Project: Car Advert Dataset Price Analysis and prediction

Table of Contents

Introduction

Car Advert Dataset Price Analysis and prediction: I will be working with a dataset provided by AutoTrader, which contains anonymized car sale adverts with information on various features such as brand, type, color, mileage, and selling price. My task is to perform a structured set of tasks to uncover interesting associations and group differences that have a significant impact on the valuation of vehicles. I am looking forward to using my knowledge and skills in data understanding, exploration, preparation, and hypothesis testing to uncover valuable insights from this dataset. I am eager to dive into the world of data science and see what insights I can uncover from this dataset. The Dataset originally contained more than 400,000 but i am working with a sampled 200,000 rows.

Columns Descriptions

00 - public_reference:

- This is a unique identifier used for listing products on the autotrader website. The value is represented as an integer in the dataset.

01 - mileage:

- This represents the annual mileage of a car in miles and you can select a specfic mileage to view the corresponding price. The value is represented as a Float in the dataset.

02 - reg_code:

- The registration code is used as an extra descriptor for what year the car was registered. An example can be seen on for this car. On the autotrader website, the reg code is also used by customers to check the previous owner(s) of a car. - link The value is represented as an object in the dataset.

03 - standard_colour:

- This represents the color of the external chasis of the car. The value is represented as an object in the dataset.

04 - standard_make:

- This represents the brand/manufacturer of a car. The value is represented as an object in the dataset.

05 - standard_model:

- This represents the name of a specific vehicle type of a brand. The value is represented as an object in the dataset.

06 - vehicle_condition:

- This represents the condition of a car. The value is represented as an object in the dataset.

07 - year_of_registration:

- Year of registration refers to the date the vehicle was registered. The value is represented as an Float in the dataset.

08 - price:

- This is the value of a car as listed on the website in pounds sterling. The value is represented as an integer in the dataset.

09 - body_type:

- This refers to the style of the car's chasis. The value is represented as an object in the dataset.

10 - crossover_car_and_van:

- This specifies an additional body_type of a car and represents cars that are either crossovers and vans or not. The value is represented as an bool in the dataset.

11 - fuel_type:

- This represents the type of energy used to power the car. The value is represented as an object in the dataset.

IMPORTING ALL NECESSARY PACKAGES AND LIBRARIES

In [1]:
```
# load datasets
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_style('darkgrid')


import warnings
```

```
# ignore all warnings
warnings.filterwarnings("ignore")

df = pd.read_csv(
    'https://raw.githubusercontent.com/Brunchcode/Car-Advert-Dataset-Price-A
                )
```

In [2]:
```
# identify the columns you want to keep based on a certain condition
cols_to_drop = df.columns[df.columns.str.contains('Unnamed: 0')]

# drop the columns you don't want to keep
df = df.drop(cols_to_drop, axis=1)
```

In [3]:
```
# Lets confirm we have our data in the notebook
adv = df.copy()

adv.head()
```

Out[3]:

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model
0	202010285527283	139000.0	10	Black	Toyota	Prius
1	202010305616862	45591.0	66	Grey	Volkswagen	Sharan
2	202010155020300	53913.0	67	Grey	Vauxhall	Insignia
3	202005219452018	0.0	NaN	Grey	Mitsubishi	Eclipse Cross
4	202008142474829	0.0	NaN	Blue	Vauxhall	Corsa

In [4]:
```
adv.shape
```

Out[4]:
```
(200000, 12)
```

This indicates that the autoTrader dataframe has 200000 rows and 12 columns. This means there are 402005 observations (e.g. car adverts) in the dataframe and each observation has 12 features (e.g. make, model, year, etc.). This information can be useful for understanding the size and structure of the dataset and can inform further analysis.

Data Wrangling

General Properties

In [5]:
```
# Loading dataframe and Performing general operations to inspect data
# types and look for instances of missing or possibly errant data.
# Lets plot the historian data for the no show appointments
adv.head(10)
```

Out[5]:

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model
0	202010285527283	139000.0	10	Black	Toyota	Prius
1	202010305616862	45591.0	66	Grey	Volkswagen	Sharan
2	202010155020300	53913.0	67	Grey	Vauxhall	Insignia
3	202005219452018	0.0	NaN	Grey	Mitsubishi	Eclipse Cross
4	202008142474829	0.0	NaN	Blue	Vauxhall	Corsa
5	202009304416322	94200.0	62	Grey	MINI	Countryman
6	202010094804020	27158.0	67	Multicolour	Audi	A5 Cabriolet
7	202010064652628	33016.0	66	Black	Nissan	Juke
8	202009304394214	28412.0	65	Black	Nissan	Qashqa
9	202010245375713	52169.0	18	White	Renault	Clic

Observing the dataframe, the public reference can be made into the DataFrame's (row) index.But lets check the number of unique values.

In [6]:
```
#checking the number pf unique values
adv['public_reference'].nunique()
```

Out[6]:
```
200000
```

We will have to drop the public reference column as it has 200000 unique values hence it does not help our analysis

In [7]:
```
#lets check the datatypes of all the features
adv.dtypes
```

Out[7]:
```
public_reference          int64
mileage                 float64
reg_code                 object
standard_colour          object
standard_make            object
standard_model           object
vehicle_condition        object
year_of_registration    float64
price                     int64
body_type                object
crossover_car_and_van      bool
fuel_type                object
dtype: object
```

As we can see from the observation of the datatypes, The following features are:

- Quantitative features: 'mileage', 'year_of_registration' and 'price'
- Categorical features: 'reg_code', 'standard_colour', 'standard_make', 'standard_model', 'vehicle_condition', 'body_type', 'crossover_car_and_van', fuel_type

Quantitative features of the dataset

Analysing quantitative features in the dataset, lets take a subset of the adv, which will have the quantitative features(mileage, year_of_registration and price)

In [8]:
```
#creating a list of quantitative features
quantitative_features = ['mileage', 'year_of_registration', 'price']

#creating a dataframe of the quantitative features
quantitative_df = adv[quantitative_features]
quantitative_df.head(5)
```

Out[8]:

	mileage	year_of_registration	price
0	139000.0	2010.0	5190
1	45591.0	2016.0	14991
2	53913.0	2017.0	10351
3	0.0	NaN	25595
4	0.0	NaN	14576

In [9]:
```
quantitative_df.describe(include='all')
```

Out[9]:

	mileage	year_of_registration	price
count	199937.000000	183437.000000	2.000000e+05
mean	37898.275792	2014.966125	1.743430e+04
std	35026.333087	8.985369	5.715433e+04
min	0.000000	999.000000	1.200000e+02
25%	10558.000000	2013.000000	7.495000e+03
50%	28900.000000	2016.000000	1.250000e+04
75%	57000.000000	2018.000000	1.999900e+04
max	999999.000000	2020.000000	9.999999e+06

In [10]:
```
#checking the skewness value
quantitative_df.skew()
```

Out[10]:
```
mileage                   1.602010
year_of_registration    -84.436721
price                   138.364002
dtype: float64
```

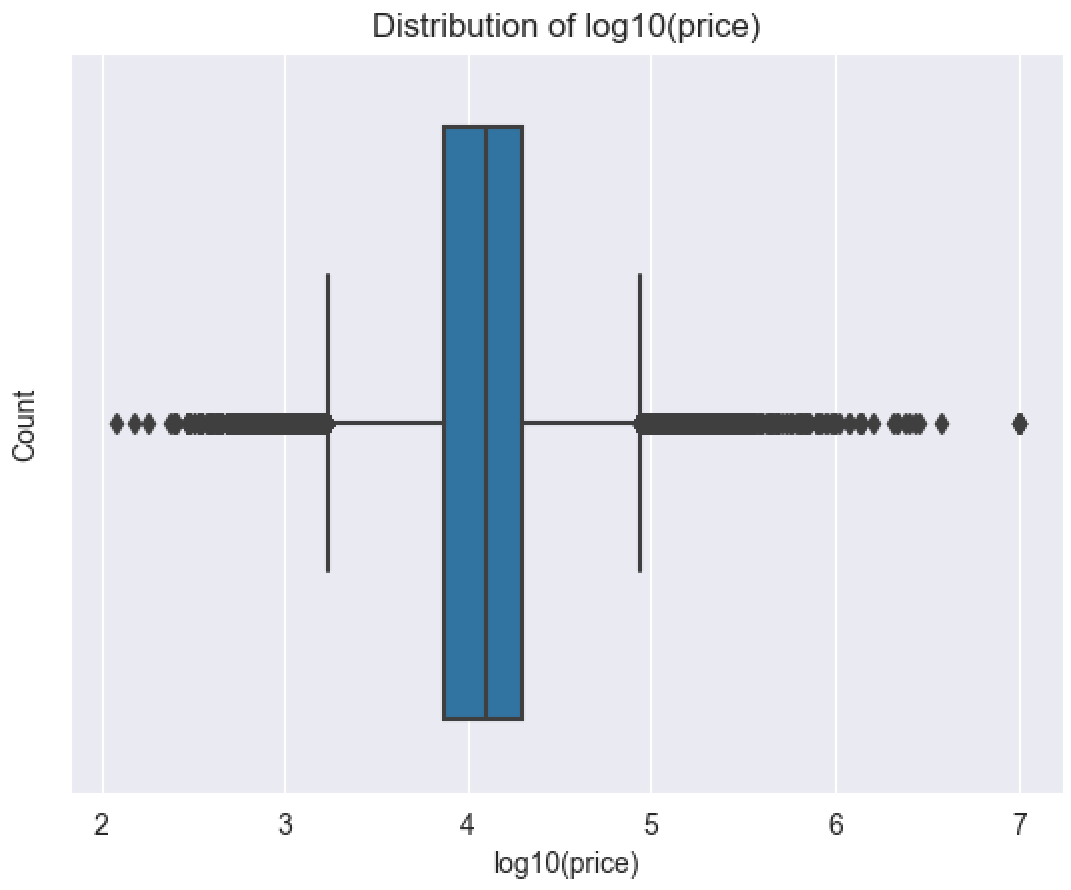
- mileage - the skewness value of 1.602010 for the mileage column suggests that the distribution of mileage values is skewed to the right, meaning that there are more values on the higher end of the range and fewer on the lower end.
- year of registration - The negative skewness value of -84.436721 for the year_of_registration column suggests that the distribution is skewed to the left, meaning that there are more values on the lower end of the range (older cars) and fewer on the higher end (newer cars). This skewness value indicates that there are a large number of older cars in the dataset, and relatively fewer newer cars. This might indicate that the dataset has a large number of used cars, and relatively fewer new cars.

- price - The skewness value of 138.364002 for the price column suggests that the distribution is heavily skewed to the right, meaning that there are a large number of lower-priced cars and a relatively small number of higher-priced cars.

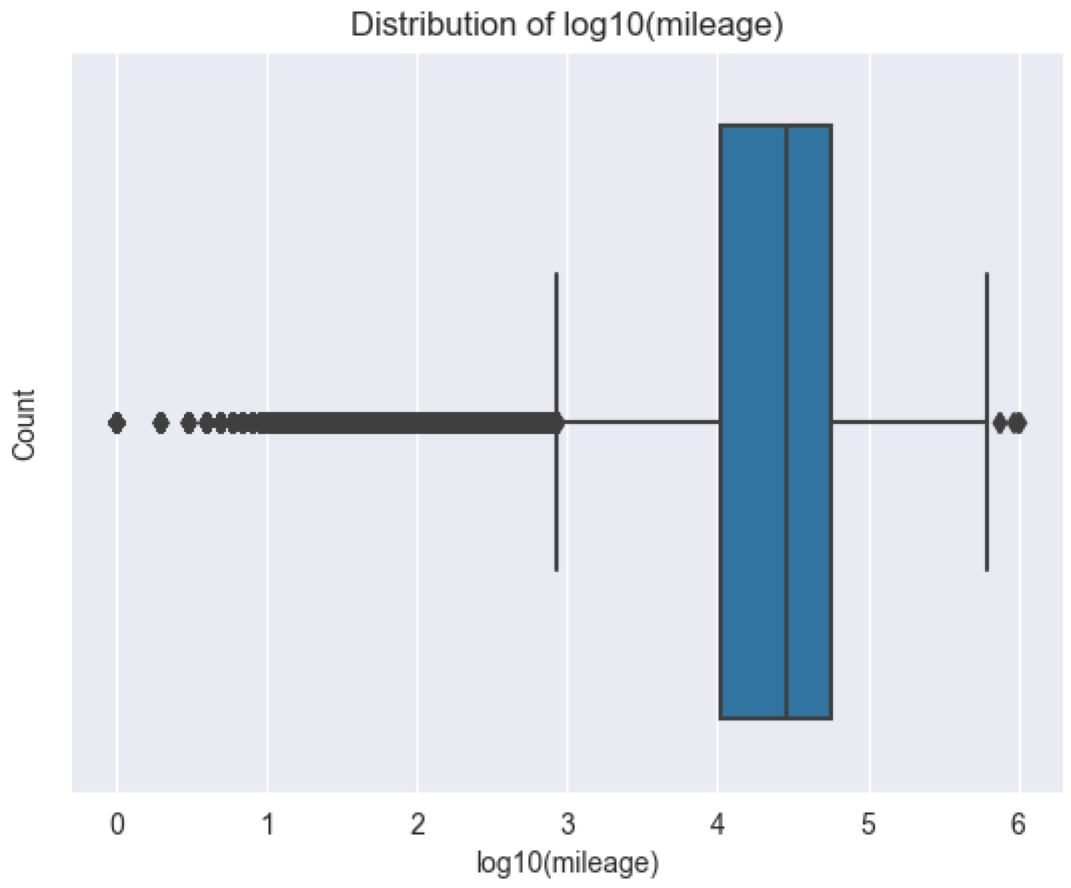
This can be further confirmed by checking the Boxplots of the quantitative features below

In [11]:
```
#Creating a function to plot
def boxplotter (column, x_label, y_label, title):

    if column == 'year_of_registration':
        column_log10 = quantitative_df[column]

        # create a box plot of the price_log10 variable
        sns.boxplot(x = column_log10)

        #add x-axis label
        plt.xlabel(x_label)

        #add y-axis label
        plt.ylabel(y_label)

        #add title
        plt.title(title)
    else:
        column_log10 = np.log10(quantitative_df[column])

        # create a box plot of the price_log10 variable
        sns.boxplot(x = column_log10)

        #add x-axis label
        plt.xlabel(x_label)

        #add y-axis label
        plt.ylabel(y_label)

        #add title
        plt.title(title)
    return
```

