1. Data/Domain Understanding and Exploration

```
In [370]: import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          import seaborn as sns
          import plotly.express as px
```

```
In [371]: cars = pd.read_csv('adverts.csv')
```

```
In [372]: cars.info()
```

```
<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
```

Imported all the required libraries to analyse the data available in the adverts dataset.

The cars.info() provides us the information of the dataset, the column names and the non-null count of each column and their data types.

We have all the columns and their non null count and also their data types provided by the .info() function.

```
cars1 = cars.sample(n=50000, replace=True, ignore_index=True)
```

```
cars1.head()
```

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	crossover_(
0	202007301888026	34000.0	17	Blue	Vauxhall	Zafira Tourer	USED	2017.0	9999	MPV	
1	202010014441685	83000.0	13	Blue	Toyota	Auris	USED	2013.0	5495	Estate	
2	202010024514416	90229.0	08	Grey	SKODA	Fabia	USED	2008.0	2490	Hatchback	
3	202010275485796	91177.0	60	Silver	Nissan	X-Trail	USED	2011.0	7250	SUV	
4	202009244141806	75000.0	60	Blue	SEAT	Alhambra	USED	2011.0	8000	MPV	

```
cars1.shape
```

```
(50000, 12)
```

```
cars1.columns
```

```
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'],
      dtype='object')
```

The cars1 is the variable having 50000 samples of the original dataset.

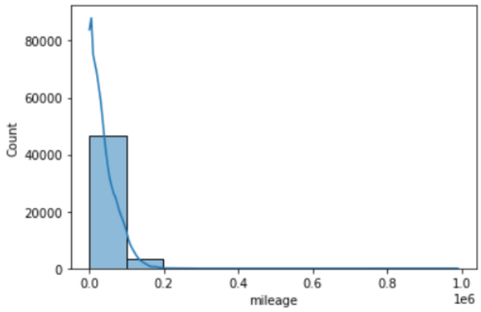
We use .head() to see the top 5 entries of the dataset and .shape() gives us the number of rows and columns.

.columns gives us the name of the columns present in the dataset.

```python
cars['mileage'].describe()
```

```
count    401878.000000
mean      37743.595656
std       34831.724018
min           0.000000
25%       10481.000000
50%       28629.500000
75%       56875.750000
max      999999.000000
Name: mileage, dtype: float64
```

```python
sns.histplot(cars1['mileage'], bins=10, kde=True);
```



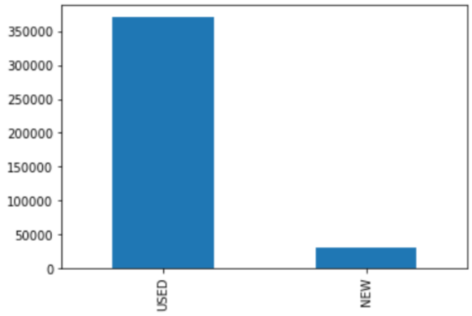
By using the .describe function against any column name we get the statistical values of that particular column. If the data has quantative values it provides the mean, standard deviation, minimum value etc and for alphabetical or mixed values it gives the count, top , frequency etc.

By using the seaborn library we can plot a histogram for a particular column to visualize the data. For example we have the histogram plot for mileage.

```python
cars['vehicle_condition'].value_counts()
```

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

```python
cars['vehicle_condition'].value_counts().plot.bar();
```




```python
cars['vehicle_condition'].unique()
```

```
array(['NEW', 'USED'], dtype=object)
```

.value_counts() function helps to find the number of unique entries present in a column. We can also use this data to plot a bar graph by using the .plot.bar() function.

.unique() function helps to find all the unique entries present in a particular column.

```
price_99percentile = cars['price'].quantile(0.99)
price_99percentile
```

```
88899.52000000025
```

```
cars.loc[ cars['price']>=price_99percentile ].sample(5)
```

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	cros
250835	202008142478239	6667.0	16	Yellow	Lamborghini	Huracan	USED	2016.0	159950	Convertible	
266033	202010295571296	1000.0	20	Black	McLaren	720S	USED	2020.0	179995	Coupe	
307128	202001256578661	17300.0	61	Black	Lamborghini	Aventador	USED	2011.0	148995	Coupe	
326773	202010225316874	7968.0	19	Black	Mercedes-Benz	G Class	USED	2019.0	144950	SUV	
98548	202009254208101	18400.0	63	Multicolour	Rolls-Royce	Ghost	USED	2013.0	110000	Saloon	

```
len(cars.loc[ cars['price']>=price_99percentile ])
```

```
4021
```

```
cars.loc[ cars['price']>=price_99percentile ].sort_values('price').head(5)
```

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	cros
397529	202010094806610	3600.0	20	Black	Porsche	Panamera	USED	2020.0	88900	Hatchback	
44391	202009254193762	3615.0	18	Blue	Porsche	911	USED	2018.0	88900	Convertible	
128415	202010024503657	22211.0	67	Black	Porsche	Panamera	USED	2017.0	88900	Hatchback	
365839	202009093531462	8500.0	16	Orange	McLaren	570S	USED	2016.0	88900	Coupe	
355421	202009113603574	NaN	70	Green	Lotus	Exige	USED	2020.0	88920	Coupe	

.quantile() function can be used to find the values that lies above a certain quantile or certain range.

