Department of Computing and Mathematics

M.Sc. in Artificial Intelligence

Assessment:

AUTOTRADER CARPRICE PREDICTION

Module: **MACHINE LEARNING CONCEPTS**

Semester 1: 2022/2023

Module: **6G7V0017_223_1F**

Name: ADERIYE ABIOLA MIRACLE

Student ID: @22545807

Tutor: LUCIANO GERBER

ROCHELLE TAYLOR

Chapter 1.0 DATA DOMAIN UNDERSTANDING AND EXPLORATION

1.1 Importing Libraries

Imported Pandas library as pd and numpy as np for my data manipulation and analysis.

CODE

```
# manipulation of data
import pandas as pd
import numpy as np
```

1.2Importing More Libraries To Better The Data

I imported the seaborn and matplotlib libraries.

CODE

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='ticks',color_codes=True)
```

%matplotlib inline enables the visualization to be seen in my notebook's output cells.
The code sns.set(style='ticks',color codes=True) changes the plot style to "ticks" and allows colour codes for plots generated by the seaborn. The reason for this is to make my plots more easily visualizable and for better understanding.

1.3Importing my moodles from Scikit-Learn Library

I used several modules from the scikit-learn library.

- **RandomForestRegressor** for constructing and fitting a random forest regression model.
- **train_test_split** is used for splitting the Autotrader's dataset into training and testing sets.
- **GridSearchCV** is for finding grid search on my model with several hyperparameter values majorly just like combination.
- **make_pipeline** basically for transforming a final estimator.
- **StandardScaler** is for standardizing features, we remove the mean and scale to unit variance.
- **MinMaxScaler** is for scaling features to a given range.
- **LabelEncoder** is a class for encoding categorical variables as integers (This is weird though).
- **mean_squared_error**: This is used calculate the mean squared error between the predicted values and the observed values.

CODE

```
# modeling
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler,MinMaxScaler
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
```

- **Seed = 60**, **random.seed(seed)** and **np.random.seed(seed)** This guys are just there to make sure when my tutor reproduce the codes and try to see my figures, the random numbers are still the same as the ones I used basically.

  CODE

```
import math, random
seed = 60
random.seed(seed)
np.random.seed(seed)
```

1.4Loading My Autotrader's CSV Dataset

These reads the data and shows me the five lines in other to get the overall understanding of the data and see if the data was loaded correctly or not.

CODE

```
data = pd.read_csv("adverts.csv")
data.head()
```

RESULT

Here, I can see the data has about 12 columns and multiple rows but I need to be certain how many rows and columns there is which leads me to the next step.

1.5 Inspected my Data with

CODE

```
data.shape
```

RESULT

I discovered my data has 402k rows and 12 features to work with, how interesting!

```
(402005, 12)
```

1.6 Summary of all the 12 features in detail

CODE

```
data.info()
```

RESULT

This showed the DataFrame/Series structure and values.

i. I noticed, as you can see that reg code, standard colour, make, model, vehicle condition and body type all took "object" structure will be somewhat strange to deal with but we will get to that. Mileage, price and public reference all took numbers structure either float or integer which is good. Lastly Crossover car and van took the Boolean way.

ii. There was quite a large chunk of missing data from year of registration, reg code and standard color, they are missing between 5k – 25k of data which would definitely affect my analysis but I'll deal with it later on in this work. Also, the other features have below 2,000 missing data but not as much as the once above.

iii. The thing about these features are that some are useless with regards to predicting price as needed. Features such as year of registration or reg code these freatures have no value whatsoever to how much a car is worth. Also, public reference, seem so unneeded for our analysis. Mileage, yes, it is needed and fuel type, these features seems reasonable enough to predict car price but we will not just conclude now. We will check later on the feature importance.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 402005 entries, 0 to 402004
Data columns (total 12 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   public_reference       402005 non-null  int64
 1   mileage                401878 non-null  float64
 2   reg_code               370148 non-null  object
 3   standard_colour        396627 non-null  object
 4   standard_make          402005 non-null  object
 5   standard_model         402005 non-null  object
 6   vehicle_condition      402005 non-null  object
 7   year_of_registration   368694 non-null  float64
 8   price                  402005 non-null  int64
 9   body_type              401168 non-null  object
 10  crossover_car_and_van  402005 non-null  bool
 11  fuel_type              401404 non-null  object
dtypes: bool(1), float64(2), int64(2), object(7)
memory usage: 34.1+ MB
```

1.7VISAUALIZATION

i. MILEAGE

I am going to visualize my data using histogram and boxplot to know how devastating my outliers are.