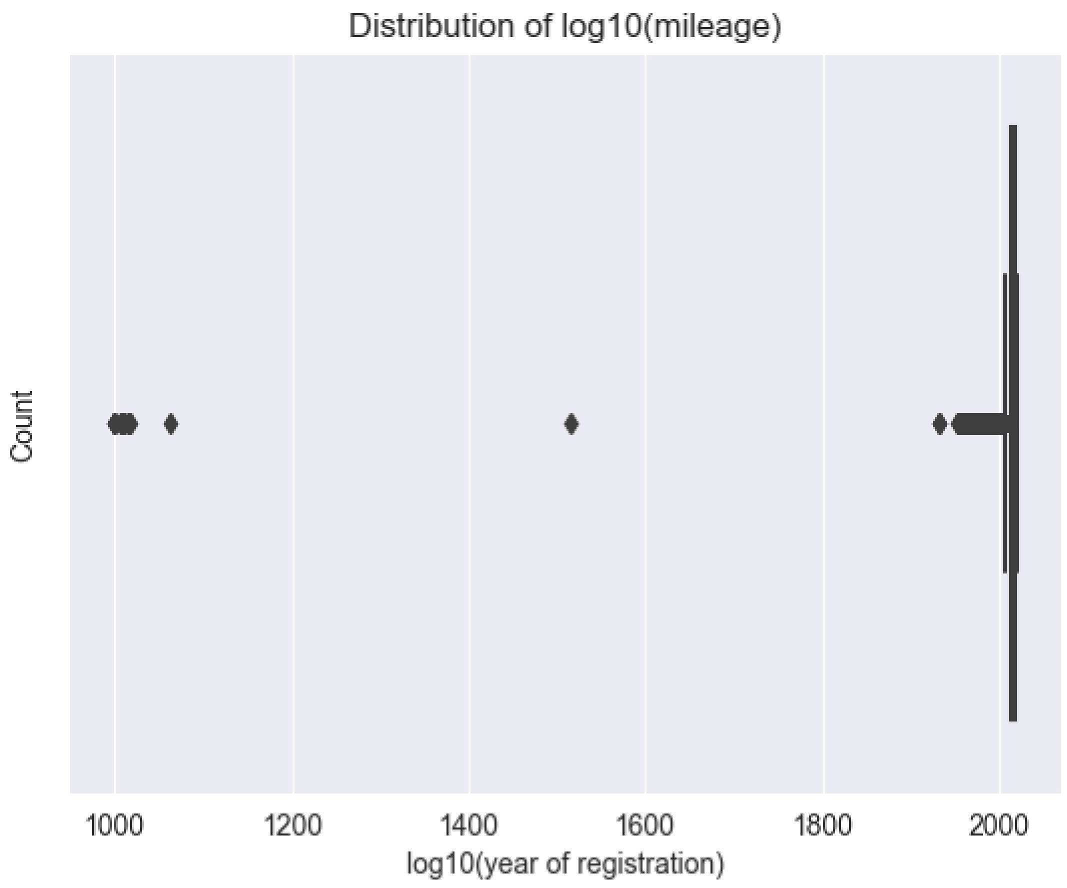
In [12]:
```
boxplotter('price', "log10(price)", "Count", "Distribution of log10(price)")
```



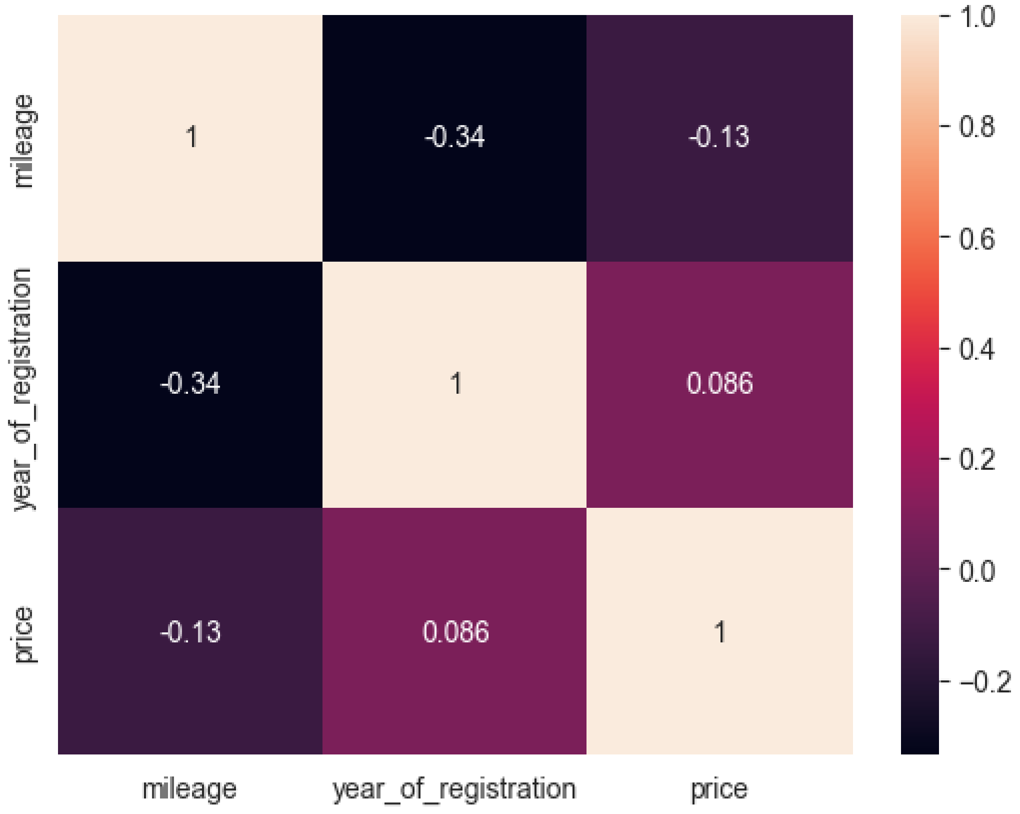
In [13]:
```
boxplotter('mileage', "log10(mileage)", "Count", "Distribution of log10(mile
```



In [14]:
```
boxplotter('year_of_registration', "log10(year of registration)", "Count", "
```



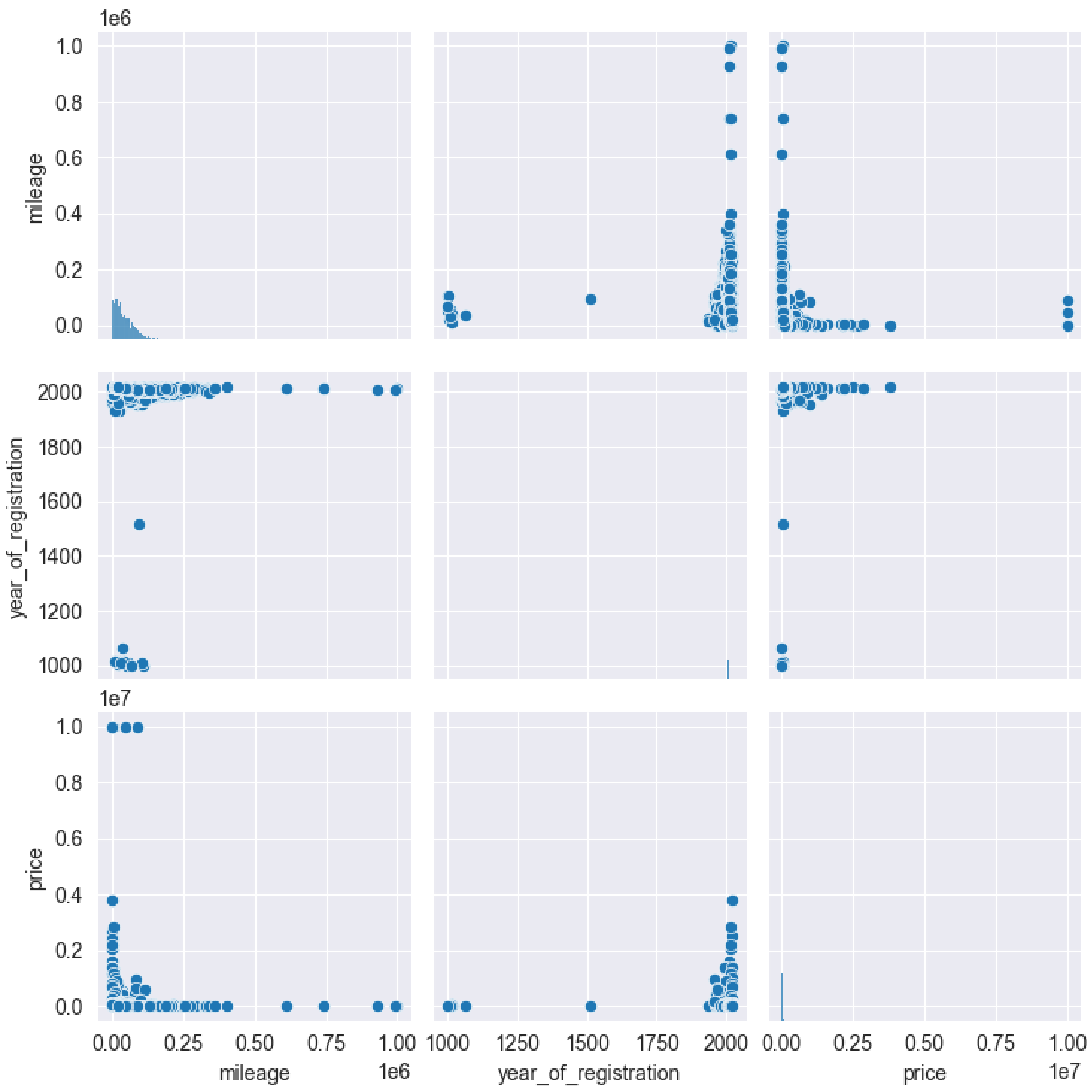
In [15]:
```
sns.heatmap(quantitative_df.corr(), annot=True)
plt.show()
```



- correlation coefficient of -0.13 between mileage and price is an indicator of the relationship between these two variables, it's a weak negative correlation, meaning that as the mileage of a car increases, the price of the car decreases, but the correlation is very weak, mileage of a car does not have a large effect on the price of a car.

- the correlation coefficient of 0.086 between year of registration and price is an indicator of the relationship between these two variables, it is a weak positive correlation, meaning that as the year of registration of a car increases, the price of the car increases too, but the correlation is weak, year of registration of a car does not have a large effect on the price of a car.
- the correlation coefficient of -0.34 between year of registration and mileage is an indicator of the relationship between these two variables, it is a moderate negative correlation, meaning that as the year of registration of a car increases, the mileage of the car decreases, meaning that newer cars are driven less than older cars.

In [16]:
```
sns.pairplot(quantitative_df)
plt.show()
```



observing the relationship from an initial glance of the pairplot, we can see that

- price vs mileage: The higher price, the lower the mileage of the car
- price vs year_of_registration: The higher the price, the higher the year_of_registration

Categorical features of the dataset

In [17]:
```
#creating a list of categorical features
cat_features = ['reg_code',
                'standard_colour',
                'standard_make',
                'standard_model',
                'vehicle_condition',
                'body_type',
                'crossover_car_and_van',
                'fuel_type'
               ]

#creating a dataframe of Categorical features
cat_df = adv[cat_features]
cat_df.head(5)
```

Out[17]:

	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	body_typ
0	10	Black	Toyota	Prius	USED	Hatchbac
1	66	Grey	Volkswagen	Sharan	USED	MP
2	67	Grey	Vauxhall	Insignia	USED	Hatchbac
3	NaN	Grey	Mitsubishi	Eclipse Cross	NEW	SU
4	NaN	Blue	Vauxhall	Corsa	NEW	Hatchbac

In [18]:
```
#There are 8 categorical features
cat_df.dtypes
```

Out[18]:
```
reg_code                 object
standard_colour          object
standard_make            object
standard_model           object
vehicle_condition        object
body_type                object
crossover_car_and_van      bool
fuel_type                object
dtype: object
```

Creating a funtion for getting the value count of the Data in a Categorical feature

In [19]:
```
def value_counts(dataframe,column):
    '''
    This function returns the value_counts of the specified column of a data

    Parameters:
    dataframe : dataframe (pandas dataframe)
    column : str (column name on which value_counts is to be applied)

    Returns:
    value_counts : pandas series

    '''
    # value_counts of the column passed in the function argument
    value_counts = dataframe[column].value_counts()
    return value_counts
```

Creating a funtion for plotting the Data in a Categorical feature

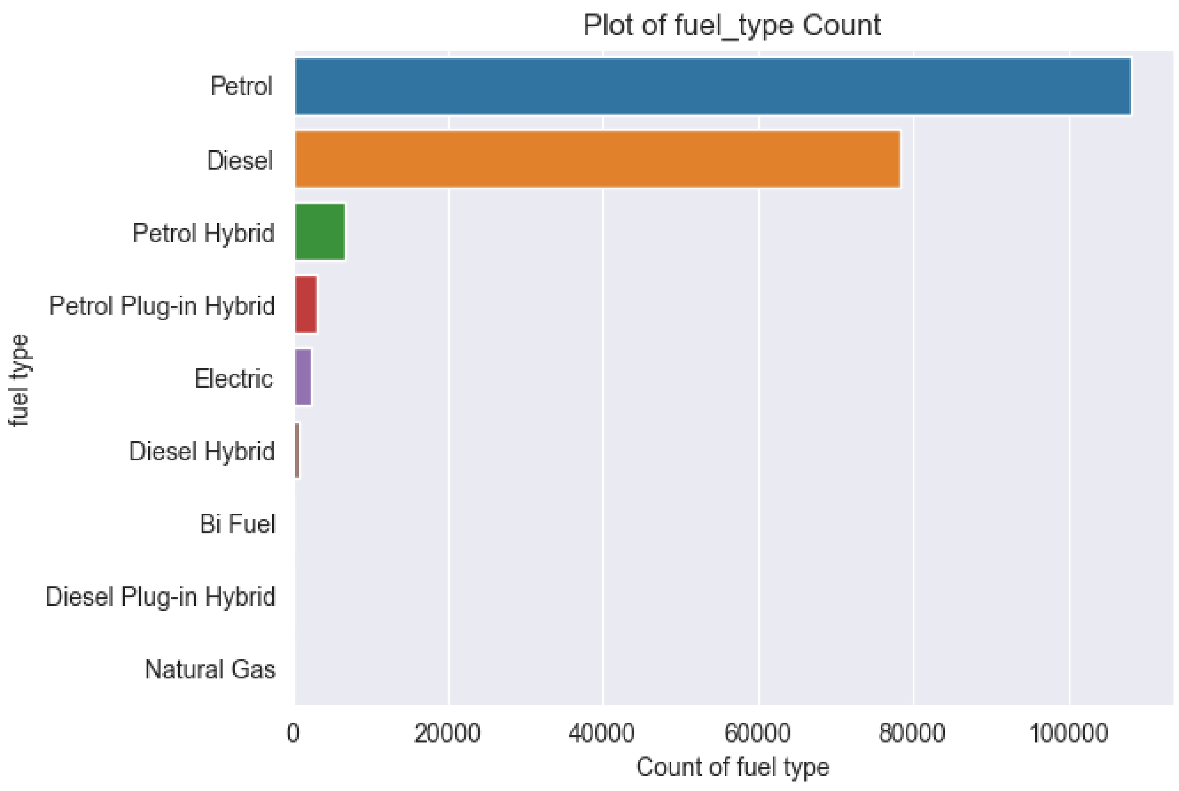
In [20]:
```
def countplot(dataframe, column, title, xlabel, ylabel):
    # sort the column in descending order
    sorted_column = dataframe[column].value_counts().sort_values(ascending=F
    # create a countplot using seaborn countplot method and y-axis as the sc
    ax = sns.countplot(y=dataframe[column], order=sorted_column)
    # set the title of the plot as the title passed in the function argument
    ax.set_title(title)
    # set the x-axis label as the xlabel passed in the function argument
    ax.set_xlabel(xlabel)
    # set the y-axis label as the ylabel passed in the function argument
    ax.set_ylabel(ylabel)
    # show the final plot
    plt.show()
```