1.1. Identification/Commenting on Missing Values (2-3)

```
cars1.isna().sum()
```

```
public_reference            0
mileage                     9
reg_code                 4058
standard_colour           647
standard_make               0
standard_model              0
vehicle_condition           0
year_of_registration     4236
price                       0
body_type                  83
crossover_car_and_van       0
fuel_type                  66
dtype: int64
```

```
cars1[cars1.isnull().any(axis=1)]
```

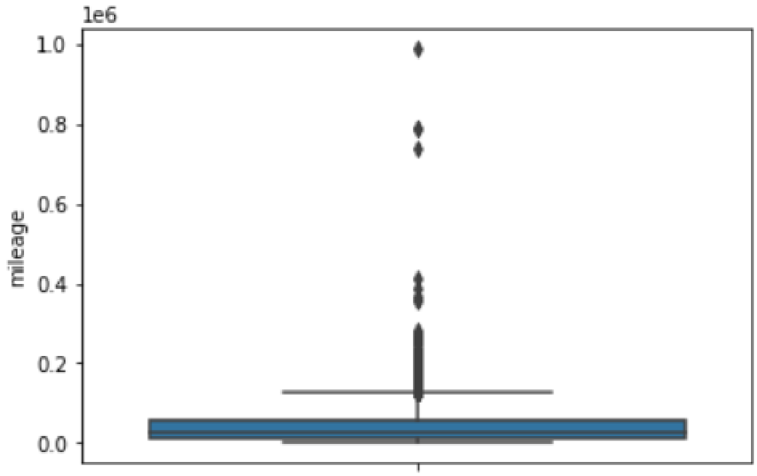
	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	cross
7	202010145008101	0.0	NaN	NaN	Fiat	500	NEW	NaN	15499	Hatchback	
25	202006190288881	5.0	NaN	White	Vauxhall	Corsa	NEW	NaN	20155	Hatchback	
30	202009193926402	0.0	NaN	White	Mercedes-Benz	GLB Class	NEW	NaN	42033	SUV	
31	202009013191274	60000.0	V	NaN	Lotus	Elan	USED	2000.0	6995	Convertible	
39	202010175118254	3155.0	69	Grey	SEAT	Ateca	USED	NaN	33000	SUV	
...	...	...	...	...	...	...	...	...	...	...	
49975	202010124889687	0.0	NaN	Grey	Volvo	V60	NEW	NaN	42460	Estate	
49981	202010285524214	37000.0	Y	NaN	Fiat	Punto	USED	2001.0	1495	Hatchback	
49983	202010235333444	0.0	NaN	Black	Fiat	500C	NEW	NaN	16850	Convertible	
49997	202010305616619	12497.0	67	Grey	Volkswagen	Golf	USED	NaN	20799	Hatchback	

.isna() function stands for 'is not assigned' and is used to find the missing values in the dataset, by adding the .sum() function we get the number of null values for each column.

By exploring the data we found that when the vehicle_condition == 'NEW' then the reg_code and year_of_registration has the value nan i.e. not assigned.

1.3. Identification/Commenting on Outliers and Noise

```
sns.boxplot(y=cars1['mileage']);
```



From the box plot graph of mileage we can see that it has many outliers above the its range and can be removed using the quantile functions.

```
cars['year_of_registration'].unique()
```

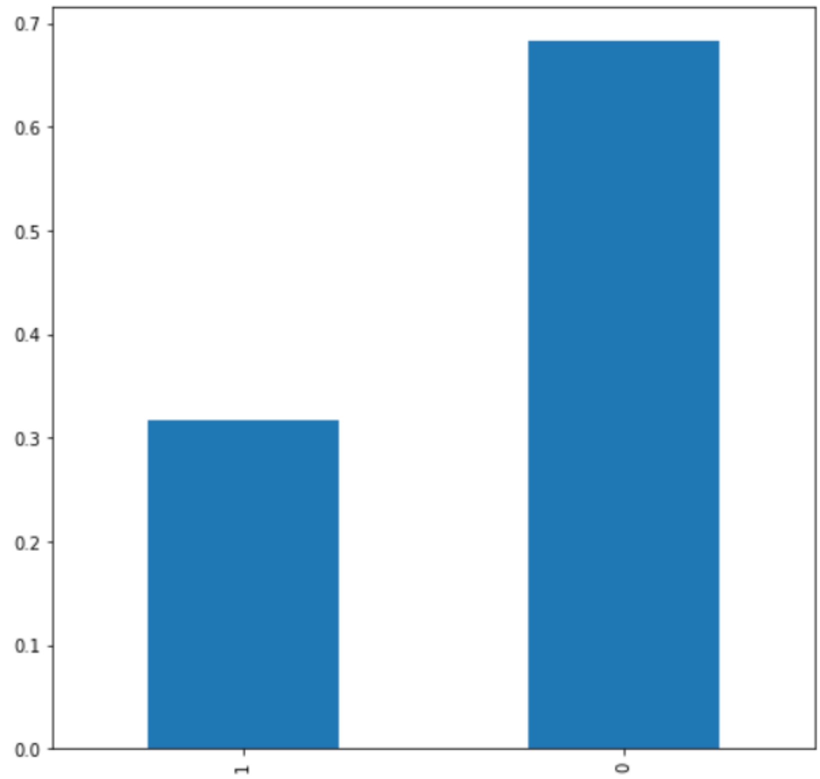
```
array([  nan, 2011., 2017., 2016., 2015., 2013., 2008., 2019., 2010.,
       2012., 2018., 2009., 1984., 2014., 2003., 2006., 2020., 2005.,
       2000., 2002., 2007., 2004., 1991., 2001., 1986., 1998., 1990.,
       1993., 1987., 1994., 1999., 1970., 1988., 1995., 1997., 1969.,
       1992., 1989., 1996., 1976., 1983., 1980., 1973., 1962., 1967.,
       1972., 1982., 1968., 1979., 1964., 1933., 1981., 1985., 1978.,
       1971., 1974., 1966., 1977., 1961., 1965., 1007., 1957., 1515.,
       1963., 1063., 1954., 1975., 1955., 1009., 1016., 1960., 1956.,
       1959., 1909., 1934., 1958., 1010., 1950., 1008., 1018.,  999.,
       1017., 1952., 1006., 1015.])
```