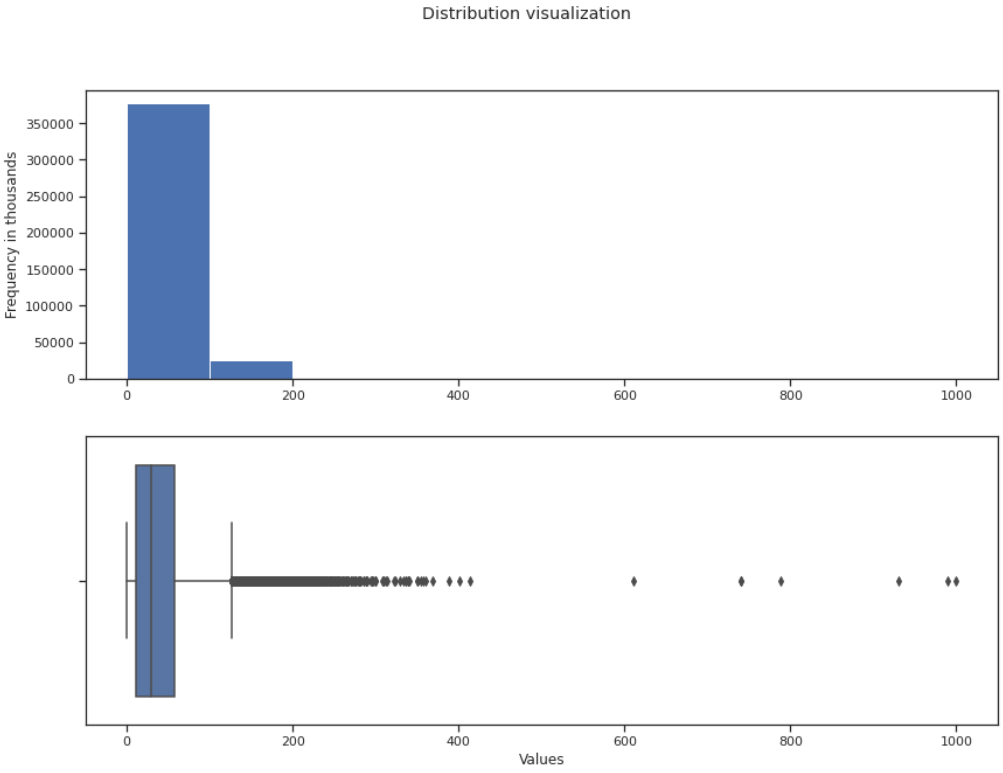
CODE

```
def show_distribution(col):

#This function will make a distribution (graph) and display it

# Creating our figure for 2 subplots (2rows, 1 column)
  fig, ax = plt.subplots(2, 1, figsize = (14,10))

# Ploting a histogram for our visualization
  ax[0].hist(col)
  ax[0].set_ylabel('Frequency in thousands')
# Ploting a histogram for our visualization
#ax[1].boxplot(col) matplotlib this was not working so i had to use
seaborn
  sns.boxplot(col, orient="h" ,ax=ax[1])
  ax[1].set_xlabel('Values')
# Adding a title to the Figure
  fig.suptitle('Distribution visualization')
```

```
# Show the figure
show_distribution(data["price"]/1000)
```

RESULT

The plots below shows the barchart in thousands the mileage feature and the boxplots with the numbers of outliers above the whisker.