In [21]:
```
#Value_counts in 'fuel_type'
value_counts(cat_df,'fuel_type')
```

Out[21]:
```
Petrol                    108064
Diesel                     78431
Petrol Hybrid               6781
Petrol Plug-in Hybrid       3101
Electric                    2418
Diesel Hybrid                699
Bi Fuel                      110
Diesel Plug-in Hybrid         85
Natural Gas                    1
Name: fuel_type, dtype: int64
```

Observing that in the fuel_type column and Petrol, Diesel are the most fuel type that vehicle use

In [22]:
```
#countplot of 'Vehicle_condition'
countplot(cat_df, 'fuel_type',
          'Plot of fuel_type Count',
          'Count of fuel type',
          'fuel type'
         )
```



In [23]:
```
#proportions of fuel type
fuel_type_prop = cat_df['fuel_type'].value_counts() / cat_df['fuel_type'].cc
fuel_type_prop
```

Out[23]:
```
Petrol                   0.541159
Diesel                   0.392764
Petrol Hybrid            0.033958
Petrol Plug-in Hybrid    0.015529
Electric                 0.012109
Diesel Hybrid            0.003500
Bi Fuel                  0.000551
Diesel Plug-in Hybrid    0.000426
Natural Gas              0.000005
Name: fuel_type, dtype: float64
```

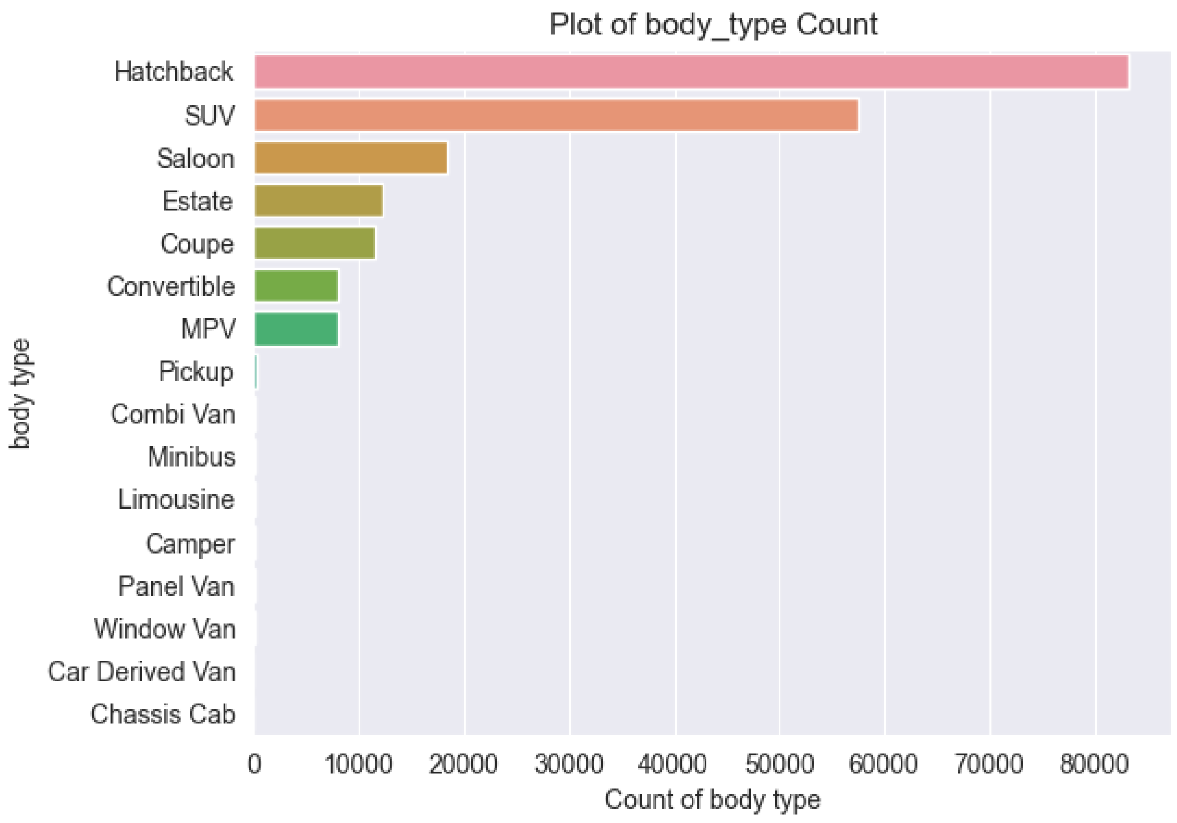
- 54.11% of the cars in the dataset use petrol
- 39.27% of the cars in the dataset use diesel

In [24]:
```
#Value_counts in 'fuel_type'
value_counts(cat_df,'body_type')
```

Out[24]:
```
Hatchback          83169
SUV                57520
Saloon             18381
Estate             12261
Coupe              11572
Convertible         8075
MPV                 7977
Pickup               298
Combi Van             98
Minibus               76
Limousine             67
Camper                41
Panel Van             35
Window Van            18
Car Derived Van        1
Chassis Cab            1
Name: body_type, dtype: int64
```

Hatchback, SUV, Saloon, and Estate are the most popular body type of vehicles respectively

In [25]:
```
#countplot of 'Vehicle_condition'
countplot(cat_df, 'body_type',
          'Plot of body_type Count',
          'Count of body type',
          'body type'
         )
```



In [26]:
```
#proportions of fuel type
body_type_prop= cat_df['body_type'].value_counts() / cat_df['body_type'].cou
body_type_prop
```

Out[26]:
```
Hatchback           0.416699
SUV                 0.288191
Saloon              0.092094
Estate              0.061431
Coupe               0.057979
Convertible         0.040458
MPV                 0.039967
Pickup              0.001493
Combi Van           0.000491
Minibus             0.000381
Limousine           0.000336
Camper              0.000205
Panel Van           0.000175
Window Van          0.000090
Car Derived Van     0.000005
Chassis Cab         0.000005
Name: body_type, dtype: float64
```

- 41.66% of the cars in the dataset are of body type Hatchback
- 28.81% of the cars in the dataset are of body type SUV
- 9.20% of the cars in the dataset are of body type Saloon
- 6.14% of the cars in the dataset are of body type Estate

Observations

Browsing through the dataframe we can see some important columns like the mileage', 'standard_colour', 'standard_make','standard_model', 'vehicle_condition', 'year_of_registration', 'price', 'crossover_car_and_van', 'fuel_type' which are factors that can help with our analysis.

These columns help us ask questions like

- What's the correlation between the price and other features? What feature influences the price of cars?
- Does mileage impact the value of cars?
- What is the average price of vehicles by body type?
- what is the average price of vehicles by fuel_type?
- Does the year of registration affect the average price

In [27]:
```
# Let's confirm the total number of rows and columns
adv.shape
```

Out[27]:
```
(200000, 12)
```

In [28]:
```
# General information about the noshowappointments dataframe
adv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 12 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   public_reference       200000 non-null  int64  
 1   mileage                199937 non-null  float64
 2   reg_code               184150 non-null  object 
 3   standard_colour        197308 non-null  object 
 4   standard_make          200000 non-null  object 
 5   standard_model         200000 non-null  object 
 6   vehicle_condition      200000 non-null  object 
 7   year_of_registration   183437 non-null  float64
 8   price                  200000 non-null  int64  
 9   body_type              199590 non-null  object 
 10  crossover_car_and_van  200000 non-null  bool   
 11  fuel_type              199690 non-null  object 
dtypes: bool(1), float64(2), int64(2), object(7)
memory usage: 17.0+ MB
```

Data Cleaning (Dealing with Missing values, outliers and Noise) of the Car Advert Dataset

The following actions have to be performed on the dataset columns:

- Drop the public reference column
- Deal with the missing values in the following column - year_of_registration
- Drop the NEW cars in the vehicle_condition which effectively drops the entire column
- Deal with the missing values in the following columns - reg_code
- Deal with the missing values in the following columns mileage, standard_colour, body_type, and fuel_type
- Deal with error values in year of registration column e.g 999
- Detecting outliers using interquatile range and dropping all outliers from the dataset

In [29]:
```
# General description
adv.describe(include = 'all')
```

Out[29]:

	public_reference	mileage	reg_code	standard_colour	standard_make	stand
count	2.000000e+05	199937.000000	184150	197308	200000	
unique	NaN	NaN	64	22	95	
top	NaN	NaN	17	Black	BMW	
freq	NaN	NaN	18262	42901	18754	
mean	2.020070e+14	37898.275792	NaN	NaN	NaN	
std	1.724386e+10	35026.333087	NaN	NaN	NaN	
min	2.013092e+14	0.000000	NaN	NaN	NaN	
25%	2.020090e+14	10558.000000	NaN	NaN	NaN	
50%	2.020093e+14	28900.000000	NaN	NaN	NaN	
75%	2.020102e+14	57000.000000	NaN	NaN	NaN	
max	2.020110e+14	999999.000000	NaN	NaN	NaN	

Observing the dataframe, the public reference can be dropped as it has 2000000 unique values.

In [30]:
```
# drop the columns you don't want to keep
adv = adv.drop('public_reference', axis=1)
adv.head(1)
```

Out[30]:

	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition
0	139000.0	10	Black	Toyota	Prius	USED

We can notice that there are missing values in the dataset, The missing values are in the following columns mileage, reg_code, standard_colour, year_of_registration, body_type, and fuel_type

In [31]:
```
# confirming all datatypes
adv.dtypes
```

Out[31]:
```
mileage                  float64
reg_code                  object
standard_colour           object
standard_make             object
standard_model            object
vehicle_condition         object
year_of_registration     float64
price                      int64
body_type                 object
crossover_car_and_van       bool
fuel_type                 object
dtype: object
```

In [32]:
```
#the dataframe without null values
adv[adv.isna().any(axis=1)].shape
```

Out[32]:
```
(19187, 11)
```

In [33]:
```
#checking for the total number of missing values in the dataset
adv.isnull().sum().sum()
```

Out[33]:
```
35888
```

In [34]:
```
#checking for the missing values in each features
adv.isnull().sum().sort_values(ascending=False)
```

Out[34]:
```
year_of_registration     16563
reg_code                 15850
standard_colour           2692
body_type                  410
fuel_type                  310
mileage                     63
standard_make                0
standard_model               0
vehicle_condition            0
price                        0
crossover_car_and_van        0
dtype: int64
```

Dealing with year_of_registration

In [35]:
```
adv.year_of_registration.isnull().sum()
```

Out[35]:
```
16563
```

There are 16563 missing values in the year of registration

It was Observed that

1. From the https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_the_United_Kingdom there exist a relationship between reg_code and year_of_registration.
2. it was also noticed that in the vehicle_condition, NEW vehicles do not have have year_of_registration and reg_code. To help with the analysis of the valuation of prices through the dataframe will with drop the NEW cars in the vehicle_condition which effectively drops the entire column

In [36]:
```
#subsetting the year of registration is null, reg_code is null, and NEW conc
adv[adv['year_of_registration'].isnull() & (adv['vehicle_condition'] == 'NEW
```

Out[36]:

	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_conc
3	0.0	NaN	Grey	Mitsubishi	Eclipse Cross	
4	0.0	NaN	Blue	Vauxhall	Corsa	
12	0.0	NaN	Blue	Vauxhall	Grandland X	
21	10.0	NaN	Black	BMW	X5	
32	0.0	NaN	Black	SKODA	Superb	
...	...	...	...	...	...	
199963	0.0	NaN	White	Fiat	500	
199968	0.0	NaN	Purple	Mitsubishi	Mirage	
199974	10.0	NaN	White	Mercedes-Benz	A Class	
199988	0.0	NaN	Blue	Land Rover	Range Rover	
199995	0.0	NaN	Grey	Jaguar	I-PACE	

15549 rows × 11 columns

In [37]:
```
#subsetting and dropping the year of registration is null, reg_code is null
adv.drop(adv.loc[(adv['year_of_registration'].isnull()) &
               (adv['vehicle_condition'] == 'NEW') &
               (adv['reg_code'].isnull())].index, inplace=True)
```

In [38]:
```
#Checking to confirm the subset has been dropped
adv[adv['year_of_registration'].isnull() & (adv['vehicle_condition'] == 'NEW
```

Out[38]:

mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	y

In [39]:
```
#checking to confirm that there are no NEW vehicles in the vehicle_condition
adv.vehicle_condition.value_counts()
```

Out[39]:
```
USED    184451
Name: vehicle_condition, dtype: int64
```

In [40]:
```
#dropping vehicle condition as it contains all USED vehicles
adv = adv.drop(["vehicle_condition"], axis=1)
adv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 184451 entries, 0 to 199999
Data columns (total 10 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   mileage                184388 non-null  float64
 1   reg_code               184150 non-null  object 
 2   standard_colour        182287 non-null  object 
 3   standard_make          184451 non-null  object 
 4   standard_model         184451 non-null  object 
 5   year_of_registration   183437 non-null  float64
 6   price                  184451 non-null  int64  
 7   body_type              184069 non-null  object 
 8   crossover_car_and_van  184451 non-null  bool   
 9   fuel_type              184228 non-null  object 
dtypes: bool(1), float64(2), int64(1), object(6)
memory usage: 14.2+ MB
```

In [41]:
```
#checking where year of registration and mileage is null
#we cannot fill the missing values in year of registration because the we ne
#reg code is null in this instance
adv[adv['year_of_registration'].isnull() & (adv['reg_code'].isnull())]
```

Out[41]:

	mileage	reg_code	standard_colour	standard_make	standard_model	year_of_regi
2136	9000.0	NaN	Blue	Ferrari	F40	
2904	26684.0	NaN	Grey	Vauxhall	Corsa	
4010	1726.0	NaN	NaN	Ferrari	812 Superfast	
6516	29300.0	NaN	Grey	Toyota	C-HR	
7157	13548.0	NaN	Silver	Mercedes-Benz	C Class	
...	...	...	...	...	...	
195095	17441.0	NaN	Black	Peugeot	2008	
195429	21376.0	NaN	Grey	Peugeot	208	
195590	19568.0	NaN	Blue	Porsche	911	
196151	35294.0	NaN	Red	Hyundai	Tucson	
198118	10750.0	NaN	Grey	Land Rover	Range Rover Evoque	

156 rows × 10 columns

In [42]:
```
#we should drop the null values in year of registration where the regcode is
adv.drop(adv[adv['year_of_registration'].isnull() &
            (adv['reg_code'].isnull())].index,
        inplace = True
       )

#checking for the dropped rows
adv[adv['year_of_registration'].isnull() & (adv['reg_code'].isnull())]
```

Out[42]:

mileage	reg_code	standard_colour	standard_make	standard_model	year_of_registration

In [43]:
```
#checking for how many null values are left in the year_of_Registration
adv.year_of_registration.isnull().sum()
```

Out[43]:
```
858
```

- initially we had 16563 null values in the year_of_registration and we have reduced the null value to 858
- We know that there is a relationship between year of registration and reg_code based on the link in https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_the_United_Kingdom, therefore we will check for a relationship between reg code and year of registration in our dataset.

