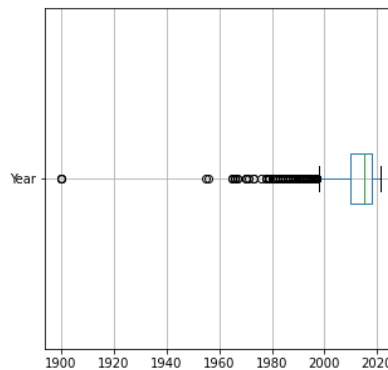
```
In [20]: df.describe().T
```

```
Out[20]:
```

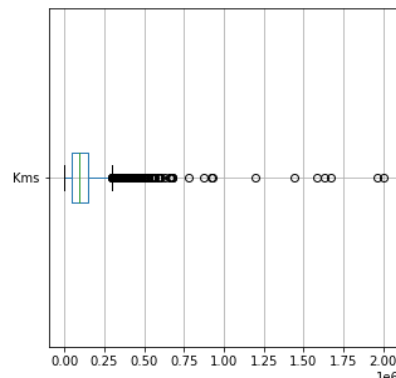
	count	mean	std	min	25%	50%	75%	max
<b>Year</b>	28446.0	2013.185123	6.185965e+00	1900.0	2010.0	2015.0	2018.0	2.021000e+03
<b>Kms</b>	28446.0	107172.846130	7.999218e+04	0.0	48103.0	95635.0	149221.5	1.999999e+06
<b>Hp</b>	28446.0	172.762849	1.075914e+02	1.0	116.0	140.0	190.0	4.759000e+03
<b>Fuel_cons</b>	28446.0	5.992319	2.627293e+00	0.0	4.5	5.3	6.9	9.700000e+01
<b>Doors</b>	28446.0	4.558180	8.297342e-01	3.0	5.0	5.0	5.0	5.000000e+00
<b>ZIP</b>	28446.0	534902.241545	3.034993e+07	1001.0	11130.0	28260.0	35118.0	1.810018e+09
<b>Price_EUR</b>	28446.0	23664.862300	5.085818e+04	220.0	9000.0	14990.0	22990.0	4.350545e+06

Next step is to check for outliers for the numerical features described above:

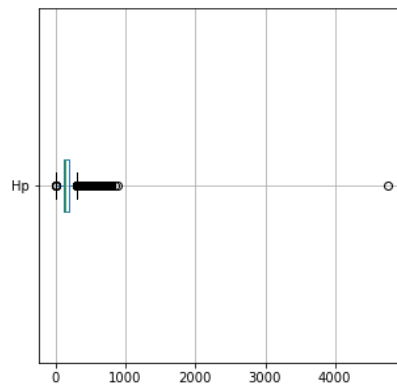
- Year: we identified some cars dated in 1900. After a quick analysis of them we assume these cars have not been correctly specified so we will proceed to delete them using command drop from the data frame



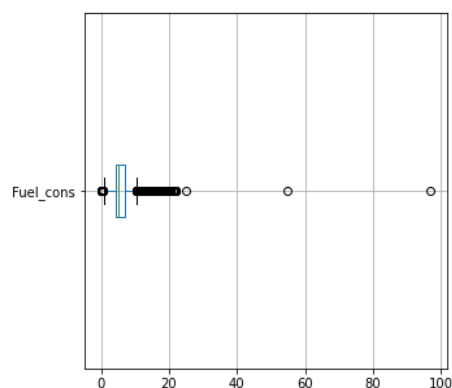
- Kilometers: from the describe command we can see that the minimum value for the feature kilometers is 0 Kms while the maximum is 1,9M Kms. Even though it is possible to see cars with a mileage over 1M, it is not very usual and we will proceed to drop them from the case study.



- Power(Hp): we have easily spotted an outlier in terms of power and it will be dropped from the data frame as there is a car with more than 4000 Hp.