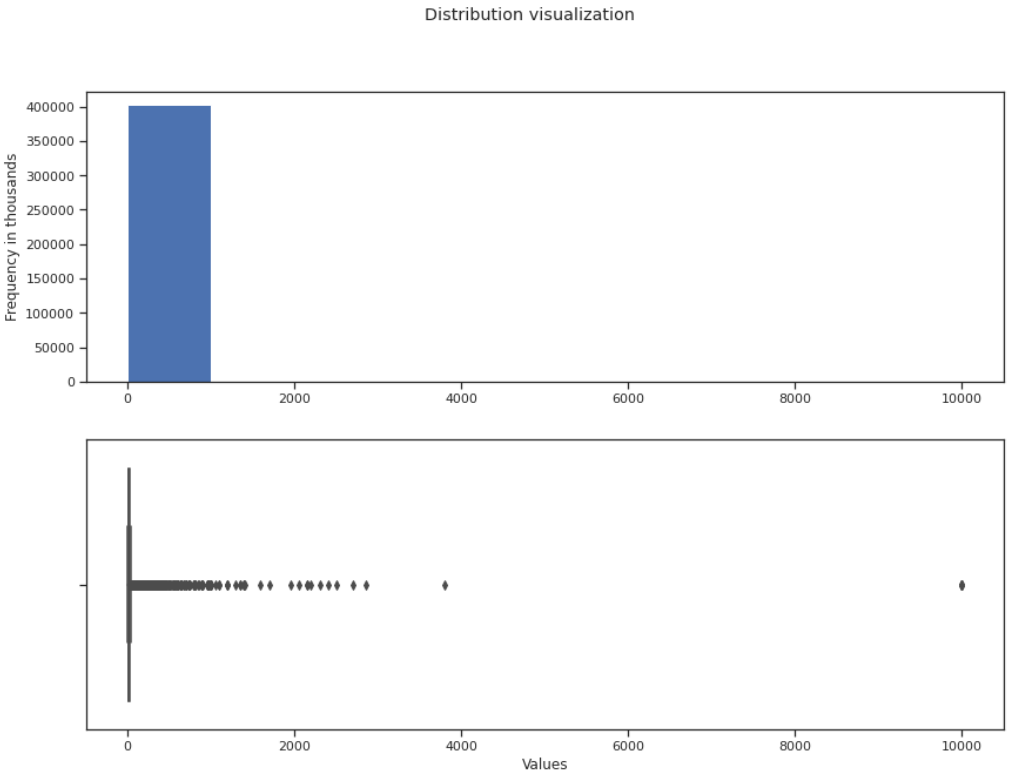


ii. PRICE

CODE

```
show_distribution(data["price"]/1000)
```

RESULT: Almost all the cars except the outliers are priced between lower than 2,000. The Ouliers are above the whisker showing extreme prices of cars recorded.

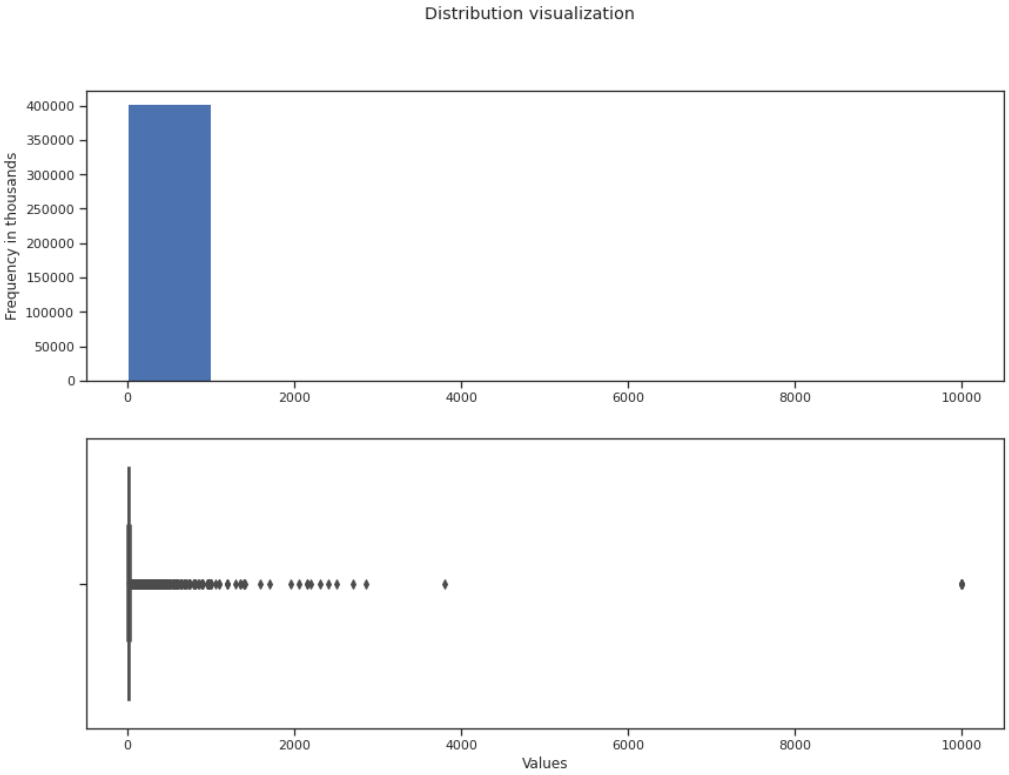


i. PUBLIC REFERENCE

CODE

```
show_distribution(data["public_reference"]/1000)
```

RESULT: Almost all the cars except the outliers were registered in between 2019 -2020 The Ouliers are above the whisker showing bunch of cars registered every year.