In [44]:
```
#taking a sample of 20 rows where reg code is 17 and comparing the values of
#subsetting to confirm relationship of reg_code and year of registration whe
adv[(adv['reg_code'] == '17')]
```

Out[44]:

	mileage	reg_code	standard_colour	standard_make	standard_model	year_of_regi
16	39000.0	17	Grey	BMW	1 Series	
28	18653.0	17	Blue	Vauxhall	Astra	
50	25186.0	17	Black	BMW	3 Series	
71	44622.0	17	Black	BMW	5 Series	
76	31154.0	17	White	Audi	A1	
...	...	...	...	...	...	
199941	17146.0	17	White	BMW	1 Series	
199958	79804.0	17	Blue	Toyota	Avensis	
199976	25421.0	17	White	Toyota	AYGO	
199986	13229.0	17	Blue	Vauxhall	Insignia	
199992	20000.0	17	Black	BMW	1 Series	

18262 rows × 10 columns

As shown above there is a relationship between year of registration and reg_code, as the value of reg_code == 17 is 2017 on year_of_registration

In [45]:
```
#creating a dictionary using the reg_code column and the year_of_registratic
#Create a dictionary mapping the values in column 'reg_code'
#to the corresponding values in column 'year_of_registration'

mapping_dict = adv.set_index('reg_code')['year_of_registration'].to_dict()
mapping_dict
```

Out[45]:
```
{'10': 2010.0,
 '66': 2016.0,
 '67': 2017.0,
 '62': 2012.0,
 '65': 2015.0,
 '18': 2018.0,
 '15': 2015.0,
 '20': 2020.0,
 '04': 2004.0,
 '64': 2015.0,
 '17': 2017.0,
 '11': 2011.0,
 '05': 2005.0,
 '68': 2019.0,
 '54': 2004.0,
 '16': 2016.0,
 '56': 2006.0,
 '59': 2009.0,
 '19': 2019.0,
 '57': 2007.0,
 '69': 2020.0,
 '13': 2013.0,
 '63': 2013.0,
 '09': 2009.0,
 '14': 2014.0,
 '60': 2010.0,
 '70': 2020.0,
 '61': 2011.0,
 '52': 2002.0,
 '12': 2012.0,
 '08': 2008.0,
 '53': 2004.0,
 '06': 2006.0,
 '58': 2008.0,
 '55': 2005.0,
 '07': 2007.0,
 'P': 1997.0,
 'J': 1992.0,
 'D': 1966.0,
 '02': 2002.0,
 'Y': 2014.0,
 '51': 2001.0,
 'R': 1998.0,
 'K': 1993.0,
 'W': 2000.0,
 '03': 2003.0,
 'N': 1996.0,
 'F': 1968.0,
 'G': 1989.0,
 'T': 1999.0,
 'H': 1991.0,
 nan: 2000.0,
 'V': nan,
 'M': 1995.0,
 'B': 1985.0,
 'S': 1998.0,
 'C': 1985.0,
 'X': 2001.0,
 'A': 1963.0,
 'E': 1988.0,
 'L': 1973.0,
 '38': nan,
 '94': nan,
 's': 2001.0,
 'p': 1957.0}
```

In [46]:
```
# Fill null values in column 'year of registration' using the mapping_dict
adv['year_of_registration'] = adv['year_of_registration'].fillna(adv['reg_cc
```

In [47]:
```
#checking the null values left in year
adv.year_of_registration.isnull().sum()
```

Out[47]:
```
4
```

In [48]:
```
#lets see the null values left in the year of registration
adv[adv['year_of_registration'].isnull()]
```

Out[48]:

	mileage	reg_code	standard_colour	standard_make	standard_model	year_of_reg
61579	23157.0	38	Black	Mercedes-Benz	E Class	
69760	61370.0	94	Black	Vauxhall	Mokka	
95918	198014.0	V	Green	Land Rover	Discovery	
196496	9000.0	V	Black	Aston Martin	V8	

- '94': 2044 (from the future which is a used car)
- '38': 2038 (from the future which is a used car)
- 'V' : 1 September 1999 – 29 February 2000

The cars that have their year of registration pointing to the future will be dropped as we are can see from the wikipedia website about UK vehicle registration

In [49]:
```
leftover_dict = {'V': 2000}
leftover_dict
```

Out[49]:
```
{'V': 2000}
```

In [50]:
```
# Fill null values in column 'year of registration' using the mapping_dict
adv['year_of_registration'] = adv['year_of_registration'].fillna(adv['reg_cc

#we should drop the null values in year of registration where the regcode is
adv.drop(adv[adv['year_of_registration'].isnull()].index,
        inplace = True
       )

#checking
adv.year_of_registration.isnull().sum()
```

Out[50]:
```
0
```

Dealing with reg_code

In [51]:
```
#Total number of null values in reg_code
adv.reg_code.isnull().sum()
```

Out[51]:
```
145
```

There are 145 missing values in the reg_code

In the case of the features year_of_registration and reg_code in a dataframe, they are providing the same information in the dataframe. This can cause problems in the data analysis because they may not be able to distinguish the unique contributions of each variable.

To address this problem, we will remove the reg_code.

In [52]:
```
#dropping reg_code as it contains similar or the same information as year_of
adv = adv.drop(["reg_code"], axis=1)
adv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 184293 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   mileage                184232 non-null  float64
 1   standard_colour        182147 non-null  object 
 2   standard_make          184293 non-null  object 
 3   standard_model         184293 non-null  object 
 4   year_of_registration   184293 non-null  float64
 5   price                  184293 non-null  int64  
 6   body_type              183941 non-null  object 
 7   crossover_car_and_van  184293 non-null  bool   
 8   fuel_type              184076 non-null  object 
dtypes: bool(1), float64(2), int64(1), object(5)
memory usage: 12.8+ MB
```

reg_code has been dropped!!!

In [53]:
```
adv.isnull().sum().sort_values(ascending=False)
```

Out[53]:
```
standard_colour          2146
body_type                 352
fuel_type                 217
mileage                    61
standard_make               0
standard_model              0
year_of_registration        0
price                       0
crossover_car_and_van       0
dtype: int64
```

Dealing with standard_colour, body_type, fuel_type, and mileage

Missing values can be filled with the mode (most common value) when the data is categorical or ordinal in nature. This is because the mode represents the value that occurs most frequently in the dataset, which makes it a good estimate for missing values. However, it is important to consider the context and specific characteristics of the dataset before filling missing values with the mode, as it may not always be the best approach.

In [54]:
```
def filler(data, used_col, col_null, usercase = True):
    if usercase is True:
        # create new column with mode of for each country, where there is mo
        data['filled'] = data.groupby(used_col)[col_null].transform(lambda x

        # replace null values in content_rating with mode of the country
        data[col_null] = np.where(data[col_null].isnull(), data['filled'], d
        data.drop('filled', axis=1, inplace=True)

    else:
        data[col_null] = df[col_null].fillna(df.groupby(used_col)[col_null].
```

This code defines a function called "filler" that takes in four parameters: "data", "used_col", "col_null", and "usercase". The "data" parameter is a DataFrame that the function will be applied to. The "used_col" parameter is a column in the DataFrame that will be used to group the data for calculating the mode or mean. The "col_null" parameter is the column in the DataFrame that will have its null values filled in. The "usercase" parameter is a Boolean that determines whether to fill in the null values in "col_null" with the mode of the group or the mean of the group.

If "usercase" is set to True, the function will first create a new column in the DataFrame called "filled" that contains the mode of the values in "col_null" for each group defined by "used_col". Then it will replace any null values in "col_null" with the corresponding value in the "filled" column. Finally, it will drop the "filled" column.

If "usercase" is set to False, the function will fill the null values in "col_null" with the mean of the group defined by "used_col".

This code can be used to fill in missing data in a DataFrame by using the mode or mean of the data for a specific group.

In [55]:
```
#Dealing with standard_colour null values
#checking the total number of null values present in standard_colour
adv.standard_colour.isnull().sum()
```

Out[55]:
```
2146
```

In [56]:
```
#using the filler function to fill standard_colour
filler(adv, 'standard_make', 'standard_colour')

#check for null values
adv.standard_colour.isnull().sum()
```

Out[56]:
```
2
```

In [57]:
```
#Dealing with body_type null values
#checking the total number of null values present in body_type
adv.body_type.isnull().sum()
```

Out[57]:
```
352
```

In [58]:
```
#using the filler function to fill body_type
filler(adv, 'standard_make', 'body_type')

#check for null values
adv.body_type.isnull().sum()
```

Out[58]:
```
4
```

The standard_colour and the body_type have 2 and 4 null values respectively. this null values are cannot be filled by either standard make and model hence lets fill it by using mode in both cases.

In [59]:
```
adv['standard_colour'].fillna(adv['standard_colour'].mode()[0], inplace=True
adv['body_type'].fillna(adv['body_type'].mode()[0], inplace=True)
```

In [60]:
```
adv.isnull().sum().sort_values(ascending=False)
```

Out[60]:
```
fuel_type                  217
mileage                     61
standard_colour              0
standard_make                0
standard_model               0
year_of_registration         0
price                        0
body_type                    0
crossover_car_and_van        0
dtype: int64
```

In [61]:
```
#Dealing with fuel_type null values
#checking the total number of null values present in fuel_type
adv.fuel_type.isnull().sum()
```

Out[61]:
```
217
```

In [62]:
```
#using the filler function to fill fuel_type
filler(adv, 'body_type', 'fuel_type')

#check for null values
adv.fuel_type.isnull().sum()
```

Out[62]:
```
0
```

In [63]:
```
#Dealing with mileage null values
#checking the total number of null values present in mileage
adv.mileage.isnull().sum()
```

Out[63]:
```
61
```

In [64]:
```
#using the filler function to fill mileage
filler(adv, 'year_of_registration', 'mileage', usercase = False)

#check for null values
adv.mileage.isnull().sum()
```

Out[64]:
```
2
```

In [65]:
```
adv[adv['mileage'].isnull()]
```

Out[65]:

	mileage	standard_colour	standard_make	standard_model	year_of_registration	pr
74028	NaN	Blue	Mercedes-Benz	E Class	2010.0	79
174046	NaN	Purple	Vauxhall	Astra	1989.0	49

In [66]:
```
adv['mileage'] = adv['mileage'].fillna(adv['mileage'].mean())

#check for null values
adv.mileage.isnull().sum()
```

Out[66]:
```
0
```

In [67]:
```
adv.isnull().sum().sort_values(ascending=False)
```

Out[67]:
```
mileage                  0
standard_colour          0
standard_make            0
standard_model           0
year_of_registration     0
price                    0
body_type                0
crossover_car_and_van    0
fuel_type                0
dtype: int64
```

All missing values have been sufficiently dealt with

Dealing with Noise and Outliers in the Dataset

In [68]:
```
adv.describe()
```

Out[68]:

	mileage	year_of_registration	price
count	184293.000000	184293.000000	1.842930e+05
mean	41100.220864	2014.969690	1.583748e+04
std	34629.961663	8.976709	2.721082e+04
min	0.000000	999.000000	1.200000e+02
25%	14592.000000	2013.000000	6.999000e+03
50%	32000.000000	2016.000000	1.179900e+04
75%	60000.000000	2018.000000	1.850000e+04
max	999999.000000	2020.000000	3.799995e+06

The outliers are most found in numerical data such as continuous variables (e.g. mileage, price) or discrete variables (e.g. count data). Outliers can have a large impact on the analysis and results of a dataset, and can skew the overall distribution of the data.

Outliers can be caused by various factors such as measurement error, data entry errors, or genuine extreme cases. It's important to identify and handle outliers appropriately, as they can have a significant impact on the statistical properties of a dataset, such as the mean, median, and standard deviation.

in our case we can see:

- the year of registration of 999 which is an error
- A max value of 999999 in mileage
- A max value of 3,799,995 in price

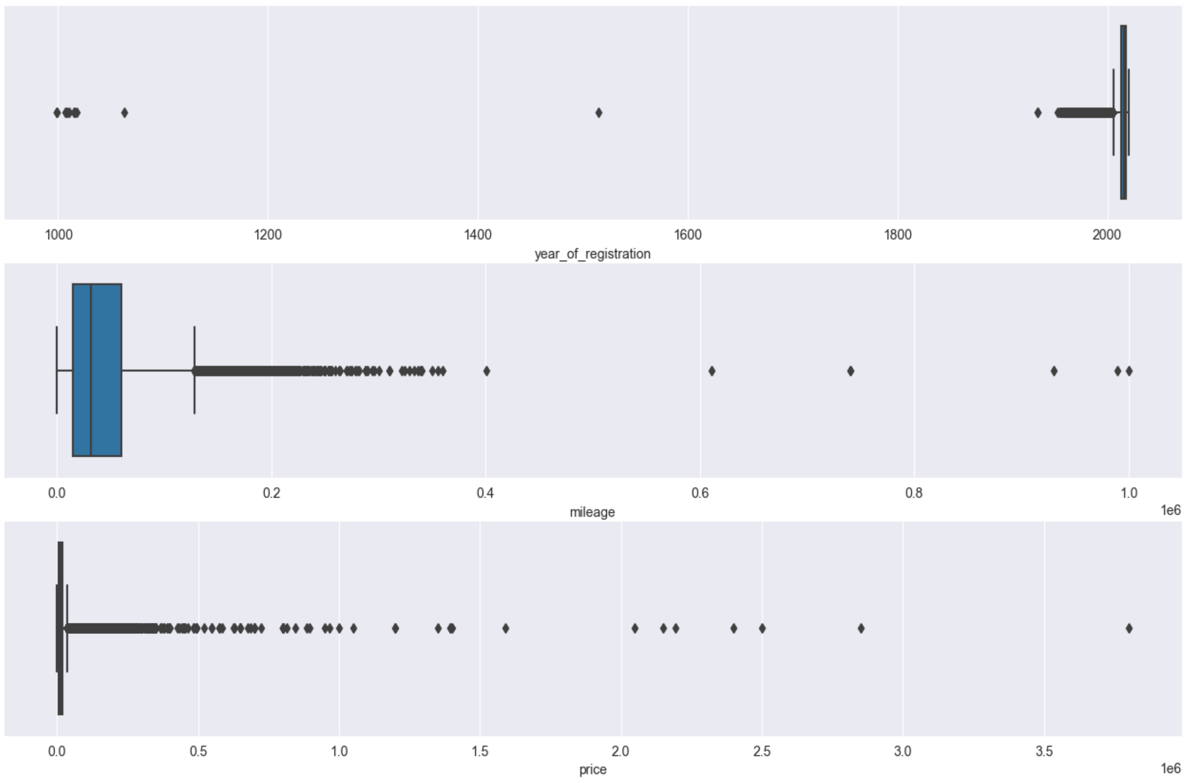
lets visualize:

In [69]:
```
#quick visualization of the year_of_registration, mileage, and price
plt.figure(figsize=(16,10))

#boxplot of year_of_registration
plt.subplot(3,1,1)
sns.boxplot(x = adv['year_of_registration'])

#boxplot of year_of_registration
```

```
plt.subplot(3,1,2)
sns.boxplot(x = adv['mileage'])

#boxplot of year_of_registration
plt.subplot(3,1,3)
sns.boxplot(x = adv['price'])
```

Out[69]:

```
<Axes: xlabel='price'>
```



In [70]:

```
# checking for noise and error values in the year of registration
adv[adv['year_of_registration'] < 1900]
```

Out[70]:

	mileage	standard_colour	standard_make	standard_model	year_of_registration
25918	27200.0	Black	MINI	Clubman	1016.0
46010	107934.0	Blue	Audi	A3	999.0
48041	96659.0	Black	Audi	A4 Avant	1515.0
79415	19000.0	Silver	Mercedes-Benz	C Class	1007.0
97494	104000.0	Silver	BMW	1 Series	1008.0
100890	54569.0	Silver	BMW	Z4	999.0
106032	8600.0	Silver	BMW	M2	1018.0
112005	58470.0	Black	Fiat	Punto Evo	1010.0
113169	39624.0	Red	MINI	Clubman	1015.0
113668	30000.0	Red	Toyota	AYGO	1009.0
150202	37771.0	Black	Smart	fortwo	1063.0
182920	69346.0	Red	Mazda	Mazda3	999.0

In [71]:

```
#Reg code was droppped previously during cleaning,
#we need it to deal with the error values in year of registration
df[df['year_of_registration'] < 1900]
```

Out[71]:

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_
25918	202010064654489	27200.0	66	Black	MINI	C
46010	202010094789497	107934.0	13	Blue	Audi	
48041	202010155035879	96659.0	65	Black	Audi	A
79415	202008042076716	19000.0	57	Silver	Mercedes-Benz	
97494	202010225311657	104000.0	08	Silver	BMW	
100890	202009304380359	54569.0	08	Silver	BMW	
106032	202010134937656	8600.0	68	Silver	BMW	
112005	202010205206488	58470.0	10	Black	Fiat	Pu
113169	202010195174849	39624.0	65	Red	MINI	C
113668	202008102305925	30000.0	59	Red	Toyota	
150202	202009163810376	37771.0	63	Black	Smart	
182920	202010155037484	69346.0	64	Red	Mazda	I

In [72]:

```
#using functions and mapping dictionary to replace value in the column
def replace_value(df, mapping_dict):
    """
    Replace the values in the 'year_of_registration' column of the DataFrame

    Parameters:
        df (DataFrame): Dataframe which needs to be modified
        mapping_dict (dict): Dictionary containing the mapping of old values
    Returns:
        DataFrame : Modified Dataframe
    """
    # Replace the values in the year of registration column using the mappin
    df['year_of_registration'].replace(mapping_dict, inplace=True)
    return df
```

In [73]:

```
#creating a mapping dictionary using the regcode from the UK vehicle registr
fix_error_dict = {
                1016.0: 2016, #66
                999.0: 2014, #13
                1515.0: 2015, #65
                1007.0: 2007, #57
                1008.0: 2008, #08
                999.0: 2008, #08
                1018.0: 2018, #68
                1010.0: 2010, #10
                1015.0: 2015, #65
                1009.0: 2009, #59
                1063.0: 2013, #63
                999.0: 2014, #64
               }
fix_error_dict
```

Out[73]:
```
{1016.0: 2016,
 999.0: 2014,
 1515.0: 2015,
 1007.0: 2007,
 1008.0: 2008,
 1018.0: 2018,
 1010.0: 2010,
 1015.0: 2015,
 1009.0: 2009,
 1063.0: 2013}
```

In [74]:
```
#fixing the error in the year of registration column that has years less tha
adv = replace_value(adv, fix_error_dict)
#check
adv.loc[(adv['year_of_registration'] < 1900)]
```

Out[74]:

mileage	standard_colour	standard_make	standard_model	year_of_registration	price	bo

Dealing with outliers

Interquartile range

In [75]:
```
adv.shape
```

Out[75]:
```
(184293, 9)
```

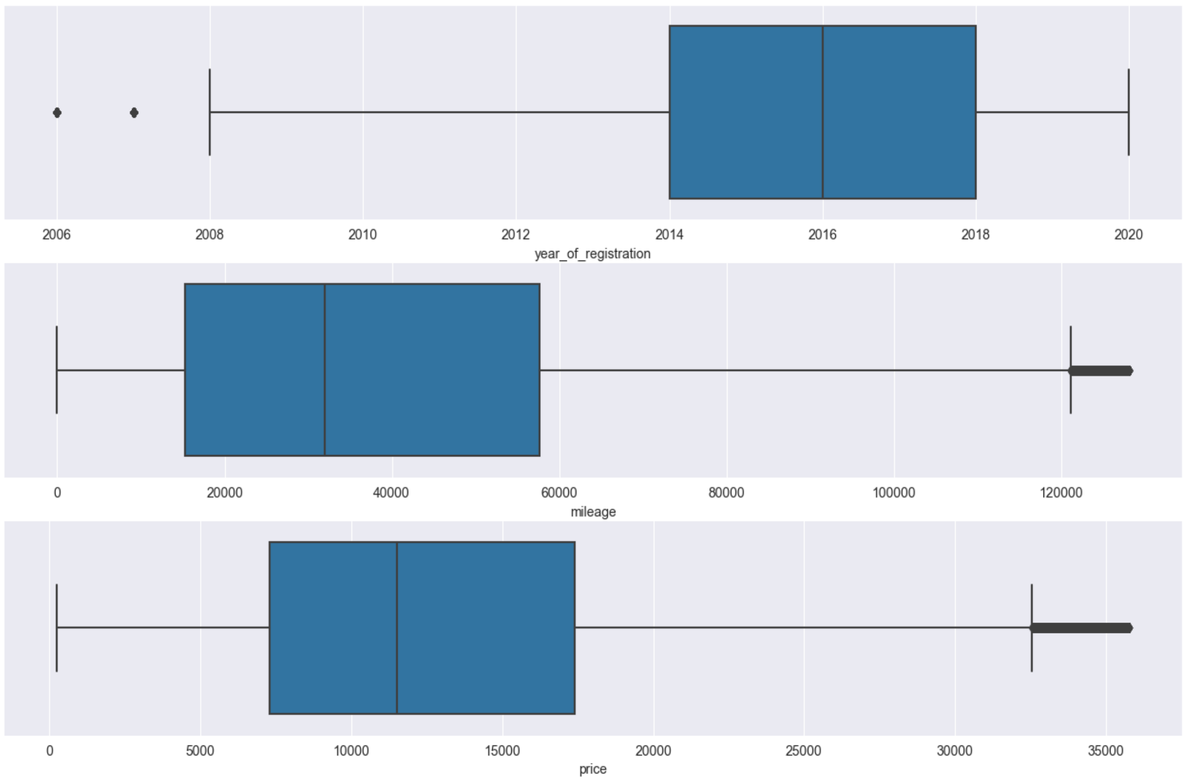
In [76]:
```
# Step 1: Calculate the interquartile range
numeric_cols = adv.select_dtypes(include=[np.number]).columns # select only
Q1 = adv[numeric_cols].quantile(0.25) # first quartile
Q3 = adv[numeric_cols].quantile(0.75) # third quartile
IQR = Q3 - Q1 # interquartile range

# Step 2: Identify the outliers
outliers = ((adv[numeric_cols] < (Q1 - 1.5 * IQR)) | (adv[numeric_cols] > (Q

# Step 3: remove them from the dataset
adv = adv[~outliers.any(axis=1)] # remove outliers from the dataset

# Step 4: check the shape of the dataset
adv.shape
```

Out[76]:
```
(165149, 9)
```

lets visualize the data

In [77]:
```
#quick visualization of the year_of_registration, mileage, and price
plt.figure(figsize=(16,10))

#boxplot of year_of_registration
plt.subplot(3,1,1)
sns.boxplot(x = adv['year_of_registration'])

#boxplot of year_of_registration
plt.subplot(3,1,2)
sns.boxplot(x = adv['mileage'])

#boxplot of year_of_registration
plt.subplot(3,1,3)
sns.boxplot(x = adv['price'])
```

Out[77]:
```
<Axes: xlabel='price'>
```



In [78]:
```
adv.sort_values('price', ascending=False).head(1)
```

Out[78]:

	mileage	standard_colour	standard_make	standard_model	year_of_registration	p
195679	11550.0	White	Mercedes-Benz	GLC Class	2019.0	35

In [79]:
```
adv.sort_values('year_of_registration', ascending=False).head(1)
```

Out[79]:

	mileage	standard_colour	standard_make	standard_model	year_of_registration	p
153765	1632.0	Grey	Kia	Sportage	2020.0	24

In [80]:
```
adv.sort_values('mileage', ascending=False).head(1)
```

Out[80]:

	mileage	standard_colour	standard_make	standard_model	year_of_registration	p
157137	128000.0	Black	Audi	A4 Avant	2011.0	5

We have cleaned up our dataframe. We can now explore the dataframe by performing some Data Transformations

Data Transformation of the Car Advert Dataset

The following data transformation can be performed on the dataset columns:

- Calculating the age of the vehicle based on the current date and the year of registration
- Creating categorical variable from year of registration column
- Creating categorical variable from mileage column

Calculating the age of the vehicle based on the current date and the year of registration

In [81]:
```
# Convert the year_of_registration column to integer
adv['year_of_registration'] = adv['year_of_registration'].astype(int)
adv.dtypes
```

Out[81]:
```
mileage                  float64
standard_colour           object
standard_make             object
standard_model            object
year_of_registration       int64
price                      int64
body_type                 object
crossover_car_and_van       bool
fuel_type                 object
dtype: object
```

In [82]:
```
import datetime

# Assume the year of registration is stored in the 'year_of_registration' co
# and the current date is stored in the 'current_date' column

#this brings the current year which is 2023 but the max year currently on th
current_year = datetime.datetime.now().year

# Create a new column 'age'
adv = adv.assign(age=current_year - adv['year_of_registration'])

adv.head()
```

Out[82]:

	mileage	standard_colour	standard_make	standard_model	year_of_registration	price
1	45591.0	Grey	Volkswagen	Sharan	2016	14991
2	53913.0	Grey	Vauxhall	Insignia	2017	10351
5	94200.0	Grey	MINI	Countryman	2012	6995
6	27158.0	Multicolour	Audi	A5 Cabriolet	2017	24750
7	33016.0	Black	Nissan	Juke	2016	10489

Creating categorical variable from year of registration column.

In [83]:
```
#checking for the max and min of year of registration
adv['year_of_registration'].min(), adv['year_of_registration'].max()
```

Out[83]:
```
(2006, 2020)
```

In [84]:
```
adv.year_of_registration.describe()
```

Out[84]:
```
count    165149.000000
mean       2015.452585
std           3.328769
min        2006.000000
25%        2014.000000
50%        2016.000000
75%        2018.000000
max        2020.000000
Name: year_of_registration, dtype: float64
```

In [85]:
```
# Create a new column 'year_of_registration_category'
adv['condition'] = pd.cut(adv['year_of_registration'],
                          bins=[2006,2012,2021],
                          labels=['OLD','NEW']
```

```
                                    )

adv.head()
```

Out[85]:

	mileage	standard_colour	standard_make	standard_model	year_of_registration	price
1	45591.0	Grey	Volkswagen	Sharan	2016	14991
2	53913.0	Grey	Vauxhall	Insignia	2017	10351
5	94200.0	Grey	MINI	Countryman	2012	6995
6	27158.0	Multicolour	Audi	A5 Cabriolet	2017	24750
7	33016.0	Black	Nissan	Juke	2016	10489

In [86]:

```
# Convert the new vehicle_condition column to object
adv['condition'] = adv['condition'].astype('object')
adv.dtypes
```

Out[86]:

```
mileage                    float64
standard_colour             object
standard_make               object
standard_model              object
year_of_registration         int64
price                        int64
body_type                   object
crossover_car_and_van         bool
fuel_type                   object
age                          int64
condition                   object
dtype: object
```

Creating categorical variable from mileage column.

In [87]:

```
adv['mileage'].min(), adv['mileage'].max()
```

Out[87]:

```
(0.0, 128000.0)
```

In [88]:

```
adv['mileage'].describe()
```

Out[88]:

```
count    165149.000000
mean      39054.724367
std       29768.760412
min           0.000000
25%       15277.000000
50%       31969.000000
75%       57594.000000
max      128000.000000
Name: mileage, dtype: float64
```

In [89]:

```
# Create a new column 'year_of_registration_category'
adv['usage'] = pd.cut(adv['mileage'],
                      bins=[0.0,30000.0,60000.0,125000.0],
                      labels=['LOW','AVERAGE','HIGH'],
                      right=False,
                      include_lowest=True
                     )

adv.head()
```

Out[89]:

	mileage	standard_colour	standard_make	standard_model	year_of_registration	price
1	45591.0	Grey	Volkswagen	Sharan	2016	14991
2	53913.0	Grey	Vauxhall	Insignia	2017	10351
5	94200.0	Grey	MINI	Countryman	2012	6995
6	27158.0	Multicolour	Audi	A5 Cabriolet	2017	24750
7	33016.0	Black	Nissan	Juke	2016	10489

In [90]:
```python
# Convert the new usage column to object
adv['usage'] = adv['usage'].astype('object')
adv.dtypes
```

Out[90]:
```
mileage                  float64
standard_colour           object
standard_make             object
standard_model            object
year_of_registration       int64
price                      int64
body_type                 object
crossover_car_and_van       bool
fuel_type                 object
age                        int64
condition                 object
usage                     object
dtype: object
```

We have performed data transformation on our dataframe. We can now explore the dataframe by performing some exploratory analysis

Exploratory Data Analysis

Q1: Whats the correlation between the price and other features?