We can see in all the distinct values for the column 'year_of_registration' we have outliners such as ['999','1015,'1006','1007','1515','1063','1009','1016','1010','1008','1018','1017']

This values are either incorrect entries in the data set or some false data.

Thus, to further analyse the dataset we should consider only the values greater than 1900 for the best prediction of the Price.

```
plt.subplots(figsize=(6,4))
cars.loc[
    cars['price']>=cars['price'].quantile(0.95), 'vehicle_condition'
].value_counts(sort=False, normalize=True).plot.bar(figsize=(8,8));
```



We are using the quantile function to find the number of vehicles according to the vehicle condition that are above the 95% quantile area and are can be considered as outliers.

2. Data Processing for machine learning

2.1. Dealing with Missing Values, Outliers, and Noise

```
cars['year_of_registration'] = cars['year_of_registration'].mask((cars['vehicle_condition']=='NEW'), 2022)
cars
```

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	cros
0	202006039777689	0.0	NaN	Grey	Volvo	XC90	NEW	2022.0	73970	SUV	
1	202007020778260	108230.0	61	Blue	Jaguar	XF	USED	2011.0	7000	Saloon	
2	202007020778474	7800.0	17	Grey	SKODA	Yeti	USED	2017.0	14000	SUV	
3	202007080986776	45000.0	16	Brown	Vauxhall	Mokka	USED	2016.0	7995	Hatchback	
4	202007161321269	64000.0	64	Grey	Land Rover	Range Rover Sport	USED	2015.0	26995	SUV	
...	...	...	...	...	...	...	...	...	...	...	
402000	202010315652942	5179.0	69	Grey	Peugeot	208	USED	2019.0	10595	Hatchback	
402001	202010315657341	110000.0	59	Red	Peugeot	107	USED	2009.0	2000	Hatchback	
402002	202010315659271	52760.0	62	White	Nissan	Qashqai	USED	2012.0	7250	SUV	
402003	202011015662436	10250.0	65	Red	Abarth	595	USED	2015.0	11490	Hatchback	
402004	201512149444029	14000.0	14	Silver	Audi	A4 Avant	USED	2014.0	20520	Estate	

402005 rows × 12 columns

In this line we have masked the value of 2022 in the column year_of_registration whenever the vehicle_condition is 'NEW'.

```
cars['reg_code'] = cars['reg_code'].mask((cars['vehicle_condition']=='NEW'), 0)
```

```
cars['year_of_registration'] = cars['year_of_registration'].fillna(0)
```

```
cars['year_of_registration'] = cars['year_of_registration'].astype(int)
```

```
cars['mileage'].fillna(cars['mileage'].mean(), inplace=True)
```

```
cars['mileage'] = cars['mileage'].mask((cars['vehicle_condition']==1), 0)
```

```
cars['standard_colour'].fillna(cars['standard_colour'].mode()[0], inplace=True)
```

```
cars.isna().sum()
```

```
public_reference          0
mileage                   0
reg_code                608
standard_colour           0
standard_make             0
standard_model            0
vehicle_condition         0
year_of_registration      0
price                     0
body_type               837
crossover_car_and_van     0
fuel_type               601
dtype: int64
```

By using .mask() function on reg_code we can provide a condition that the vehicle condition should be 'NEW' and the value at that particular row for reg_code should be 0.

.fillna() function is used to fill the remaining nan values present in the year_of_registration column.

.astype(int) function converted the data type of year_of_registration from float to int as there are no decimal values in a year.

Again the mask function was used to set the mileage as 0 for the vehicle whose condition is new.

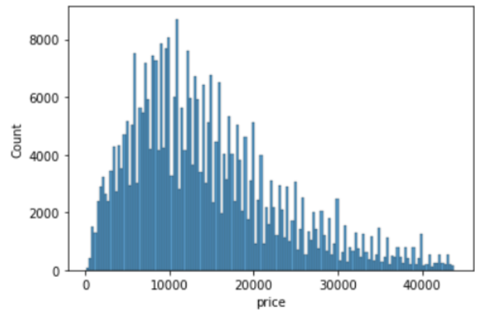
.fillna() function was used to replace the missing values of standard_colour with its mode i.e. the most frequently used colour.