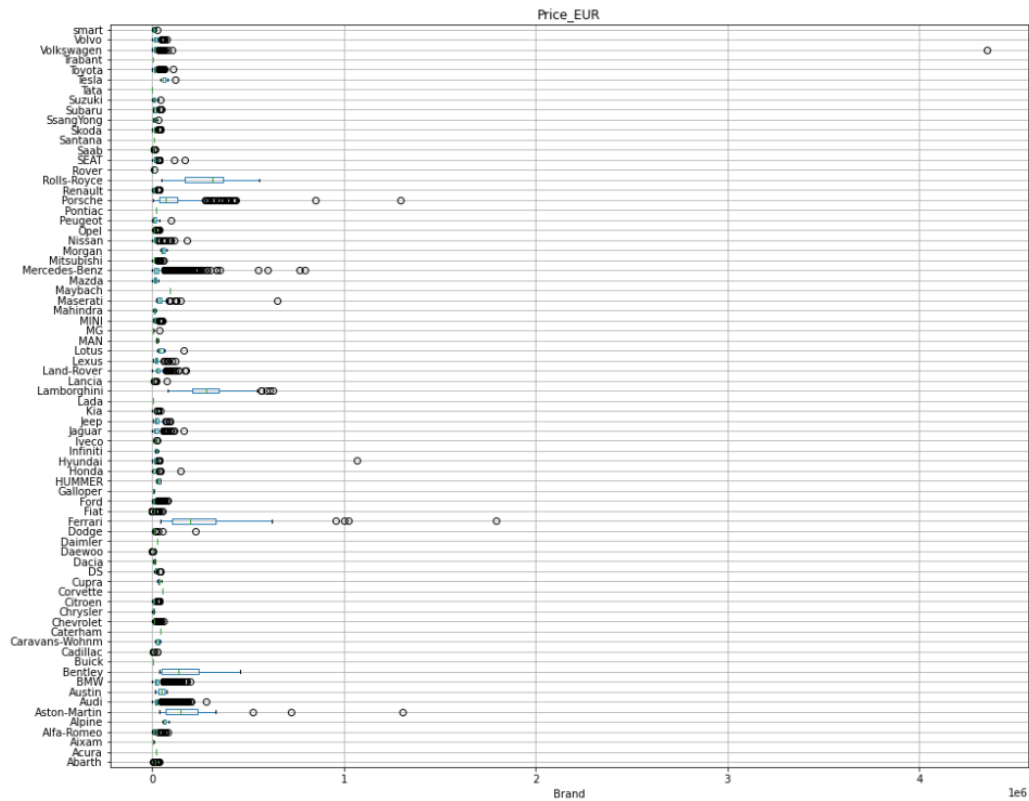
- Fuel Consumption: cars with a fuel consumption over 40 L/100km will be considered as outliers and will be dropped from the data frame. Cars with a fuel consumptoin equal to "0" and a fuel type not equal to "Electric" will be filled with the fuel consumption mean value of the data frame.



- Price: this is our target feature so having outliers in this feature could have an important impact on the accuracy of the model. In order to better check the data I have plotted the data by Brands as we could notice easily and in a faster way the inconsistent values for each car maker.

Outliers have been found in car makers like Volkswagen with cars with prices over €4,0M or over €1,0M in Hyundai will be removed from the data frame. However, outliers found in luxury car makers like Porsche, Ferrari or Aston Martin won't be removed as they could be related to a specific model or limited edition that will supposedly have an increased value.

Boxplot grouped by Brand



## 5. Methodology

Once we have cleaned the data set and performed an exploratory analysis of it it is time to describe the methodology we have followed to reach the target, a model able to predict used cars values.

### 5.1 Feature engineering

Feature engineering is the preprocessing step that will transform our cleaned data frame into features that can be used in our predictive model.


First step I did was to divide the data frame into numerical features and the rest. Numerical features like "Year", "Kms", "Hp", "Fuel\_cons", "Doors" or "Price\_EUR" are okay for the predictive model so we need only to apply feature engineering to features types not equal to numerical. Features like "Brand", "Model", "Type", "Gear\_type" and so on will need a transformation in order to be included in the Machine Learning Algorithm.

In order to proceed to this transformation of categorical features we will proceed in two different ways (Get Dummies or encoder) depending if they possess too many unique values or not, also known as high or low cardinality.

#### 1. Dummies

The function `get_dummies` will help us to transform categorical variables to one hot variable which means converting categorical independent variables to multiple binary columns, where 1 indicates the observation belonging to that category. As this process will generate as many columns as unique values per feature will apply it only for low cardinality features.

This method will be applied to 'Gear\_type', 'Fuel\_type', 'Type', and 'Seller'.



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

#### 2. Encoder for high cardinality features

For those variables with high cardinality as they might be "Brand", "Model", "Colour" and "Province" with more than 10 unique values we will proceed with another technique called Target Encoder from scikit-learn that doesn't add to the



dimensionality of the dataset and will replace a feature's categories with some number derived from the target, in this case the Price.