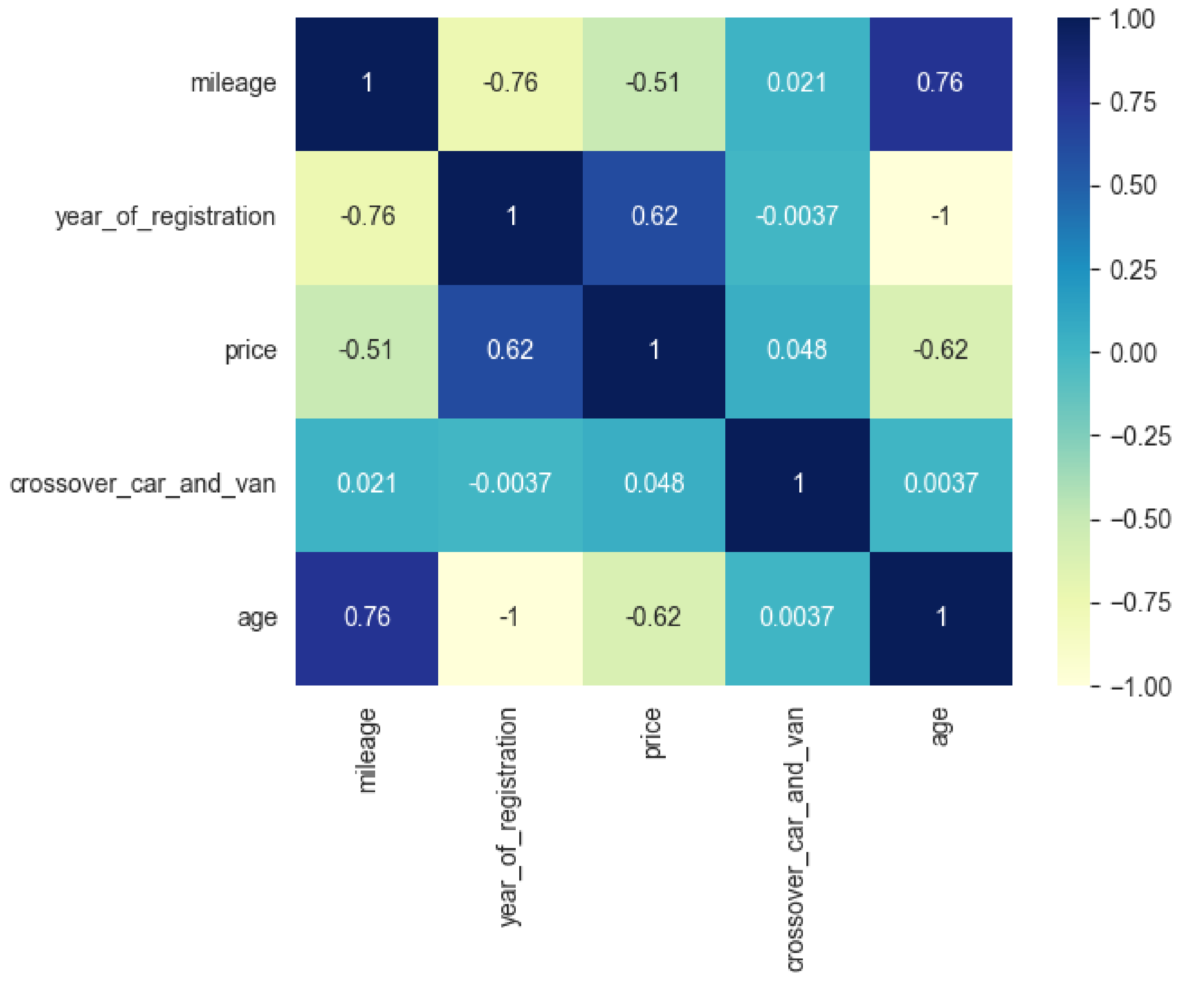
In [91]:
```python
# Select only the numerical columns
numeric_cols = ['price', 'mileage', 'year_of_registration', 'age', 'crossove
adv_numeric = adv[numeric_cols]

# Find the correlation matrix
corr_matrix = adv_numeric.corr()

# Print the correlation matrix
corr_matrix
```

Out[91]:

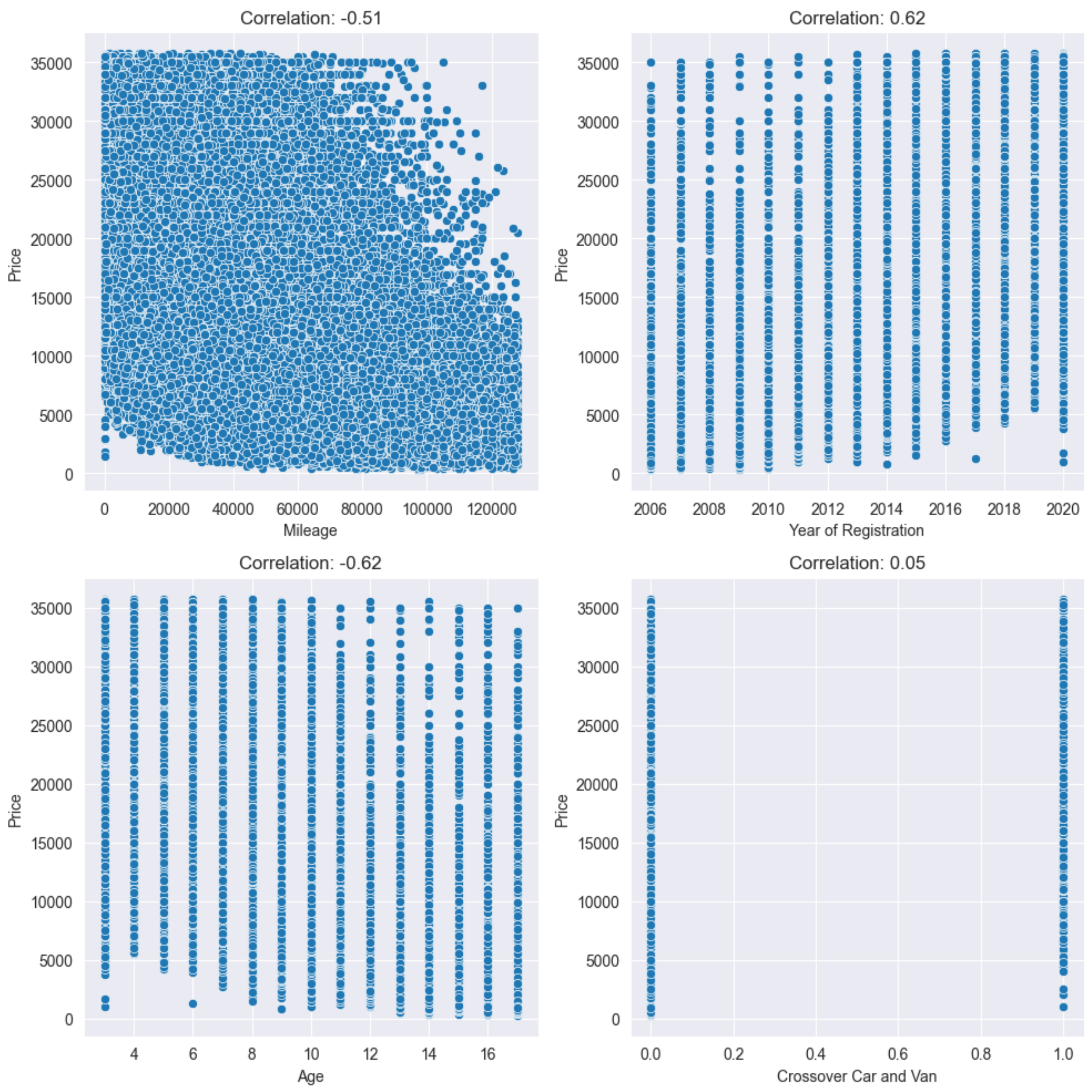
	price	mileage	year_of_registration	age	crossover_car_
price	1.000000	-0.512640	0.617176	-0.617176	
mileage	-0.512640	1.000000	-0.755556	0.755556	
year_of_registration	0.617176	-0.755556	1.000000	-1.000000	
age	-0.617176	0.755556	-1.000000	1.000000	
crossover_car_and_van	0.047616	0.021307	-0.003713	0.003713	

lets visualize the correlation

In [92]:
```
sns.heatmap(adv.corr(), annot=True, cmap="YlGnBu")
plt.show()
```



In [93]:
```
# Create a 2x2 grid of subplots
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(10,10))

# Scatter plot of price vs. mileage
sns.scatterplot(x='mileage', y='price', data=adv_numeric, ax=axs[0,0])
axs[0,0].set_xlabel('Mileage')
axs[0,0].set_ylabel('Price')
corr_coef1 = adv_numeric['mileage'].corr(adv_numeric['price'])
axs[0,0].set_title('Correlation: {:.2f}'.format(corr_coef1))

# Scatter plot of price vs. year_of_registration
sns.scatterplot(x='year_of_registration', y='price', data=adv_numeric, ax=ax
axs[0,1].set_xlabel('Year of Registration')
axs[0,1].set_ylabel('Price')
corr_coef2 = adv_numeric['year_of_registration'].corr(adv_numeric['price'])
axs[0,1].set_title('Correlation: {:.2f}'.format(corr_coef2))

# Scatter plot of price vs. age
sns.scatterplot(x='age', y='price', data=adv_numeric, ax=axs[1,0])
axs[1,0].set_xlabel('Age')
axs[1,0].set_ylabel('Price')
corr_coef3 = adv_numeric['age'].corr(adv_numeric['price'])
axs[1,0].set_title('Correlation: {:.2f}'.format(corr_coef3))

# Scatter plot of price vs. crossover_car_and_van
sns.scatterplot(x='crossover_car_and_van', y='price', data=adv_numeric, ax=a
axs[1,1].set_xlabel('Crossover Car and Van')
axs[1,1].set_ylabel('Price')
corr_coef4 = adv_numeric['crossover_car_and_van'].corr(adv_numeric['price'])
axs[1,1].set_title('Correlation: {:.2f}'.format(corr_coef4))

# Adjust the layout
fig.tight_layout()


# Show the plot
plt.show()
```



- 'price' vs 'year_of_registration': A correlation of 0.62 between price and year_of_registration means that there is a strong positive correlation between the two variables. In other words, as the value of year_of_registration increases, there is a strong tendency for the price to increase. A correlation coefficient of 0.62 is relatively large and suggests that year_of_registration is a very good predictor of price. This makes sense, as newer cars are generally more expensive than older ones due to technological advancements, increased safety features, and general wear and tear over time.

- 'price' vs 'mileage': A correlation of -0.51 between price and mileage means that there is a moderate negative correlation between the two variables. In other words, as the value of mileage increases, there is a tendency for the price to decrease. A correlation coefficient of -0.51 is relatively moderate and suggests that mileage is a fairly good predictor of price, but not as strong as year_of_registration. This is expected since cars with high mileage are usually considered to be more worn and thus less valuable.

- price vs age: A correlation of -0.62 between price and age means that there is a strong negative correlation between the two variables. In other words, as the value of age increases, there is a strong tendency for the price to decrease. A correlation coefficient of -0.62 is relatively large and suggests that age is a very good predictor of price. This is a reasonable result, as older cars are generally less valuable than newer ones due to wear and tear and advances in technology.
- price vs crossover_car_and_van: A correlation of 0.05 between price and crossover_car_and_van means that there is a weak positive correlation between the two variables. In other words, as the value of crossover_car_and_van increases, there is a slight tendency for the price to increase as well, but the relationship is not very strong. A correlation coefficient of 0.05 is relatively low and suggests that crossover_car_and_van is not a very good predictor of price.

Q2: Does mileage impact the value of cars?

In [94]:
```
adv.sort_values('mileage', ascending=False).head(1)
```

Out[94]:

	mileage	standard_colour	standard_make	standard_model	year_of_registration	p
157137	128000.0	Black	Audi	A4 Avant	2011	5

This sorts the DataFrame adv by mileage in descending order and returns the top row with the highest mileage. This can be used to check the maximum mileage in the dataset.

In [95]:
```
# Group the data by 10,000-mile intervals and calculate the mean price for e
mileage_price = adv.groupby(pd.cut(adv['mileage'], bins=range(0, 140000, 100
mileage_price
```

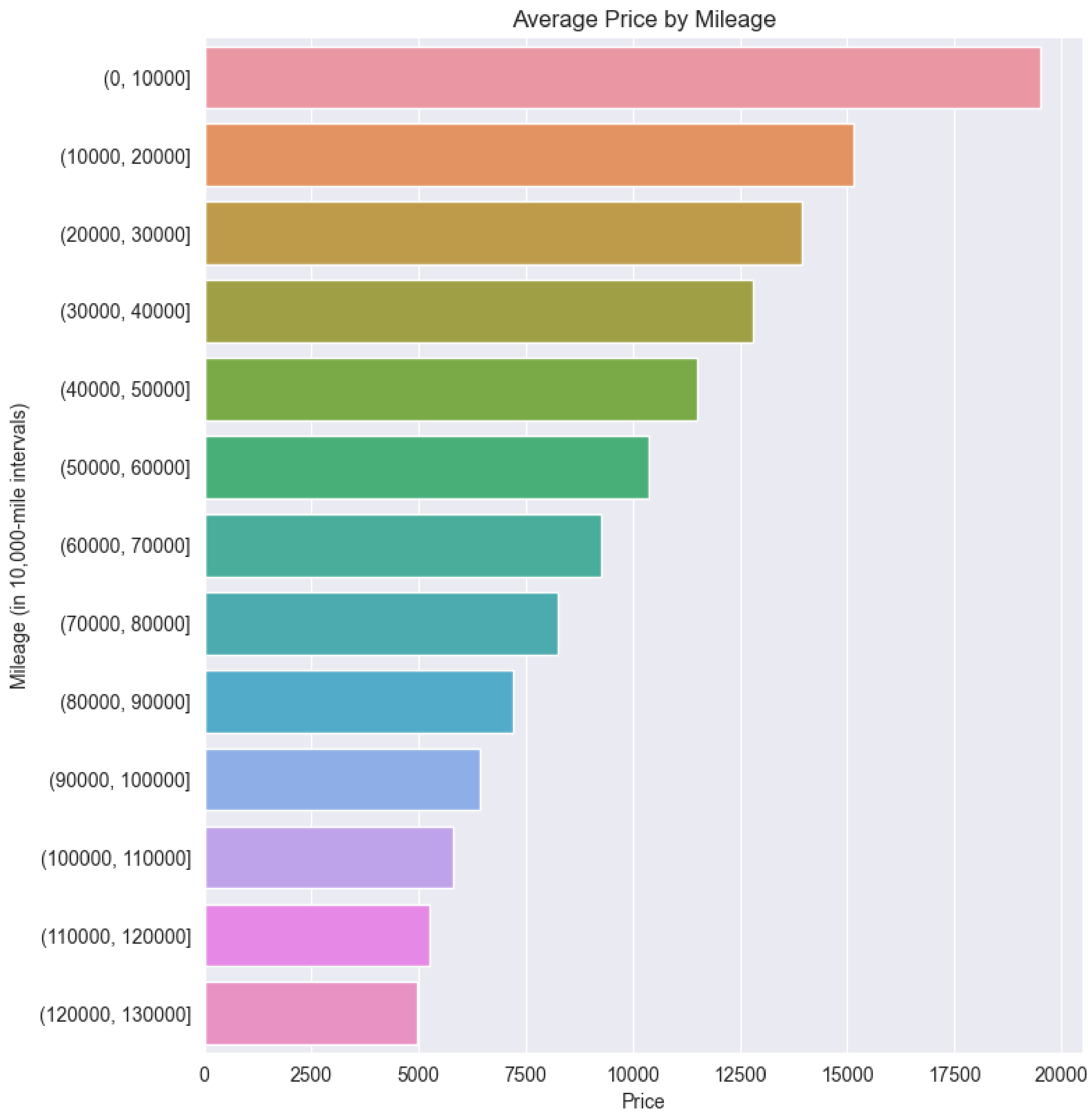
Out[95]:

	mileage	price
0	(0, 10000]	19516.893327
1	(10000, 20000]	15159.739497
2	(20000, 30000]	13945.619778
3	(30000, 40000]	12804.563921
4	(40000, 50000]	11497.359031
5	(50000, 60000]	10361.570825
6	(60000, 70000]	9263.225593
7	(70000, 80000]	8234.735291
8	(80000, 90000]	7207.531152
9	(90000, 100000]	6440.936282
10	(100000, 110000]	5802.727615
11	(110000, 120000]	5254.288563
12	(120000, 130000]	4946.195477

This groups the DataFrame adv by mileage ranges of 10,000 miles using pd.cut and calculates the mean price for each mileage range using groupby. The resulting DataFrame mileage_price can be used to create a catplot to visualize the relationship between mileage and price. By analyzing this plot, we can determine if there is a clear impact of mileage on the value of cars.