Now we can se we have replaced many missing values with some significant data to improve the data clearity.

```
cars_outliers =cars.loc[(cars['price']<= cars['price'].quantile(0.95))]
cars_outlier =cars_outliers.loc[(cars_outliers['mileage']<= cars_outliers['mileage'].quantile(0.95))]
```

```
sns.histplot(data= cars_outlier, x='price')
```

```
<AxesSubplot:xlabel='price', ylabel='Count'>
```




```
cars_outlier
```

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	cros:
2	202007020778474	7800.0	17	Grey	SKODA	Yeti	USED	2017	14000	SUV	
3	202007080986776	45000.0	16	Brown	Vauxhall	Mokka	USED	2016	7995	Hatchback	
4	202007161321269	64000.0	64	Grey	Land Rover	Range Rover Sport	USED	2015	26995	SUV	
5	202009304412074	16000.0	17	Blue	Audi	S5	USED	2017	29000	Convertible	
6	202007080998445	24075.0	17	Red	Vauxhall	Viva	USED	2017	5861	Hatchback	

We made a new variable called cars_outlier which does not have the values of prices and mileage from the range above of 95% quantile. Thus, we can assume that the outliers for the price and mileage were removed and this data could be used forward.

2.2. Feature Engineering, Data Transformations

```
cars_outlier['vehicle_condition'].replace({'USED':0,'NEW': 1}, inplace =True)
```

```
C:\Users\Abhishek\anaconda3\lib\site-packages\pandas\core\generic.py:6619: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
```

```
cars_outlier['crossover_car_and_van'] = cars_outlier['crossover_car_and_van'].astype(int)
```

```
C:\Users\Abhishek\AppData\Local\Temp/ipykernel_58188/3277959275.py:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
```

In the first line we have used the .replace() function to change the ‘NEW’ to 1 and ‘USED’ to 0 from the vehicle_condition column.

In the second line we have used astype(int) and changed the data type of crossover_car_and_van from Boolean to int and thus the values will be shown as 1’s and 0’s.

```python
from sklearn.preprocessing import MinMaxScaler
m = MinMaxScaler()
data_transform= cars_outlier.copy()
```

```python
data_transform.mileage = m.fit_transform(data_transform[['mileage']])
data_transform.price = m.fit_transform(data_transform[['price']])
```

```python
data_transform
```

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	c
2	202007020778474	0.072907	17	Grey	SKODA	Yeti	0	2017	0.317475	SUV	
3	202007080986776	0.420620	16	Brown	Vauxhall	Mokka	0	2016	0.180124	Hatchback	
4	202007161321269	0.598215	64	Grey	Land Rover	Range Rover Sport	0	2015	0.614707	SUV	
5	202009304412074	0.149554	17	Blue	Audi	S5	0	2017	0.660567	Convertible	
6	202007080998445	0.225032	17	Red	Vauxhall	Viva	0	2017	0.131313	Hatchback	
...	...	...	...	...	...	...	...	...	...	...	
401999	202010315651841	0.698528	59	Blue	Toyota	Auris	0	2009	0.056725	Hatchback	
402000	202010315652942	0.048409	69	Grey	Peugeot	208	0	2019	0.239593	Hatchback	
402002	202010315659271	0.493153	62	White	Nissan	Qashqai	0	2012	0.163083	SUV	
402003	202011015662436	0.095808	65	Red	Abarth	595	0	2015	0.260064	Hatchback	
402004	201512149444029	0.130859	14	Silver	Audi	A4 Avant	0	2014	0.466606	Estate	

362808 rows × 12 columns

From the sklearn.preprocessing library we are importing MinMaxscaler

We are using this minmaxscaler to transfer the mileage and price columns to handle and access all the numbers present in this two columns with ease.

```python
# Import label encoder
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()

feat_cars.standard_colour= label_encoder.fit_transform(feat_cars['standard_colour'])
feat_cars.standard_make = label_encoder.fit_transform(feat_cars['standard_make'])
```

```python
feat_cars
```

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	cros
2	202007020778474	7800.0	17	8	79	Yeti	0	2017	14000	SUV	
3	202007080986776	45000.0	16	4	91	Mokka	0	2016	7995	Hatchback	
4	202007161321269	64000.0	64	8	46	Range Rover Sport	0	2015	26995	SUV	
5	202009304412074	16000.0	17	2	6	S5	0	2017	29000	Convertible	
6	202007080998445	24075.0	17	17	91	Viva	0	2017	5861	Hatchback	
...	...	...	...	...	...	...	...	...	...	...	
401999	202010315651841	74732.0	59	2	89	Auris	0	2009	2600	Hatchback	
402000	202010315652942	5179.0	69	8	67	208	0	2019	10595	Hatchback	
402002	202010315659271	52760.0	62	20	62	Qashqai	0	2012	7250	SUV	
402003	202011015662436	10250.0	65	17	0	595	0	2015	11490	Hatchback	
402004	201512149444029	14000.0	14	18	6	A4 Avant	0	2014	20520	Estate	

362808 rows × 12 columns

Again from the sklearn library we got the preprocessing.labelencoder function to changes the features such standard_colour and standard_make to numerical values. Hence, we convert this two values to numeric values for our target column 'price'.