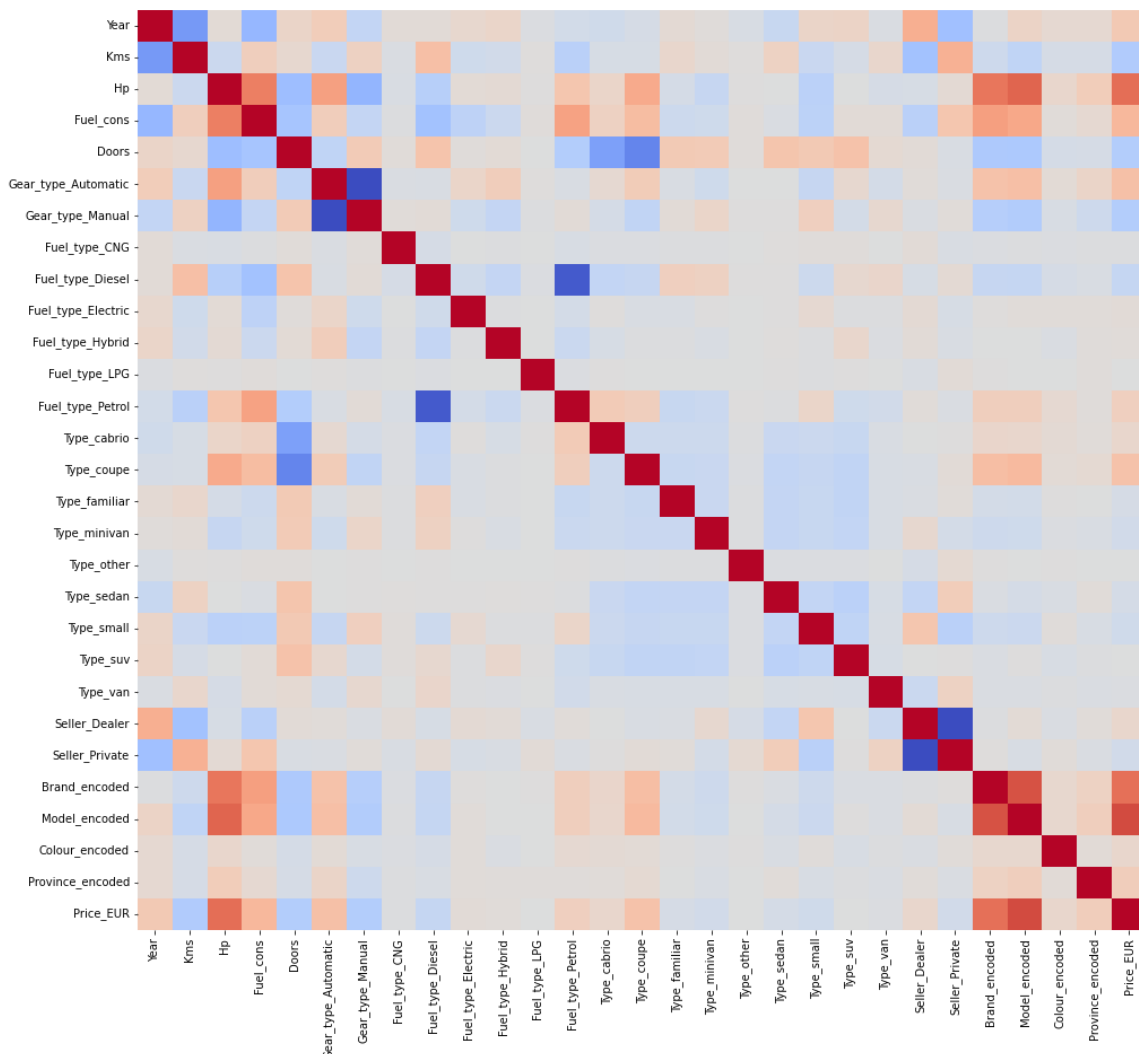
## 5.2 Model

During this chapter we will build the different models using different Machine Learning techniques to be able to select which model is more suitable for the project.

First step is to set the Features Names (X) which are all the features in the data frame excluding the variable to predict and the Target (y) which is the variable we want to predict, in this case the Price.

```
In [3]: X = df[df.columns[:-1]].to_numpy()
        y = df[df.columns[-1]].to_numpy()
```

The image below shows the correlation between all the features in the dataframe. On the one hand we can see how the features “Brand\_encoded” , “Model\_encoded” and “HP” positively correlates with the target feature(“Price\_EUR”). On the other hand, features like “Kms” or “Doors” negatively correlate with “Price\_EUR”.



After this quick view of the correlations it is time to describe the models we are going to test for this project. For that we will need to do a train test split from sklearn.model\_selection setting the test size to 20% of the dataframe prepared after the feature engineering “04.cars\_features\_def.csv”

```
from sklearn import model_selection
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size = .2)
```

After that we just need to import the models we want to perform and perform the model fit and predict.

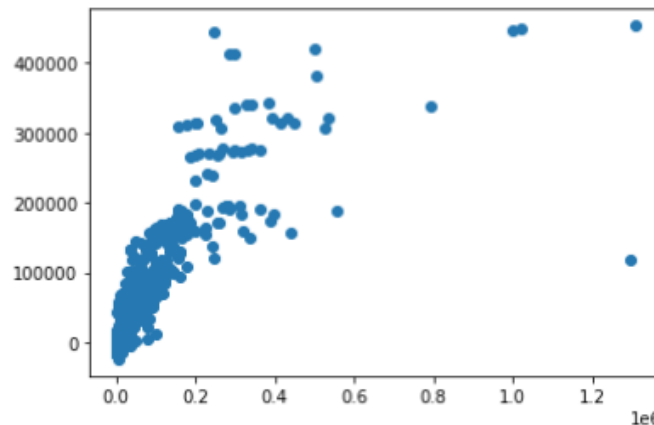
```
from sklearn.ensemble import RandomForestRegressor
```

```
rf = RandomForestRegressor()
rf.fit(X_train, y_train)
```

```
RandomForestRegressor()
```

```
y_pred = rf.predict(X_test)
```