Chapter 2.0 Data Processing for Machine Learning

2.1 Dealing with Missing Values

CODE

Here I want to see what my missing values are in each feature.

```
data[data.isnull().any(axis=1)]
```

RESULT

Here the Table below displays the missing-true values as NaN

1 to 25 of 20000 entries Filter

index	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	crossover_car_and_van	fuel_type
0	202006039777689	0.0	NaN	Grey	Volvo	XC90	NEW	NaN	73970	SUV	false	Petrol Plug-in Hybrid
17	201911043995897	5.0	NaN	NaN	Nissan	X-Trail	NEW	NaN	27595	SUV	false	Diesel
19	202008272998331	0.0	NaN	White	Volkswagen	T-Cross	NEW	NaN	25000	SUV	false	Petrol
25	202008042070611	49585.0	B	NaN	Ferrari	308	USED	1984.0	54475	Convertible	false	Petrol
37	202001306737473	0.0	NaN	White	Fiat	Panda	NEW	NaN	13999	Hatchback	false	Petrol
44	202003178531910	0.0	NaN	NaN	Honda	Civic	NEW	NaN	19495	Hatchback	false	Petrol
45	202003318812338	0.0	NaN	Grey	Volvo	S60	NEW	NaN	40264	Saloon	false	Petrol
50	202006240456063	0.0	NaN	Grey	Fiat	500	NEW	NaN	11700	Hatchback	false	Petrol
54	202007030806426	30000.0	18	Red	Vauxhall	Insignia	USED	NaN	11990	Hatchback	false	Petrol
57	202007161315831	10.0	NaN	White	BMW	3 Series	NEW	NaN	38905	Estate	false	Diesel
75	202008142494844	0.0	NaN	White	Volkswagen	Polo	NEW	NaN	16461	Hatchback	false	Petrol
83	202008222801747	42847.0	61	Red	Honda	Jazz	USED	NaN	5695	Hatchback	false	Petrol
112	202009103570045	10.0	NaN	Blue	Volkswagen	Polo	NEW	NaN	16500	Hatchback	false	Petrol
117	202009143697500	4.0	NaN	Black	Volkswagen	Golf	NEW	NaN	26292	Hatchback	false	Petrol
189	202009304409714	10.0	NaN	White	Mitsubishi	Mirage	NEW	NaN	14615	Hatchback	false	Petrol
211	202010054632179	1.0	NaN	White	BMW	X3	NEW	NaN	41980	SUV	false	Diesel Hybrid
222	202010074703739	10.0	NaN	White	Audi	A3	NEW	NaN	28255	Saloon	false	Petrol
242	202010104850439	10.0	NaN	Grey	Audi	A5 Cabriolet	NEW	NaN	39845	Convertible	false	Petrol
246	202010124898698	5.0	NaN	White	Suzuki	Ignis	NEW	NaN	15964	Hatchback	false	Petrol Hybrid
255	202010144973622	0.0	NaN	Silver	Audi	Q3	NEW	NaN	41101	SUV	false	Diesel
276	202010175109922	50.0	NaN	Blue	Toyota	Yaris	NEW	NaN	22275	Hatchback	false	Petrol Hybrid
297	202010225305457	10.0	NaN	Silver	Hyundai	i30	NEW	NaN	23990	Estate	false	Petrol
299	202010225309553	10.0	NaN	Multicolour	Volkswagen	ID.3	NEW	NaN	35880	Hatchback	false	Electric
307	202010245377951	33287.0	63	Red	Volkswagen	Caddy	USED	2013.0	10990	NaN	false	Diesel
318	202010275490949	30586.0	65	NaN	Land Rover	Range Rover Sport	USED	2016.0	37950	SUV	false	Diesel

2.2 Summarizing my missing values

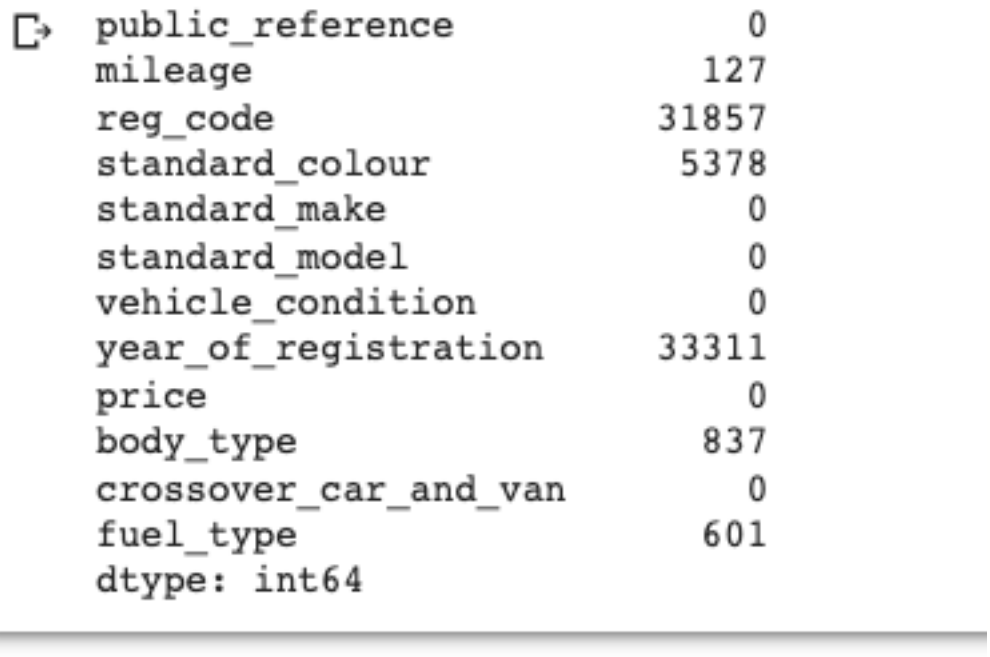
This code summarizes all the missing values and brings out the figures I need to deal with