2.3. Subsetting (e.g., Feature Selection, Data Sampling)

```python
cars.standard_colour = label_encoder.fit_transform(cars['standard_colour'])
cars.standard_make = label_encoder.fit_transform(cars['standard_make'])
cars.standard_model = label_encoder.fit_transform(cars['standard_model'])
cars.body_type = label_encoder.fit_transform(cars['body_type'])
cars.fuel_type = label_encoder.fit_transform(cars['fuel_type'])
```

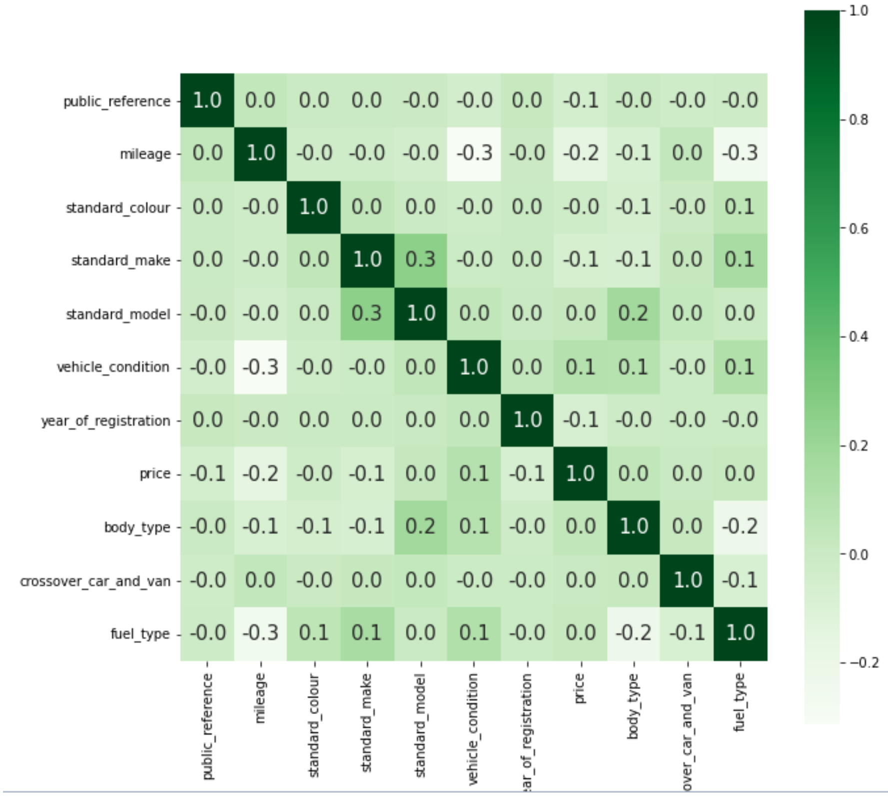
By using label encoder now we have converted all other featured columns into numeric values.

```
corr = cars.corr()
corr.shape
```

```
(11, 11)
```

```
plt.figure(figsize=(10,10))
sns.heatmap(corr, cbar=True, square= True, fmt='.1f', annot=True, annot_kws={'size':15}, cmap='Greens')
```

```
<AxesSubplot:>
```



As we can see from the heatmap the column standard_colour, standard_model, body_type, crossover_car_and_van and fuel_type have no correlation with the target column i.e. Price. Hence dropping those columns

```
cars.drop(cars.columns[[2,3,5,9,10,11]],axis=1,inplace=True)
```

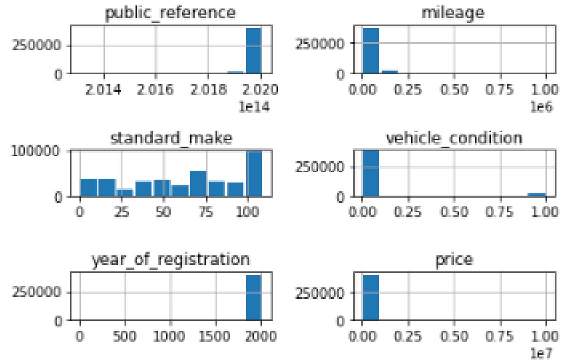
```
cars
```

	public_reference	mileage	standard_make	vehicle_condition	year_of_registration	price
0	202006039777689	0.0	106	1	2022	73970
1	202007020778260	108230.0	47	0	2011	7000
2	202007020778474	7800.0	91	0	2017	14000
3	202007080986776	45000.0	104	0	2016	7995
4	202007161321269	64000.0	54	0	2015	26995
...	...	...	...	...	...	...
402000	202010315652942	5179.0	78	0	2019	10595
402001	202010315657341	110000.0	78	0	2009	2000
402002	202010315659271	52760.0	72	0	2012	7250
402003	202011015662436	10250.0	2	0	2015	11490
402004	201512149444029	14000.0	8	0	2014	20520

402005 rows × 6 columns

Thus now we have the subset of the original dataset that is mostly required for the prediction of the target column ‘price’.

```
In [309]: cars.hist(rwidth=0.9)
          plt.tight_layout()
```

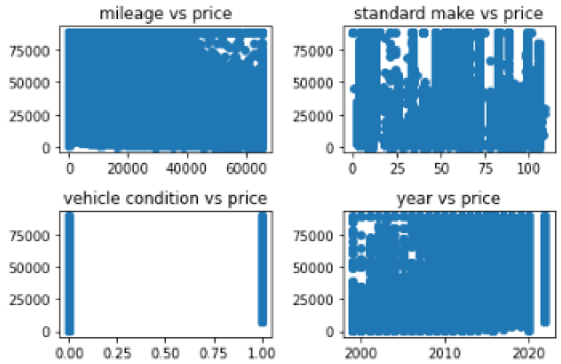



```
In [377]: plt.subplot(2,2,1)
          plt.title('mileage vs price')
          plt.scatter(cars['mileage'],cars['price'])

          plt.subplot(2,2,2)
          plt.title('standard make vs price')
          plt.scatter(cars['standard_make'],cars['price'])

          plt.subplot(2,2,3)
          plt.title('vehicle condition vs price')
          plt.scatter(cars['vehicle_condition'],cars['price'])

          plt.subplot(2,2,4)
          plt.title('year vs price')
          plt.scatter(cars['year_of_registration'],cars['price'])
          plt.tight_layout()
```