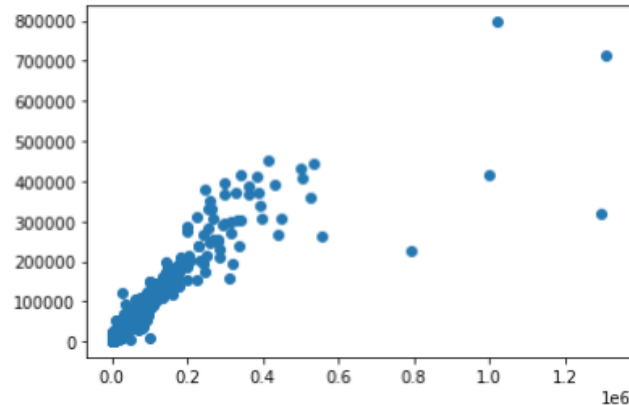
- Models studied:
  - Linear Regression: is the supervised Machine Learning model in which the model finds the best fit linear line between the independent and dependent variable i.e it finds the linear relationship between the dependent and independent variable (“Linear Regression | Introduction to Linear Regression for Data Science”)



```
from sklearn import metrics
print('R2:', lr.score(X_test, y_test))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Percentage Absolute Error:', metrics.mean_absolute_percentage_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
R2: 0.7488669313637286
Mean Absolute Error: 7032.121581333472
Mean Percentage Absolute Error: 0.5682066650628621
Mean Squared Error: 484293734.1495857
Root Mean Squared Error: 22006.67476357084
```

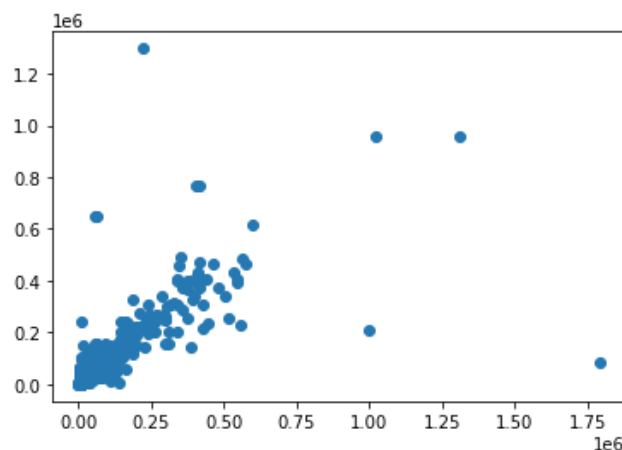
- Random Forest: is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression (“Random Forest | Introduction to Random Forest Algorithm”)



```
from sklearn import metrics
print('R2:', rf.score(X_test, y_test))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Percentage Absolute Error:', metrics.mean_absolute_percentage_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
R2: 0.8844138690892618
Mean Absolute Error: 2790.6276300055138
Mean Percentage Absolute Error: 0.13340183396471458
Mean Squared Error: 222900310.41566858
Root Mean Squared Error: 14929.846295781768
```

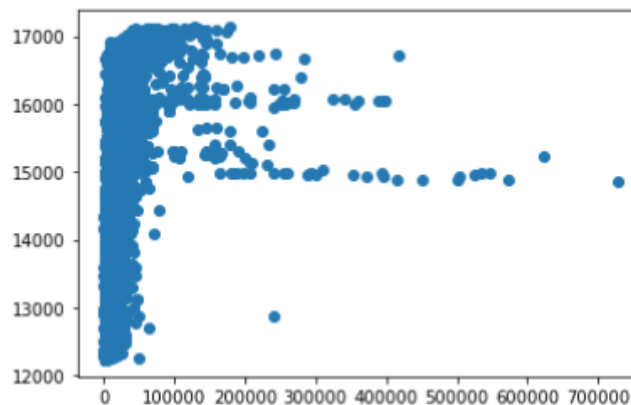
- Decision Tree Regressor: are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model (“1.10. Decision Trees — scikit-learn 1.0.2 documentation”)



```
print('R2:', dtr.score(X_test, y_test))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Percentage Absolute Error:', metrics.mean_absolute_percentage_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
R2: 0.7454113659418231
Mean Absolute Error: 3988.9943032427695
Mean Percentage Absolute Error: 0.1716787713955896
Mean Squared Error: 490957566.5586766
Root Mean Squared Error: 22157.562288272522
```

- Support Vector Regression: Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points (“Support Vector Regression”)



```
print('R2:', svr.score(X_test, y_test))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Percentage Absolute Error:', metrics.mean_absolute_percentage_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
R2: -0.021591190278107852
Mean Absolute Error: 13847.555319996176
Mean Percentage Absolute Error: 0.8119046518409502
Mean Squared Error: 1970071942.3401635
Root Mean Squared Error: 44385.49247603504
```

- Evaluation Metrics:

There are different evaluation metrics for regression models. For this project we have analyzed the R2 coefficient plus four error metrics usually used for evaluation of the performance of a regression model, which are the Mean Absolute Error, Mean Percentage Absolute Error, Mean Squared Error and the Root Mean Squared Error. We will base our selection of the model on the R2 and if we have a look at the values obtained in each model we can see that the model with a better R2 is the Random Forest Regressor (in general, the higher the R2, the better the model fits the data).

Basically the  $R^2$  is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination and represents the percentage of the response variable variation that is explained by a linear model.

$$R\text{-squared} = \text{Explained variation} / \text{Total variation}$$

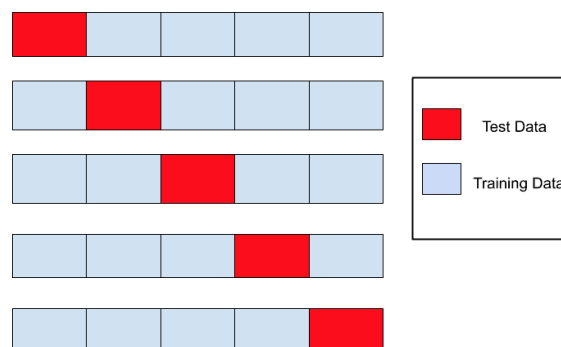
R-squared values are always between 0 and 100%, where a 0% indicates that the model explains none of the variability of the response data around its mean and a 100% indicates that the model explains all the variability of the response data around its mean (“Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit?”)