CODE

```
data.isna().sum()
```

RESULT

The result here shows all my missing values in figures, giving me idea of how to process the Data.



```
public_reference            0
mileage                   127
reg_code                31857
standard_colour          5378
standard_make               0
standard_model              0
vehicle_condition           0
year_of_registration    33311
price                       0
body_type                 837
crossover_car_and_van       0
fuel_type                 601
dtype: int64
```

2.3 Replacing my missing value

Importing this class for replacing all the missing values so my analysis can be well represented. I think I might use mean.

CODE

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer()
data["fuel_type"]=data["fuel_type"].replace(np.nan,data[
"fuel_type"].mode()[0])

data["mileage"]= data["mileage"].replace(np.nan,data["mileage"].mean())
```

RESULT *(BEFORE AND AFTER MEAN REPRESENTATION IN BODY_TYPE)*

We can see that the missing values (NaN) are no longer present and also the number of petrol increased.

```
Petrol                   216929
Diesel                   158120
Petrol Hybrid             13602
Petrol Plug-in Hybrid      6160
Electric                   4783
Diesel Hybrid              1403
NaN                         601
Bi Fuel                     221
Diesel Plug-in Hybrid       185
Natural Gas                   1
Name: fuel_type, dtype: int64
```

```
Petrol                   217530
Diesel                   158120
Petrol Hybrid             13602
Petrol Plug-in Hybrid      6160
Electric                   4783
Diesel Hybrid              1403
Bi Fuel                     221
Diesel Plug-in Hybrid       185
Natural Gas                   1
Name: fuel_type, dtype: int64
```

2.4 OUTLIERS AND NOISE

i. These quantiles are used to better understand the data and where the outliers lie.

CODE

```
quant1 = data["mileage"].quantile(0.25)
quant1
quant2 = data["mileage"].quantile(0.50)
quant2
quant3 = data["mileage"].quantile(0.75)
quant3

#percentile, quantile
quant1 = data["mileage"].quantile(0.25)
quant3 = data["mileage"].quantile(0.75)
IQR = quant3 - quant1
up_lim =    quant3 + 1.5 * IQR

#percentile, quantile quartile
data["mileage"].min(), data["mileage"].max()
#I am dropping all values above this limit
up_lim
```

RESULT

These gives the details about the three quantile range values; my min and max values in mileage (most important feature); and the upper limit which may not be needed in the analysis.

```
(10487.0), (28648.0), (56852.0), (0.0, 999999.0), (126399.5)
```

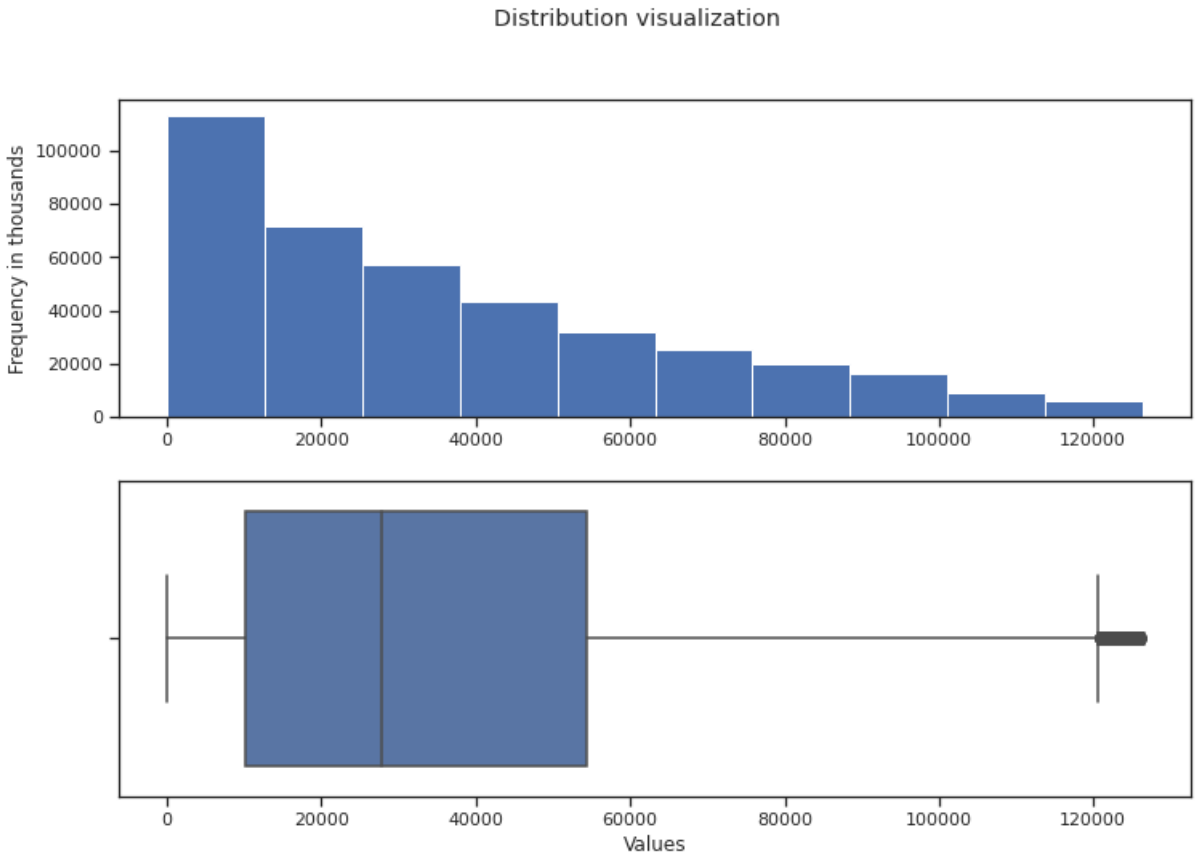
ii. I am going to filtre the noise and show the distribution removing all upper limit values.

CODE