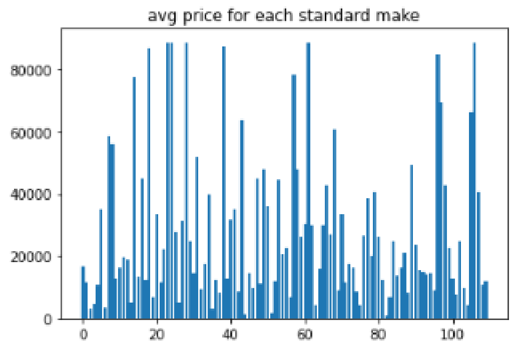


This particular code gives us the relation of all the different columns suitable for predicting price. In the first code we can se the histogram plot for public reference, mileage, standard make, vehicle condition and year of registration.

In the second plot we can see the scatter plot for mileage/standard make/vehicle condition/year vs price.

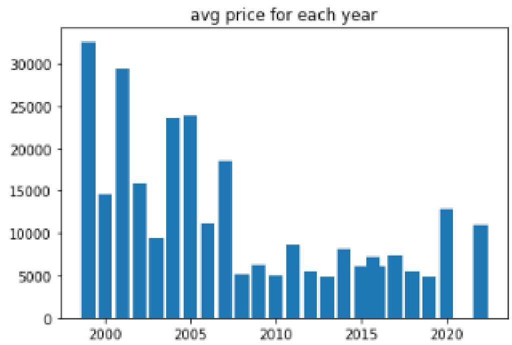
```
In [376]: plt.subplot(1,1,1)
          plt.title('avg price for each standard make')
          cat_list= cars['standard_make'].unique()
          cat_average= cars.groupby('standard_make').mean()['price']
          plt.bar(cat_list, cat_average)

Out[376]: <BarContainer object of 110 artists>
```



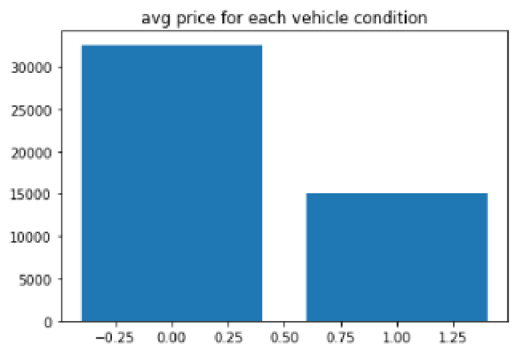

```
In [375]: plt.subplot(1,1,1)
          plt.title('avg price for each year')
          cat_list= cars['year_of_registration'].unique()
          cat_average= cars.groupby('year_of_registration').mean()['price']
          plt.bar(cat_list, cat_average)

Out[375]: <BarContainer object of 24 artists>
```




```
In [374]: plt.subplot(1,1,1)
          plt.title('avg price for each vehicle condition')
          cat_list= cars['vehicle_condition'].unique()
          cat_average= cars.groupby('vehicle_condition').mean()['price']
          plt.bar(cat_list, cat_average)

<BarContainer object of 2 artists>
```



In all this plot we are seeing the average price for standard make, year of registration and vehicle condition. For year of registration we can see that the price has been decreasing over the years and we could find a pattern to predict the price.

```
cars['year_of_registration'].describe()
```

```
count    402005.000000
mean       2005.214303
std         144.195423
min           0.000000
25%        2014.000000
50%        2017.000000
75%        2018.000000
max        2022.000000
Name: year_of_registration, dtype: float64
```

```
cars['year_of_registration'].quantile([0.01,0.05,0.1,0.15,0.90,0.95,0.99])
```

```
0.01    1999.0
0.05    2007.0
0.10    2009.0
0.15    2011.0
0.90    2020.0
0.95    2022.0
0.99    2022.0
Name: year_of_registration, dtype: float64
```

```
of_registration'] =cars['year_of_registration'].loc[(cars['year_of_registration']>= cars['year_of_registration'].quantile(0.01))]
```

```
cars[cars['year_of_registration']<1999]
```

public_reference	mileage	standard_make	vehicle_condition	year_of_registration	price

```
cars[cars['year_of_registration'].isna()]
```

	public_reference	mileage	standard_make	vehicle_condition	year_of_registration	price
25	202008042070611	49585.0	34	0	NaN	54475
54	202007030806426	30000.0	104	0	NaN	11990
83	202008222801747	42847.0	40	0	NaN	5695
426	202009033275983	175000.0	66	0	NaN	19990
865	202010084741550	43130.0	54	0	NaN	35990
...	...	...	...	...	...	...
401128	202009203972304	36000.0	88	0	NaN	7495
401314	202010315635541	12522.0	27	0	NaN	6300
401323	201909222504136	46000.0	105	0	NaN	22995
401357	202007111114611	10.0	104	0	NaN	16000
401951	202010235326414	285038.0	101	0	NaN	9990