From a further analysis we will exclude the Support Vector Regression model due to its poor performance

- K Fold Cross Validation

In order to perform a deeper analysis of our models we are also going to apply a K Fold Cross Validation which is another evaluating estimator performance that consists on splitting the data set on K number of groups (folds) where each fold is used as a testing set at some point so if we split the dataset into 5 folds, in the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 5 folds have been used as the testing set. This technique will also help us to avoid overfitting as it holds out part of the available data as a test set (“K-Fold Cross Validation. Evaluating a Machine Learning model can... | by Krishni”)

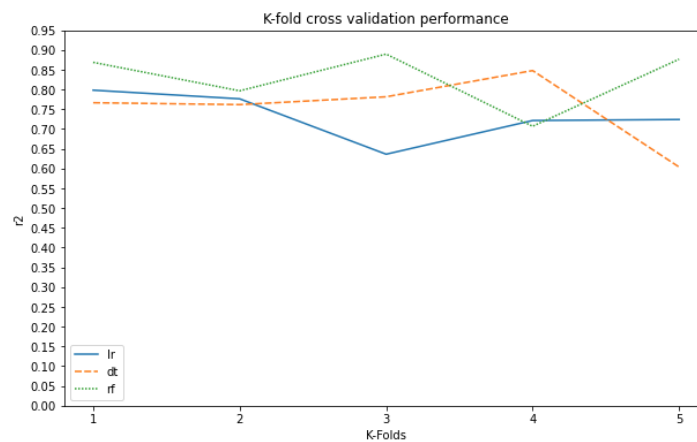
**K-fold cross validation**



Steps to follow: (“A Gentle Introduction to k-fold Cross-Validation”)

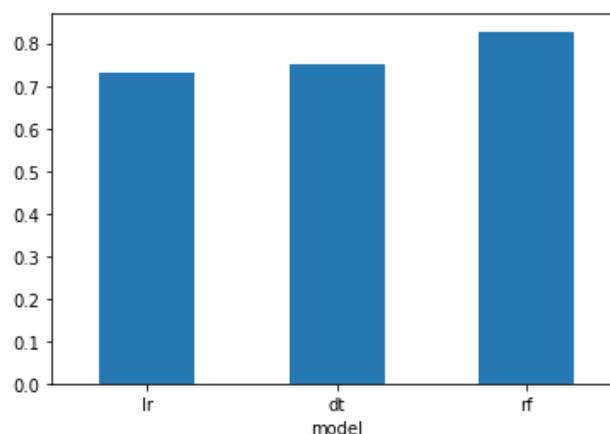
1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
  1. Take the group as a hold out or test data set
  2. Take the remaining groups as a training data set
  3. Fit a model on the training set and evaluate it on the test set
  4. Retain the evaluation score and discard the model

#### 4. Summarize the skill of the model using the sample of model evaluation scores



According to the K fold cross validation we can confirm that the Random Forests is the best model out of the three models studied during this analysis as it gives the best score.

Mean R2



### 5.3 Model Optimization

Now we have selected the best model for our project it is time to Set Up Hyperparameter Distributions. Machine learning models are built with parameters which are the internal coefficients set by training or optimizing the model on a training dataset. Parameters are learned automatically while hyperparameters are set manually to help guide the learning process. We can conclude saying that a Hyperparameter is a model configuration argument specified by the developer to guide the learning process for a specific database (“Hyperparameter Optimization With Random Search and Grid Search”)

The Randomized Search CV will help us to find an optimal set of hyperparameters for our machine learning model. It is an algorithm that trains and evaluates a series of models by taking random draws from a predetermined set of hyperparameter

distributions. The algorithm picks the most successful version of the model it's seen after training N different versions of the model with different randomly selected hyperparameter combinations, leaving you with a model trained on a near-optimal set of hyperparameters

We need to pass arguments to the RandomizedSearchCV like the dictionary of parameter distributions, the number of iterations for random search to perform, and the number of folds for it to cross validate over. Thus, The number of cross validation folds you choose determines how many times it will train each model on a different subset of data in order to assess model quality. The total number of models random search trains is then equal to  $n\_iter * cv$  ("Tune Hyperparameters with Randomized Search")

```
def perfect_model(X, y):
    model_algo = {
        'Random_forest':{
            'model': RandomForestRegressor(),
            'params': {
                'n_estimators': [x for x in range(20,150,20)],
                'max_features': ['auto', 'sqrt'],
                'max_depth': [x for x in range(5,35,5)],
                'min_samples_split': [2, 5, 10, 15, 100],
                'min_samples_leaf': [1, 2, 5, 10]}}}

    score = []
    cv = 5
    for algo_name, config in model_algo.items():
        rs = RandomizedSearchCV(config['model'], config['params'], cv=cv, return_train_score=False)
        rs.fit(X_train,y_train)
        score.append({
            'model': algo_name,
            'best_score': rs.best_score_,
            'best_params': rs.best_params_})

    result = pd.DataFrame(score,columns=['model','best_score','best_params'])
    print(result.best_params_.tolist())
    return result
```

```
perfect_model(X, y)
```

```
[{'n_estimators': 140, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 20}]
```