```
col = data["mileage"] < up_lim
show_distribution(data["mileage"][col])
```

RESULT

All the data are well represented



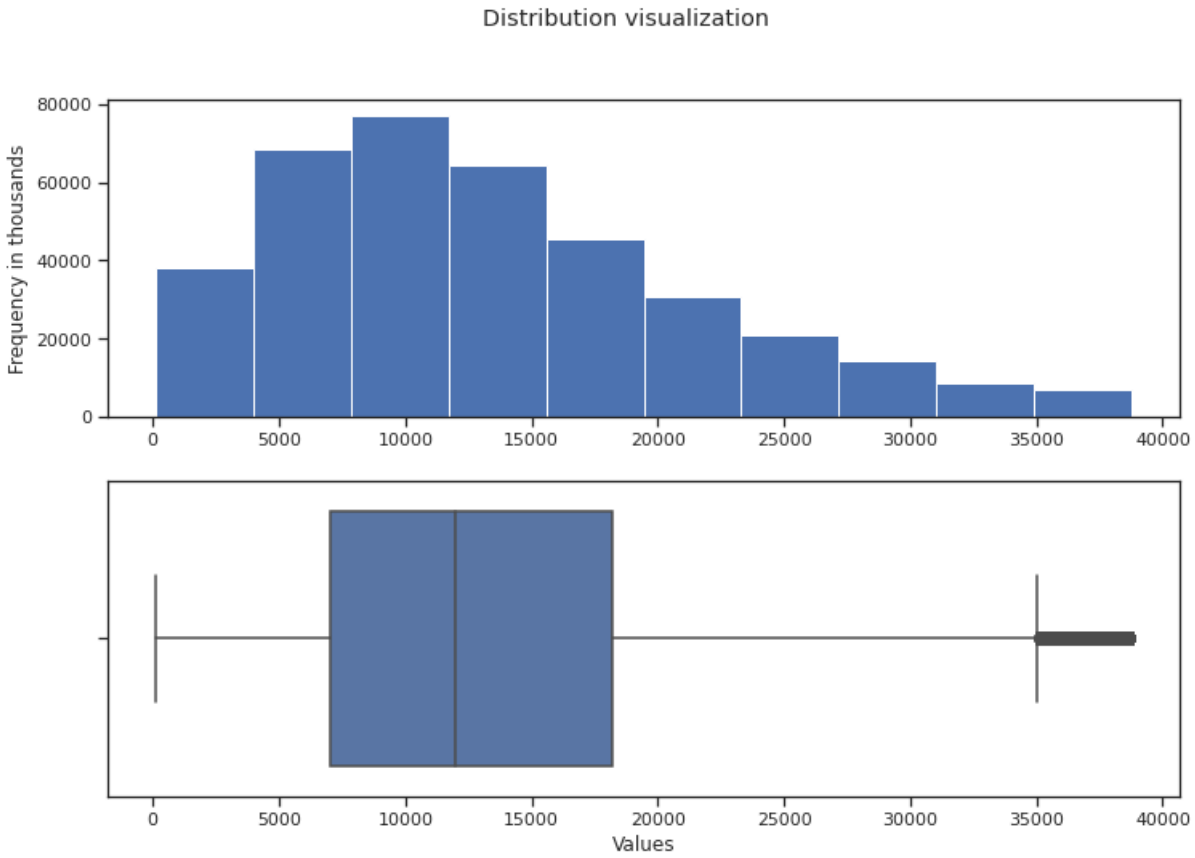
iii. This gives the breakdown for price also

CODE