3857 rows × 6 columns

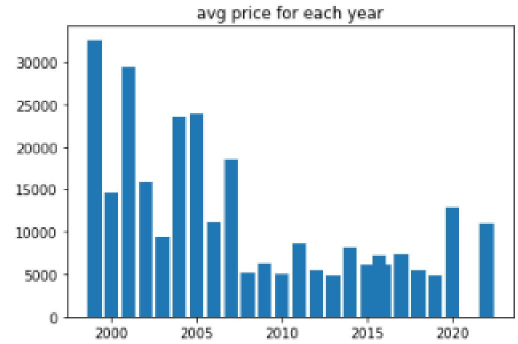
```
cars['year_of_registration'].fillna(cars['year_of_registration'].mean(), inplace=True)
```

```
cars[cars['year_of_registration'].isna()]
```

	public_reference	mileage	standard_make	vehicle_condition	year_of_registration	price

```
plt.subplot(1,1,1)
plt.title('avg price for each year')
cat_list= cars['year_of_registration'].unique()
cat_average= cars.groupby('year_of_registration').mean()['price']
plt.bar(cat_list, cat_average)
```

```
<BarContainer object of 24 artists>
```



In this particular code we have handled the outliers in the year of registration by just removing 0.01 quantile of the data that had the years such as 999, 1015, 1017 etc which were obviously wrong. After removing those outliers we have found some Nan values which were replaced by the mean of the column ‘year_of_registration’. Later we did not find any null values and the plot becomes more clearer.

In this similar way we have handled the outliers for mileage and price as well, for the column mileage we removed all the outliers above .90 quantile and replaced them with mean of mileage but for price we removed the values above .90 quantile but replaced those values with the max value.

3. Model Building

```python
max_depths = [ 2, 4, 6, 8, 10, 12, 16, 20 ]
```

```python
grid_param = {
    'max_depth': max_depths
}
```

```python
from sklearn.model_selection import GridSearchCV, ParameterGrid
```

```python
grid = GridSearchCV(
    DecisionTreeRegressor(),
    grid_param,
    scoring='neg_mean_absolute_error',
    return_train_score=True
)
```

```python
grid.fit(X_train, y_train)
```

```
GridSearchCV(estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [2, 4, 6, 8, 10, 12, 16, 20]},
             return_train_score=True, scoring='neg_mean_absolute_error')
```

```python
grid.best_params_
```

```
{'max_depth': 12}
```

```python
grid.best_score_
```

```
-4220.452518561155
```

```python
gs_results = pd.DataFrame(grid.cv_results_).sort_values('rank_test_score')
```

```python
gs_results[['param_max_depth', 'mean_train_score', 'mean_test_score', 'rank_test_score']]
```

	param_max_depth	mean_train_score	mean_test_score	rank_test_score
5	12	-3999.037626	-4220.452519	1
6	16	-3384.157181	-4248.436792	2
4	10	-4344.775445	-4435.672885	3
7	20	-2809.115420	-4523.554570	4
3	8	-5017.212080	-5050.323069	5
2	6	-5918.715208	-5930.248230	6
1	4	-6898.130030	-6903.915792	7
0	2	-7622.055939	-7622.831752	8

```python
winning_dtree = grid.best_estimator_
```

```python
mean_absolute_error(winning_dtree.predict(X_test), y_test)
```

```
4228.058936459858
```

```python
mean_absolute_error(winning_dtree.predict(X_train), y_train)
```

```
4014.833896808159
```

For decisiontreeregressor we have used grid search and ranking to determine which model is the most suitable or which model with a particular number of depths gives the best predicted value to determine the price of the vehicle. In the grid_param we have taken the values to be provided to the regressor to perform decision on a certain depth level. The values are 2,4,6,8,10,12,16,20 and out of this the grid provided us the best score with the depth of 12. In the gs result we can see the depth of the regressor, the training score, the testing score and based on the scores, their ranking.

4. Model Evaluation and Analysis

```
from sklearn.neighbors import KNeighborsRegressor
```

```
knn = KNeighborsRegressor(7)
knn.fit(X_train, y_train)
```

```
KNeighborsRegressor(n_neighbors=7)
```

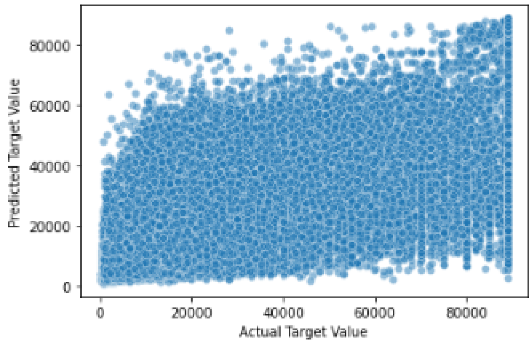
```
mean_absolute_error(knn.predict(X_test), y_test)
```

```
7916.564
```

```
mean_absolute_error(knn.predict(X_train), y_train)
```

```
6865.011
```

```
y_true = y
y_pred = knn.predict(X)
ax = sns.scatterplot(x=y_true, y=y_pred, alpha=0.5)
ax.set_xlabel('Actual Target Value')
ax.set_ylabel('Predicted Target Value')
ax.plot( alpha=0.3, lw=1);
```



We have used KNN regressor to predict the price of the car with the support of all the other columns. We have got the mean absolute error of 7916.564 for the testing data and 6865.011 for the training data. Thus the regressor for testing data gives around 1000 of mean absolute error if compared with the training data. If the number of nearest neighbours to be compared with is not given as 7 is would have given around more than 2000 of absolute mean error. You could find this in the notebook provided.

```python
from sklearn.linear_model import LinearRegression
```

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

```
LinearRegression()
```