In [96]:

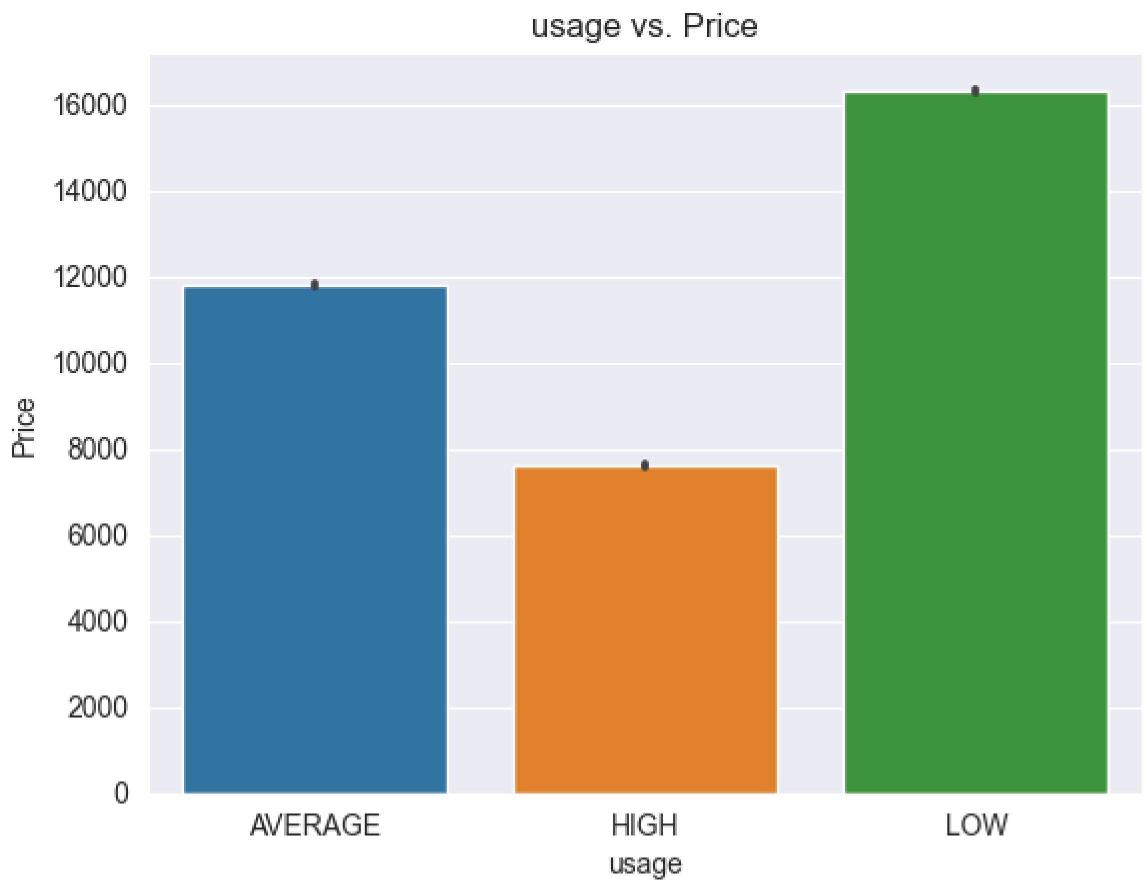
```
sns.catplot(
    data=mileage_price, x='price', y="mileage",
    kind="bar", height=8
)

# Set the plot title and axis labels
plt.title('Average Price by Mileage')
plt.xlabel('Price')
plt.ylabel('Mileage (in 10,000-mile intervals)')

# Show the plot
plt.show()
```



The first row shows that for cars with mileage between 0 and 10,000 miles, the average price is 19,516.89. Similarly, the last row shows that for cars with mileage between 120,000 and 130,000 miles, the average price is 4,946.20.

From this result, we can observe that as the mileage of the cars increases, the average price decreases. This suggests that mileage does indeed impact the value of cars.

In [97]:
```
sns.barplot(x='usage', y='price', data=adv)
plt.xlabel('usage')
plt.ylabel('Price')
plt.title('usage vs. Price')
plt.show()
```



The plot is showing the relationship between car mileage and price. It appears that as mileage increases, the price of the car decreases. This is likely due to the fact that cars with higher mileage have been used more and may have more wear and tear, making them less valuable. Additionally, the plot shows that cars with low mileage and cars that are new have relatively higher prices compared to cars with average and high mileage. This could be because new cars and cars with low mileage are considered more desirable and therefore command a higher price.

Q3: What is the average price of vehicles by body type?

In [98]:
```
adv.body_type.value_counts()
```

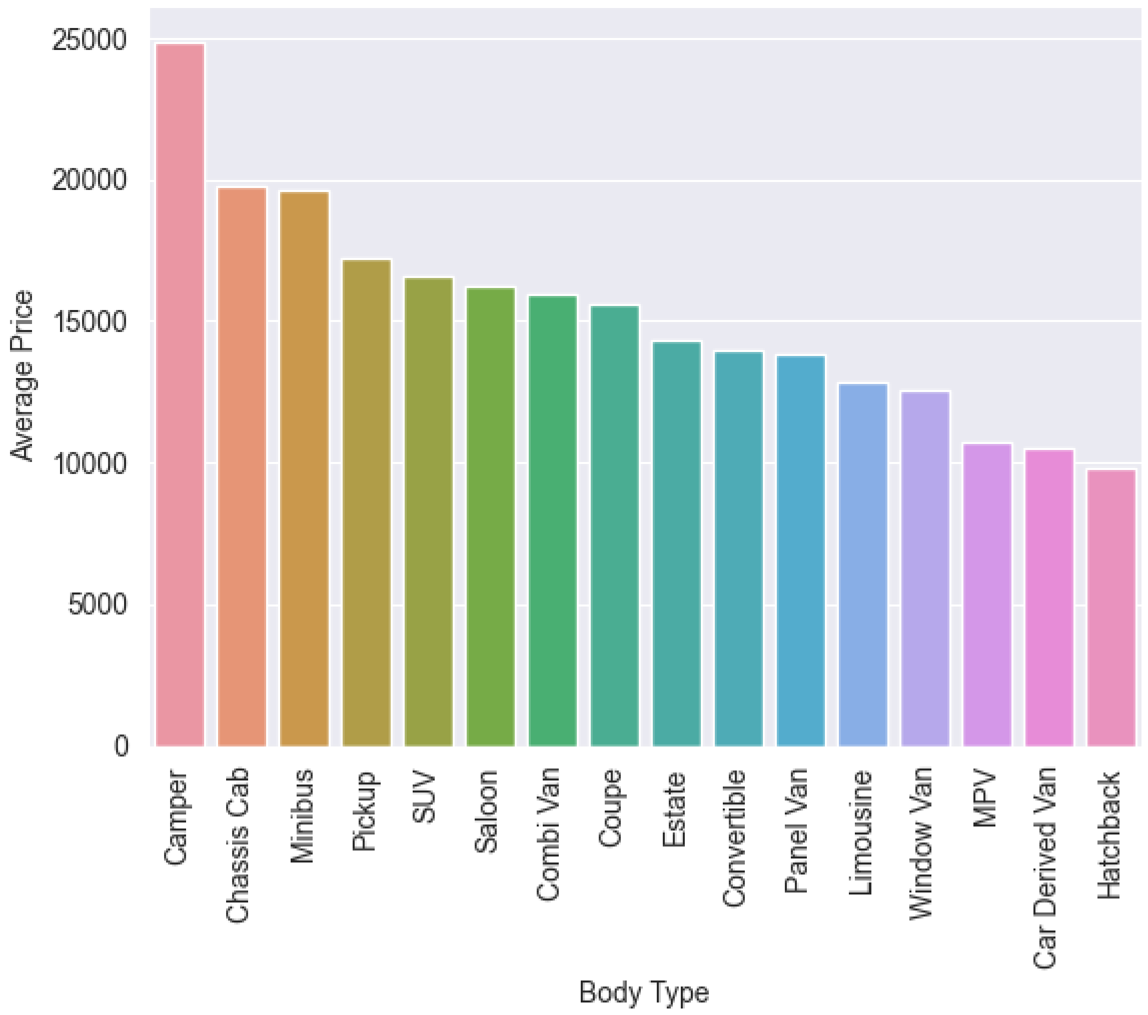
Out[98]:
```
Hatchback          74845
SUV                43694
Saloon             14946
Estate             10485
Coupe               8034
MPV                 7272
Convertible         5463
Pickup               205
Combi Van             69
Minibus               60
Limousine             26
Panel Van             24
Window Van            14
Camper                10
Car Derived Van        1
Chassis Cab            1
Name: body_type, dtype: int64
```

This is a count of the number of cars in the dataset that belong to each body type. Hatchback is the most common body type with 74,845 cars, followed by SUV with 43,694 cars, and Saloon with 14,946 cars. The least common body types are Car Derived Van and Chassis Cab, each with only one car in the dataset.

In [99]:
```
body_price = adv.groupby("body_type")["price"].mean().sort_values(ascending=
body_price
```

Out[99]:
```
body_type
Camper             24889.400000
Chassis Cab        19750.000000
Minibus            19653.133333
Pickup             17243.000000
SUV                16567.861354
Saloon             16247.966078
Combi Van          15945.289855
Coupe              15616.506348
Estate             14292.076490
Convertible        13983.629141
Panel Van          13845.125000
Limousine          12815.423077
Window Van         12587.500000
MPV                10714.655666
Car Derived Van    10495.000000
Hatchback           9819.031024
Name: price, dtype: float64
```

In [100…
```
# Create bar plot
sns.barplot(x=body_price.index, y=body_price.values)

# Add x-axis title
plt.xlabel("Body Type")

# Add y-axis title
plt.ylabel("Average Price")

# Rotate x-axis labels by 45 degrees
plt.xticks(rotation=90)

# Show plot
plt.show()
```



The body type with the highest mean price is Camper with a value of 24889.40, followed by Chassis Cab with a value of 19750.00, and so on. This result gives an idea of which body type has the highest and lowest average price.

From these results, we can infer that the most common body types are Hatchback, SUV, and Saloon. The body types with the highest mean prices are Camper, Chassis Cab, and Minibus, which suggests that these are likely to be more expensive types of vehicles. The body type with the lowest mean price is Hatchback, which suggests that this is likely to be a more affordable type of vehicle.

Overall, these results can provide insights into the market demand for different types of vehicles and can be useful for car manufacturers and dealerships in determining their pricing and marketing strategies.

Q4: What is the average price of vehicles by fuel_type

In [101…
```
adv.fuel_type.value_counts()
```

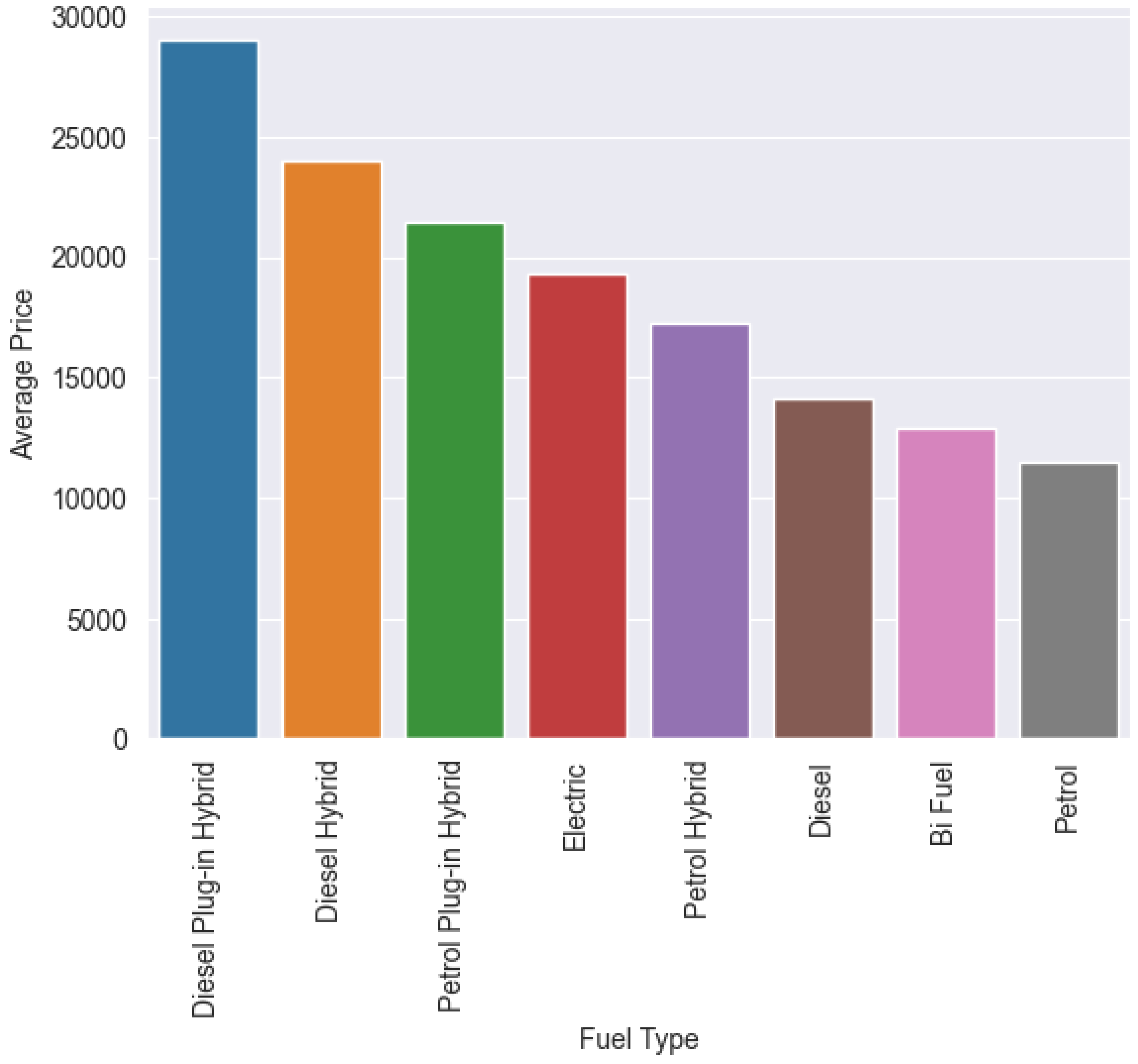
Out[101]:
```
Petrol                    87879
Diesel                    68551
Petrol Hybrid              5466
Petrol Plug-in Hybrid      1725
Electric                   1224
Diesel Hybrid               231
Diesel Plug-in Hybrid        46
Bi Fuel                      27
Name: fuel_type, dtype: int64
```

This result shows the count of cars in a dataset categorized by their fuel type. The most common fuel type is petrol, followed by diesel and then petrol hybrid. There are also smaller numbers of cars with petrol plug-in hybrid, electric, diesel hybrid, diesel plug-in hybrid, and bi-fuel engines.