```
#percentile, quantile
quant1 = data["price"].quantile(0.25)
quant2 = data["price"].quantile(0.50)
quant3 = data["price"].quantile(0.75)
IQR = quant3 - quant1
up_lim =    quant3 + 1.5 * IQR

#percentile, quantile
col = (data[data["price"] < up_lim]["price"]).reset_index(drop=True)
show_distribution(col)
```

RESULT

This shows the distribution is well represented with no noise and how my data prices are spread. More cars are sold within the range of £10,000 and less within £35k and above


Distribution visualization
Frequency in thousands
80000
60000
40000
20000
0
0
5000
10000
15000
20000
25000
30000
35000
40000
Values

Chapter 3.0 Engineer Features for machine learning purposes

3.1 Feature Engineering

The part of my model building is delicate, trying several methods seems futile so settling for vehicle condition firstly.

CODE

To know what are the categories in my column and see how to engineer it.

```
data["vehicle_condition"].value_counts()
```

RESULT

Shows used and new car conditions.

```
USED 370756 NEW 31249 Name: vehicle_condition, dtype: int64
```

3.2 Mapping for Data Transformation

i. I will make my Used "0" and New "1" for Vehicle Condition and also convert for all other categorical data as well to numeric for easier modelling.

CODE

```
#Data Mapping for Data transformation
data["vehicle_condition"] = data["vehicle_condition"].map({"USED":0,
"NEW":1})
#Transfroming more categorical data to numerical data
data = pd.get_dummies(data, columns=["vehicle_condition","fuel_type",
"body_type", "crossover_car_and_van"])
```

RESULT

`data.head ()` to show the first five lines of the tansformed data.

ii. I am going to drop the objects types not needed in my model for now. Focusing on the remaining variable X for "Price Prediction as Y"

CODE

```
# Dropping all object because we can use them for modelling
X = data.drop(labels= ["public_reference", "reg_code",
"year_of_registration", "standard_colour","price", "standard_model",
"standard_make"], axis=1)
y = data["price"]
```

RESULT

This further inspects my transformation process if it came out well

```
X.head
X.info()
```

3.3 SPLITING MY TRAIN AND TEST DATA

I set my reproducibility state to 60 and also set training and test set to 80 and 20 respectively

CODE

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=60)
X_train.shape, y_train.shape
```

RESULT

Inspecting my test and train sets I have the data arranged below as

```
((321604, 30), (321604,))
```

3.4 GETTING ROOT MEAN SQUARED ERROR TO CALCULATE PERFOMANCE

To determine the RMSE of the baseline model, using the mean car price and the determined RMSE, and using the mean of the training labels as the baseline forecast.

```
# The baseline is the mean of the y_train
y_mean = y_train.mean()
print(f"mean of my train test  (y_mean)")

y_pred_baseline = [y_mean] * len(y_train)

rmse_baseline = math.sqrt(mean_squared_error(y_train, y_pred_baseline))
print("Mean of the car price", round(y_mean, 2))
print("Baseline for RMSE:", round(rmse_baseline, 2))
```

RESULT

Generally, the lower my RMSE the better it is for now.

```
mean of my train test  (y_mean)
Mean of the car price 17331.47
Baseline for RMSE: 43498.24
```

CHAPTER 4.0 USING PIPELINE AND TRANSFORMERS

4.1 Choosing Random Forest Regressor: with 50 n_estimators, max_depth of 3, at least 3 leaf node and standard scaler to standardize. Then, I went ahead to calculate the RMSE again to see if it will reduce abit.

CODE

```
model = make_pipeline(
    StandardScaler(),
    RandomForestRegressor(n_estimators=100,max_depth=6,
min_samples_leaf=6)
)

model.fit(X_train, y_train)

predictions = model.predict(X_test)

rmse = math.sqrt(mean_squared_error(y_test, predictions))
print(f"Root Mean Squared Error: {rmse}")
```

RESULT

The scores are not desirable for me. So, I will go on to Chapter 5 grid search to five the best combination.

`Root Mean Squared Error: 55160.29678233147:` It did not.