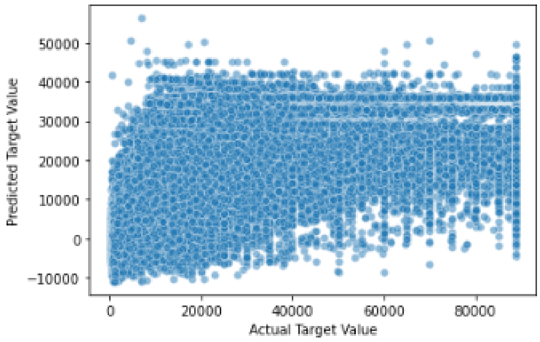
```python
mean_absolute_error(lr.predict(X_test), y_test)
```

```
7448.268605323344
```

```python
mean_absolute_error(lr.predict(X_train), y_train)
```

```
7477.806174114189
```

```python
y_true = y
y_pred = lr.predict(X)
ax = sns.scatterplot(x=y_true, y=y_pred, alpha=0.5)
ax.set_xlabel('Actual Target Value')
ax.set_ylabel('Predicted Target Value')
ax.plot( alpha=0.3, lw=1);
```



Here we are using linear regressor for predicting the price values and we find that the mean absolute error for the testing data is 7448.2686 and for the training data is 7477.8061 which is nearby the same and thus we can say that the linear regressor is a good model for predicting the price for the dataset.

```
from sklearn.tree import DecisionTreeRegressor
```

```
dt = DecisionTreeRegressor(max_depth=12)
dt.fit(X_train, y_train)
```

```
DecisionTreeRegressor(max_depth=12)
```

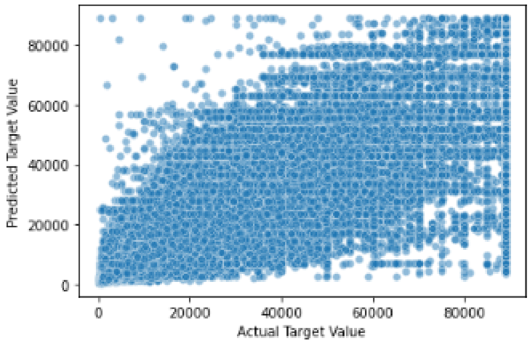
```
mean_absolute_error(dt.predict(X_test), y_test)
```

```
4229.524607765143
```

```
mean_absolute_error(dt.predict(X_train), y_train)
```

```
4058.6250424298387
```

```
y_true = y
y_pred = dt.predict(X)
ax = sns.scatterplot(x=y_true, y=y_pred, alpha=0.5)
ax.set_xlabel('Actual Target Value')
ax.set_ylabel('Predicted Target Value')
ax.plot( alpha=0.3, lw=1);
```



Here we are using the decision tree regressor which provides the mean absolute error of 4229.52 for testing data and 4058.625 for training data which are quite similar to each other as they have a mere difference of 170 and also the mean error value is also quite less compared to the other models used above. We are using the depth as 12 which we obtained as the best predictor using the grid search and rank explained in the model building section.

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

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

```
RandomForestRegressor()
```

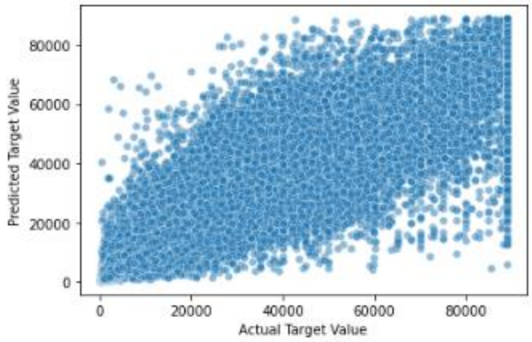
```
mean_absolute_error(rf.predict(X_test), y_test)
```

```
4262.68061878917
```

```
mean_absolute_error(rf.predict(X_train), y_train)
```

```
2021.8025071765478
```

```
y_true = y
y_pred = rf.predict(X)
ax = sns.scatterplot(x=y_true, y=y_pred, alpha=0.5)
ax.set_xlabel('Actual Target Value')
ax.set_ylabel('Predicted Target Value')
ax.plot( alpha=0.3, lw=1);
```



By using random forest regressor as a model to predict the price for the cars we have got the mean absolute error of 4262.68 for the testing data and mean absolute error of 2021.80 for the training data. In the model we have got the error rate quite less from all the others model but the difference between the mean absolute error of testing and training data is quite bigger when compared with all the other models.

From all the above models and their analysis we can conclude the decision tree regressor having the max depth of 12 is the best model for predicting price for the cars with the given advert dataset. As compared with all the other models decision tree regressor has a very small difference in the absolute mean error as well it has a lower mean absolute value when compared with other models (except random forest regressor).

```
from sklearn.tree import DecisionTreeRegressor
```

```
dt = DecisionTreeRegressor(max_depth=12)
dt.fit(X_train, y_train)
```

```
DecisionTreeRegressor(max_depth=12)
```

```
mean_absolute_error(dt.predict(X_test), y_test)
```

```
4229.524607765143
```

```
mean_absolute_error(dt.predict(X_train), y_train)
```

```
4058.6250424298387
```

```
y_true = y
y_pred = dt.predict(X)
ax = sns.scatterplot(x=y_true, y=y_pred, alpha=0.5)
ax.set_xlabel('Actual Target Value')
ax.set_ylabel('Predicted Target Value')
ax.plot( alpha=0.3, lw=1);
```