The results are the best parameters the algorithm found from all  $n\_iter$  candidate models and now we have to pass these arguments to the model and fit the model with them.

```
rf = RandomForestRegressor(n_estimators=140, min_samples_split=5, min_samples_leaf=1, max_depth=20)
rf.fit(X_train,y_train)
rf.score(X_test,y_test)
```

```
0.8862078528255388
```

```
cross_val_score(RandomForestRegressor(n_estimators=140, min_samples_split=5, min_samples_leaf=1, max_depth=20), X_train, y_train, cv=5)
array([0.84364219, 0.64367739, 0.95721151, 0.81705202, 0.71108841])
```

Last step is to save the model to a pickle file as we are going to use this model for the front-end

## 6.Frontend

The project concludes with a front-end application that will allow the users to interact with the model previously saved. This application has been developed using streamlit, an open-source python framework for building web apps for Machine Learning and Data Science.

- Layout

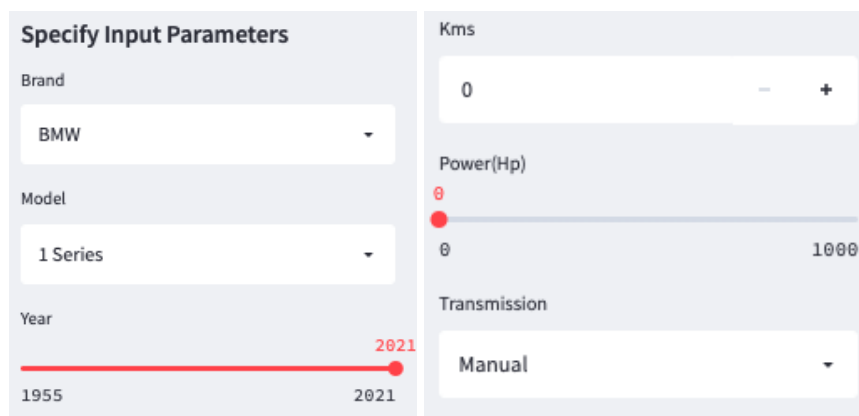
The Car Genius app will predict any used car price according to the specificities input by the user. The app is set up with four different blocks we can easily appreciate from the home page:

1. Title: A header with the logo and the name of the app on the top of the page. It also includes a briefly description of the app's functionality.



Car Genius will help you to predict Used Cars Values for the Spanish Market using a Machine Learning Algorithm updated up to 2021 to arrive at a fair value in just a few clicks!

2. User Inputs: located at the left of the page is the interactive part of the app where the users can specify the characteristics of the car they would like to predict the price.


 The image shows the "Specify Input Parameters" section of the app. It contains four input fields: "Brand" with a dropdown menu showing "BMW", "Model" with a dropdown menu showing "1 Series", "Year" with a range slider from 1955 to 2021 (the slider is set to 2021), and "Kms" with a numeric input field showing "0". Below these are "Power(Hp)" with a range slider from 0 to 1000 (the slider is set to 0) and "Transmission" with a dropdown menu showing "Manual".

3. Output: The main block. As a result of the characteristics specified by the user the app will return the car maker Logo, dataframe with a summary of the characteristics input and in the line below the predicted price for that car.



## Specified Input parameters

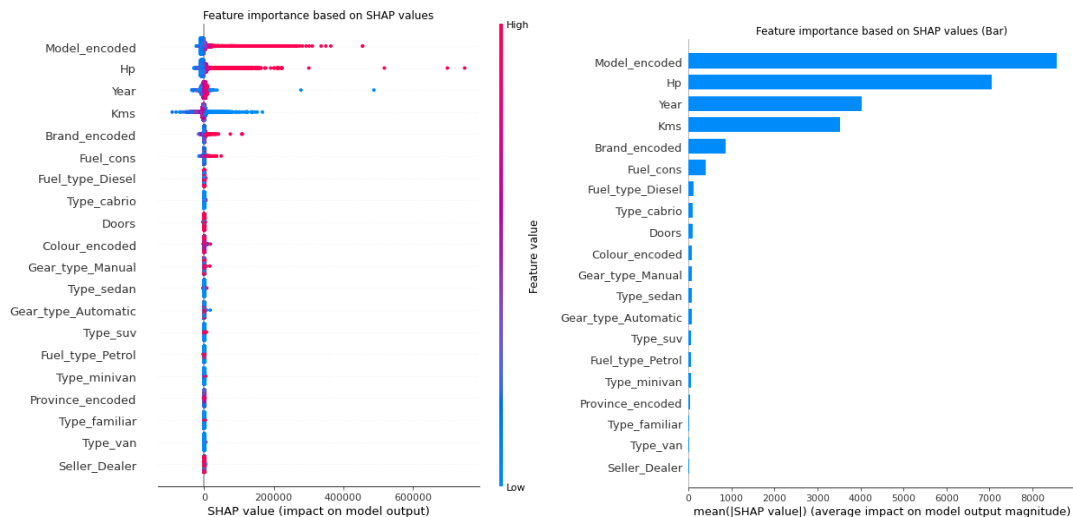


	Brand	Model	Type	Year	Kms	Hp	Gear_type	Fuel_type	Fuel_cons	
0	BMW	1 Series	small	2021	0	0	Manual	Diesel	6	

## The predicted value for this cars is:

	Price_EUR
0	€20,601

4. Information: The lower part of the page shows the shap values giving information about the features impact to the model.