In [102…
```python
fuel_price = adv.groupby("fuel_type")["price"].mean().sort_values(ascending=
fuel_price
```

Out[102]:
```
fuel_type
Diesel Plug-in Hybrid    28989.391304
Diesel Hybrid            24036.177489
Petrol Plug-in Hybrid    21460.822609
Electric                 19301.254085
Petrol Hybrid            17230.757958
Diesel                   14095.210063
Bi Fuel                  12879.407407
Petrol                   11492.180487
Name: price, dtype: float64
```

In [103…
```python
# Create bar plot
sns.barplot(x=fuel_price.index, y=fuel_price.values)

# Add x-axis title
plt.xlabel("Fuel Type")

# Add y-axis title
plt.ylabel("Average Price")

# Rotate x-axis labels by 45 degrees
plt.xticks(rotation=90)

# Show plot
plt.show()
```



The result shows the mean prices of cars grouped by their fuel type in descending order. We can see that the average price of cars with diesel plug-in hybrid fuel type is the highest followed by diesel hybrid and petrol plug-in hybrid. The average prices of electric and petrol hybrid cars are also relatively high. On the other hand, the average prices of cars with petrol and bi-fuel are the lowest.

From this result, we can infer that people who are willing to spend more on cars tend to choose plug-in hybrids, diesel hybrids, and electric cars. This might be because these types of cars are more environmentally friendly and have lower fuel costs in the long run. On the other hand, people who prioritize affordability and convenience may opt for petrol and bi-fuel cars. However, it's important to note that the choice of fuel type may also be influenced by factors such as availability, government regulations, and personal preferences.

Q5: Does year of registration affect the average price

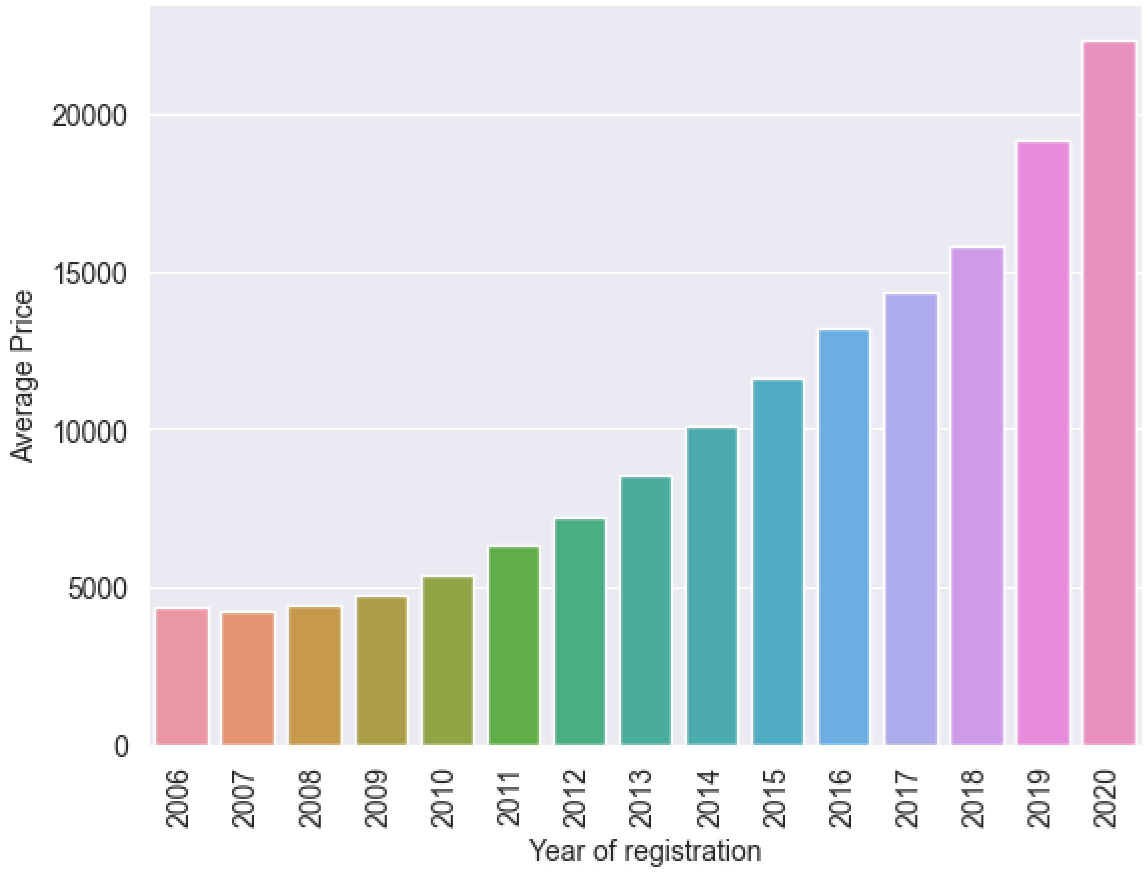
In [104…

```
year_price = adv.groupby("year_of_registration")["price"].mean().sort_values
year_price
```

Out[104]:
```
year_of_registration
2020    22367.451305
2019    19195.083441
2018    15781.463274
2017    14366.470604
2016    13184.443140
2015    11624.682380
2014    10105.651786
2013     8550.383130
2012     7253.587491
2011     6338.101635
2010     5381.743718
2009     4716.535151
2008     4406.216107
2006     4335.439475
2007     4264.547805
Name: price, dtype: float64
```

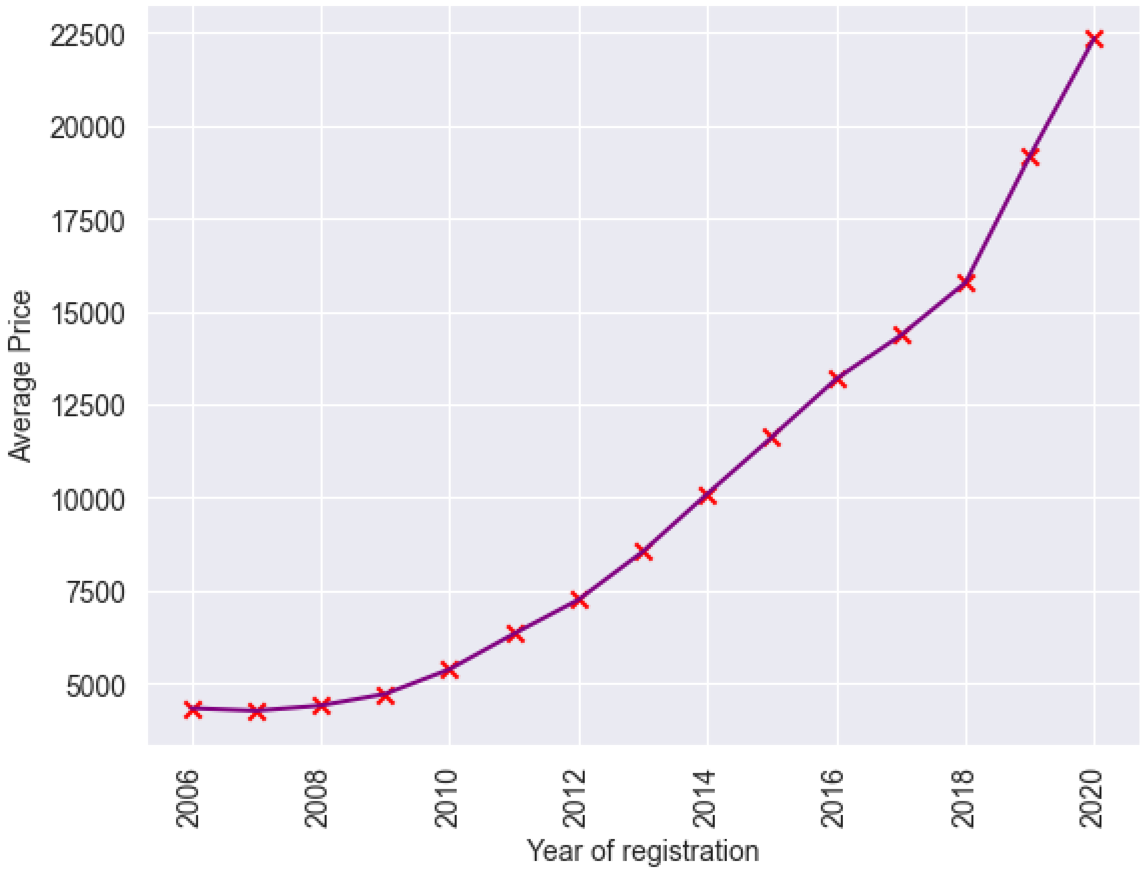
This shows that the cars that were registered in the year 2020 had the highest average price of 22367.451305. The average price of cars decreases as we move towards earlier years of registration. The cars that were registered in the year 2007 had the lowest average price of 4264.547805.

In [105…
```
# Create bar plot
sns.barplot(x=year_price.index, y=year_price.values)

# Add x-axis title
plt.xlabel("Year of registration")

# Add y-axis title
plt.ylabel("Average Price")

# Rotate x-axis labels by 90 degrees
plt.xticks(rotation=90)

# Show plot
plt.show()
```



In [106…

```
# Create line plot
sns.lineplot(x=year_price.index, y=year_price.values, c='purple')

# Add x-axis title
plt.xlabel("Year of registration")

# Add y-axis title
plt.ylabel("Average Price")

# Rotate x-axis labels by 90 degrees
plt.xticks(rotation=90)

# Add individual data points
plt.scatter(x=year_price.index, y=year_price.values, c="red", marker = 'x')

# Show plot
plt.show()
```



The result shows the mean price of cars based on the year of registration. From the result, we can see that the average price of cars tends to decrease as the year of registration goes further back in time. This is likely due to several factors such as the wear and tear of the vehicle, the availability of newer and more advanced models, and changes in the market demand.

We can infer that people are willing to pay more for newer cars, and that year of registration is an important factor to consider when determining the value of a car. Additionally, this information can be useful for car dealerships and individuals looking to sell their car, as they can use it to determine a fair asking price based on the year of registration.

From this analysis, we can infer that the year of registration does affect the average price of cars. Generally, newer cars tend to have a higher price compared to older cars. However, there may be exceptions to this trend, such as classic or vintage cars that may have a higher price due to their rarity or unique features.

Conclusions

Results: Our data suggest that

Correlation Summary

'price' vs 'year_of_registration'

- There is a strong positive correlation between price and year_of_registration.
- As the value of year_of_registration increases, there is a strong tendency for the price to increase.
- A correlation coefficient of 0.62 is relatively large and suggests that year_of_registration is a very good predictor of price.

'price' vs 'mileage'

- There is a moderate negative correlation between price and mileage.
- As the value of mileage increases, there is a tendency for the price to decrease.
- A correlation coefficient of -0.51 is relatively moderate and suggests that mileage is a fairly good predictor of price, but not as strong as year_of_registration.

'price' vs 'age'

- There is a strong negative correlation between price and age.
- As the value of age increases, there is a strong tendency for the price to decrease.
- A correlation coefficient of -0.62 is relatively large and suggests that age is a very good predictor of price.

'price' vs 'crossover_car_and_van'

- There is a weak positive correlation between price and crossover_car_and_van.
- As the value of crossover_car_and_van increases, there is a slight tendency for the price to increase as well, but the relationship is not very strong.
- A correlation coefficient of 0.05 is relatively low and suggests that crossover_car_and_van is not a very good predictor of price.

Mileage relationship with price

- It appears that as mileage increases, the price of the car decreases. This is likely due to the fact that cars with higher mileage have been used more and may have more wear and tear, making them less valuable. Additionally, the plot shows that cars with low mileage and cars that are new have relatively higher prices compared to cars with average and high mileage. This could be because new cars and cars with low mileage are considered more desirable and therefore command a higher price.

Body type relationship with price

- From these results, we can infer that the most common body types are Hatchback, SUV, and Saloon. The body types with the highest mean prices are Camper, Chassis Cab, and Minibus, which suggests that these are likely to be more expensive types of vehicles. The body type with the lowest mean price is Hatchback, which suggests that this is likely to be a more affordable type of vehicle.

- Overall, these results can provide insights into the market demand for different types of vehicles and can be useful for car manufacturers and dealerships in determining their pricing and marketing strategies.

Fuel type relationship with price

- We can see that the average price of cars with diesel plug-in hybrid fuel type is the highest followed by diesel hybrid and petrol plug-in hybrid. The average prices of electric and petrol hybrid cars are also relatively high. On the other hand, the average prices of cars with petrol and bi-fuel are the lowest.
- we can infer that people who are willing to spend more on cars tend to choose plug-in hybrids, diesel hybrids, and electric cars. This might be because these types of cars are more environmentally friendly and have lower fuel costs in the long run. On the other hand, people who prioritize affordability and convenience may opt for petrol and bi-fuel cars. However, it's important to note that the choice of fuel type may also be influenced by factors such as availability, government regulations, and personal preferences.

Year of registration relationship with price

- As the year of registration goes further back in time, the average price of cars decreases. This can be attributed to factors such as wear and tear, newer and more advanced models, and changes in market demand. This information can help in determining the value of a car and can be useful for individuals looking to sell their car or car dealerships to set a fair price based on the year of registration.
- From this analysis, we can infer that the year of registration does affect the average price of cars. Generally, newer cars tend to have a higher price compared to older cars. However, there may be exceptions to this trend, such as classic or vintage cars that may have a higher price due to their rarity or unique features.

limitations: Some limitations apply to our data:

1. Quality of data: The accuracy and completeness of the data can be a limitation. If there are missing values or errors in the data, it can affect the quality of the analysis and the conclusions drawn from it.
2. Representativeness of the sample: The dataset might not be a representative sample of the entire population of cars. For example, the dataset might overrepresent certain brands or models of cars, making it difficult to generalize the results to the entire population.

3. Time period: The data might not be up-to-date, or it might not cover a long enough time period to capture changes in the market over time.

4. Data bias: There might be biases in the data collection process that could impact the analysis. For example, if the data was collected from a specific region, it might not be applicable to other regions.

5. Confounding variables: There might be other variables that are not included in the dataset that could affect the relationship between car features and prices. For example, the dataset might not include variables such as fuel prices or economic indicators that could impact car prices.

It's important to consider these limitations when analyzing the dataset and drawing conclusions from it.

References:

> - Seaborn documentation
> - w3resource on pandas
> - w3schools python functions