`0.16938471881897166 for my Rsquared.`

CHAPTER 5.0 GRID SEARCH

5.1 This finds the best possible combination for our model using param_grid.
This step takes a long time.

CODE

```
param_grid = {
    "n_estimators": [50,100,150,200],
    "max_depth": [2,3,4,5],
    "min_samples_leaf":[2,3,4,5,6]
}
grid = GridSearchCV(RandomForestRegressor(), param_grid=param_grid,
n_jobs=1, verbose=1)
grid.fit(X_train, y_train)
grid.best_params_
```

RESULT

This gives the best possible combination from my grid search.

```
{'max_depth': 4, 'min_samples_leaf': 5, 'n_estimators': 150}
```

This summarizes the missing values
Outliers, and Noise (1-2) 10% 2.2. Feature Engineering, Data Transformations, Feature Selection (2-3)

5.2 Feature Importance

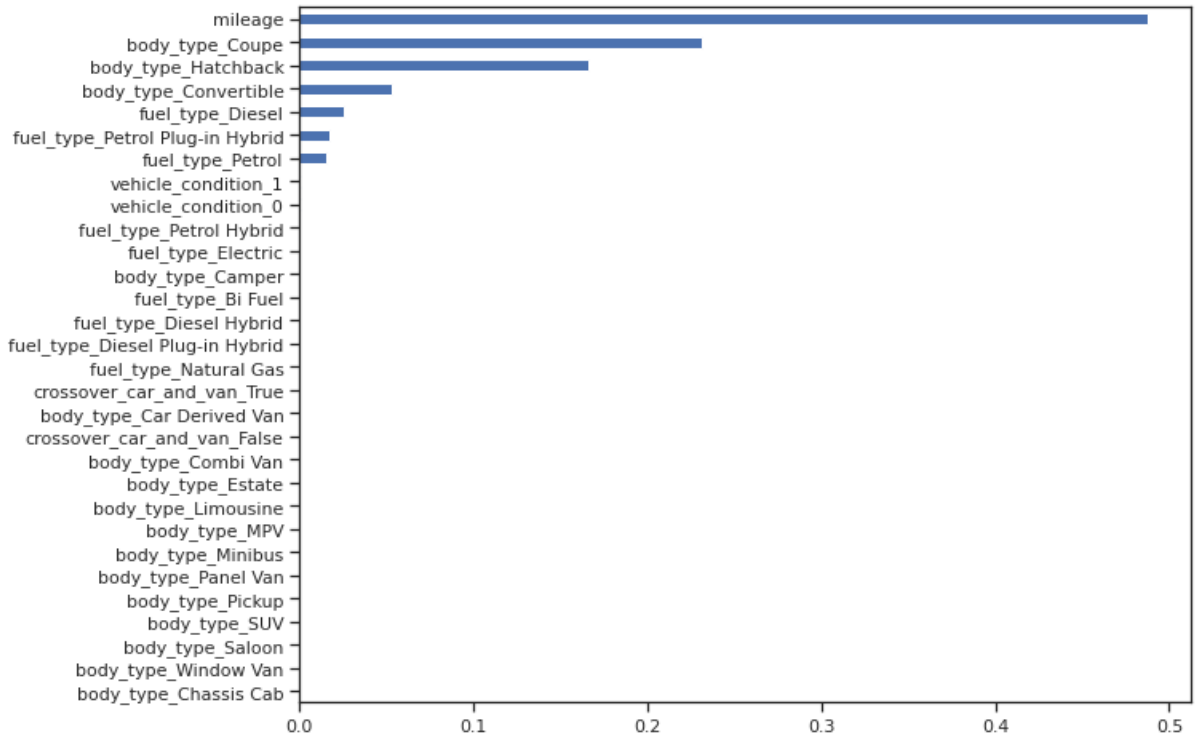
Seeing how terrible my model score performed, I will reiterate the process but first check the feature importance to know the best feature bet for my model.

CODE

```
importance = model.feature_importances_
features = pd.DataFrame({"importance":importance}, index=X.columns)
features.head()
plt.figure(figsize=(10,8))
features["importance"].sort_values().plot(kind="barh");
```

RESULT

All The Features Without Zero Blue Line Have Nothing To Do With Predicting Price but mean I can just drop them because that may have adverse effect on my model


mileage
body_type_Coupe
body_type_Hatchback
body_type_Convertible
fuel_type_Diesel
fuel_type_Petrol Plug-in Hybrid
fuel_type_Petrol
vehicle_condition_1
vehicle_condition_0
fuel_type_Petrol Hybrid
fuel_type_Electric
body_type_Camper
fuel_type_Bi Fuel
fuel_type_Diesel Hybrid
fuel_type_Diesel Plug-in Hybrid
fuel_type_Natural Gas
crossover_car_and_van_True
body_type_Car Derived Van
crossover_car_and_van_False
body_type_Combi Van
body_type_Estate
body_type_Limousine
body_type_MPV
body_type_Minibus
body_type_Panel Van
body_type_Pickup
body_type_SUV
body_type_Saloon
body_type_Window Van
body_type_Chassis Cab
0.0
0.1
0.2
0.3
0.4
0.5