- Coding

The front end code has been saved in a .py file (CarGenius.py) requested by Streamlit in order to be able to build the web page and the first thing we need to do after importing the modules is to import the pickle file saved after the model optimization.

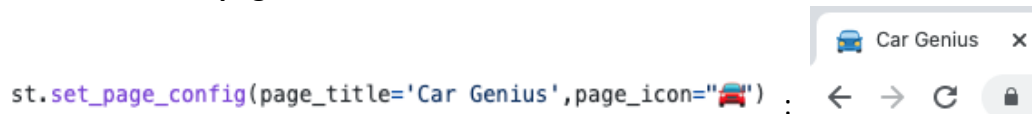
```

1 import streamlit as st
2 import pickle
3 import pandas as pd
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from category_encoders import TargetEncoder
7 import bz2
8 import _pickle as cPickle
9 import shap
10
11
12 compressed_model = bz2.BZ2File("Files/RF_price_predicting_model.pkl.pbz2", 'rb')
13 model = cPickle.load(compressed_model)

```

After that we are going to start to set up the web page:

1. Describe the **page title and the icon**



2. Specify the **Title logo** and the description using the command `st.image` and `st.write`
3. Create the **sidebar** by using the `st.sidebar` and describing the header "Specify Input Parameters"
4. Create the **Main blocks Title** "Specified Input Parameters"
5. **Import the 'cars\_final\_def.csv'** before the feature transformation and set the target and the features names (X,y). This is important because the user is going to input the specificities of one car (Car make, Model, Transmission, Hp, Province...) and then we will add this row to the data frame imported and apply the feature engineering (If we apply the feature engineering to the row specified by the user we will miss the dummy columns not selected by the user and the encoded columns will not be performed correctly)
6. Defining the **functions** for the **main Cars Make Logos** to be able display them after the users selection.
7. By defining `def user_input_features` we are **Building the sidebar** including all filters according to the features of the model and creating a data frame with the features selected by the user in order to A) Add the row to the dataframe to be able apply the feature engineering B) display the data frame in the main block "Output"

```
MODEL = st.sidebar.selectbox('Model', np.sort(cars_final[cars_final.Brand == BRAND].Model.unique()), index=0)

YEAR = st.sidebar.slider('Year', int(X.Year.min()), int(X.Year.max()), 2021)
KMS = st.sidebar.number_input('Kms', 0, 1000000, 0, step = 1)
HP = st.sidebar.slider('Power(Hp)', 0, 1000, 0)
TRANSMISSION = st.sidebar.selectbox('Transmission', X.Gear_type.unique())
FUEL = st.sidebar.selectbox('Fuel type', cars_final.Fuel_type.unique(), index=0)

CONS = st.sidebar.slider('Fuel cons', int(X.Fuel_cons.min()), int(X.Fuel_cons.max()), int(X.Fuel_cons.mean()))
DOORS = st.sidebar.slider('Doors', int(X.Doors.min()), int(X.Doors.max()), 5)
COLOUR = st.sidebar.selectbox('Colour', np.sort(cars_final.Colour.unique()), index=12)
TYPE = st.sidebar.selectbox('Type', cars_final.Type.unique(), index=0)

PROVINCE = st.sidebar.selectbox('Province', np.sort(cars_final.Province.unique()), index=29)
SELLER = st.sidebar.radio("Seller", ("Dealer", "Private"))
```

8. In the function main we are **applying machine learning** to the car options specified. We apply the feature engineering to the data frame created with the users input, after that we take only the row with the users input (df\_pred) and apply the model.predict to it. Last step would be to show the predicted price.

```
df_pred = df[:1]

# Apply Model to Make Prediction

prediction = pd.DataFrame(model.predict(df_pred))
prediction.columns = ['Price_EUR']
prediction['Price_EUR'] = prediction['Price_EUR'].map('{:,.0f}'.format)

# Apply Model to Make Prediction

prediction = pd.DataFrame(model.predict(df_pred))
prediction.columns = ['Price_EUR']
prediction['Price_EUR'] = prediction['Price_EUR'].map('{:,.0f}'.format)
```

9. Finally, the last block with the **model information** shows the shap values that explains the features importance to the model by importing the shap figures previously saved using the st.image.

## 7.Requirements

### 7.1 Dependencies

For the correct execution of the project you will need the following dependencies installed on your anaconda environment:

- Pandas
- Numpy
- Matplotlib
- Geopandas
- Bs4
- Lxml
- Seaborn
- Scikit-learn
- Pickle
- Streamlit
- Shap
- Heroku

### 7.2 User Manual

The app has been designed to be very easy to use . When you access the webpage <https://cargenius.herokuapp.com/> you will see:

- Header: App logo and a short description about it
- Sidebar: located on the left part of the page will allow users to specify their car options to allow the algorithm to make a prediction.
- Specified Input parameters summary and the result of the prediction
- Model predictions explanation using SHAP values

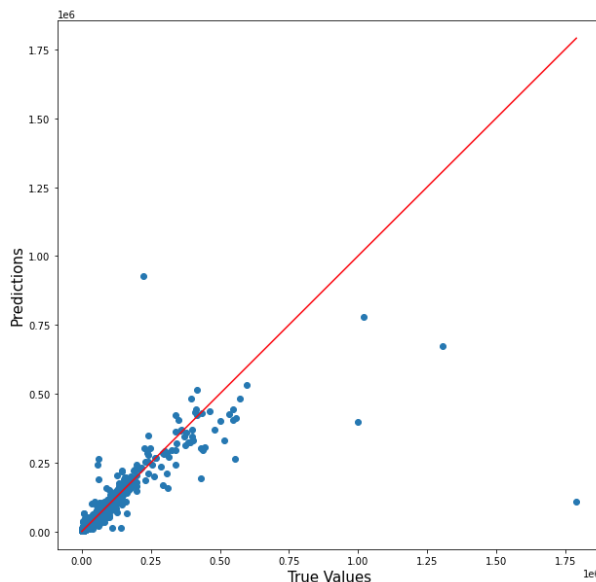
The user only needs to input the options of the car they want to make a prediction of by using the sidebar on the left of the screen "Specify Input Parameters". The algorithm will run automatically and the predicted price for the car will appear under the "The predicted value for this car is".

## 8.Conclusions

The purpose of this project was to create an accurate algorithm able to predict the used cars value for the Spanish Market and the Car Genius app serves the purpose it was developed for in a very friendly and easy to use format.

The project has been developed end to end, from the raw data extraction to an interactive app where the users can input the parameters to the algorithm to make predictions.

During the project, more than 30,000 used cars ads were studied and although it is not a large number of ads for a Machine Learning project, the predictions obtained have been quite good, reaching an 89% accuracy using the Random Forest Model.



The most time consuming part of the project was the data source selection and the data cleaning. Once you decide what your work is going to be about you have to start thinking about how you will get the data for the study and find those sites with the relevant information and open to be scrapped. After that, data cleaning is one of the keys of the project, with quality data and data cleaned accordingly (formattings, outliers, re-groupings, checking inconsistencies...) the model will be able to provide better results.

For the future, it will be interesting to monitor these used car values on a temporary basis, in order to help buyers and sellers to choose the best moment for a sale or a purchase and help the users to see if a car model is being appreciated or depreciated.

## 10. References

### Works Cited

- “Anexo:Provincias de España por código postal.” *Wikipedia*,  
[https://es.wikipedia.org/wiki/Anexo:Provincias\\_de\\_Espa%C3%B1a\\_por\\_c%C3%B3digo\\_postal](https://es.wikipedia.org/wiki/Anexo:Provincias_de_Espa%C3%B1a_por_c%C3%B3digo_postal). Accessed 16 January 2022.
- “AutoScout24.” *Wikipedia*, <https://es.wikipedia.org/wiki/AutoScout24>. Accessed 16 January 2022.
- “A Gentle Introduction to k-fold Cross-Validation.” *Machine Learning Mastery*, 23 May 2018, <https://machinelearningmastery.com/k-fold-cross-validation/>. Accessed 16 January 2022.
- “Hyperparameter Optimization With Random Search and Grid Search.” *Machine Learning Mastery*, 14 September 2020,  
<https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/>. Accessed 16 January 2022.
- “K-Fold Cross Validation. Evaluating a Machine Learning model can... | by Krishni.” *DataDrivenInvestor*, 16 December 2018,  
<https://medium.datadriveninvestor.com/k-fold-cross-validation-6b8518070833>. Accessed 16 January 2022.
- “Linear Regression | Introduction to Linear Regression for Data Science.” *Analytics Vidhya*, 25 May 2021,  
<https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-your-first-machine-learning-model-linear-regression/>. Accessed 16 January 2022.

“1.10. Decision Trees — scikit-learn 1.0.2 documentation.” *Scikit-learn*,

<https://scikit-learn.org/stable/modules/tree.html>. Accessed 16 January 2022.

“Random Forest | Introduction to Random Forest Algorithm.” *Analytics Vidhya*, 17 June 2021,

<https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/>  
. Accessed 16 January 2022.

“Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit?”

*Minitab Blog*, 30 May 2013,

<https://blog.minitab.com/en/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit>. Accessed 16 January 2022.

“Tune Hyperparameters with Randomized Search.” *James LeDoux*, 1 June 2019,

[https://jamesrledoux.com/code/randomized\\_parameter\\_search](https://jamesrledoux.com/code/randomized_parameter_search). Accessed 16 January 2022.

“Unlocking the True Power of Support Vector Regression.” *Towards Data Science*, 3

October 2020,

<https://towardsdatascience.com/unlocking-the-true-power-of-support-vector-regression-847fd123a4a0>. Accessed 16 January 2022.