To predict Audi car price analysis

Methodology Use :

1. Support Vector Machine (SVM) - Regression problem(SVR)
2. Random Forest Regressor - Regression problem
3. Multiple Linear Regression model

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn as sk

from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
```

Loading the Data

In [2]:
```
car_audi = pd.read_csv("audi.csv")
```

In [3]:
```
car_audi.shape
```

Out[3]:
```
(10668, 9)
```

Step 1 - Initial Exploratory Data Analysis & Data Preparation - Exploring Data

In [4]:
```
car_audi.columns
```

Out[4]:
```
Index(['model', 'year', 'price', 'transmission', 'mileage', 'fuelType', 'tax',
       'mpg', 'engineSize'],
      dtype='object')
```

In [5]:
```
car_audi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10668 entries, 0 to 10667
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   model         10668 non-null  object 
 1   year          10668 non-null  int64  
 2   price         10668 non-null  int64  
 3   transmission  10668 non-null  object 
 4   mileage       10668 non-null  int64  
 5   fuelType      10668 non-null  object 
 6   tax           10668 non-null  int64  
 7   mpg           10668 non-null  float64
 8   engineSize    10668 non-null  float64
dtypes: float64(2), int64(4), object(3)
memory usage: 750.2+ KB
```

In [6]:
```
car_audi.head()
```

Out[6]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	A1	2017	12500	Manual	15735	Petrol	150	55.4	1.4
1	A6	2016	16500	Automatic	36203	Diesel	20	64.2	2.0
2	A1	2016	11000	Manual	29946	Petrol	30	55.4	1.4
3	A4	2017	16800	Automatic	25952	Diesel	145	67.3	2.0
4	A3	2019	17300	Manual	1998	Petrol	145	49.6	1.0

In [7]:
```
car_audi.tail()
```

Out[7]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
10663	A3	2020	16999	Manual	4018	Petrol	145	49.6	1.0
10664	A3	2020	16999	Manual	1978	Petrol	150	49.6	1.0
10665	A3	2020	17199	Manual	609	Petrol	150	49.6	1.0
10666	Q3	2017	19499	Automatic	8646	Petrol	150	47.9	1.4
10667	Q3	2016	15999	Manual	11855	Petrol	150	47.9	1.4

In [8]:
```
car_audi['transmission'].value_counts()
```

Out[8]:
```
Manual       4369
Semi-Auto    3591
Automatic    2708
Name: transmission, dtype: int64
```

In [9]:
```
car_audi['fuelType'].value_counts()
```

Out[9]:
```
Diesel    5577
Petrol    5063
Hybrid      28
Name: fuelType, dtype: int64
```

In [10]:
```
car_audi['engineSize'].value_counts()
```

Out[10]:
```
2.0    5169
1.4    1594
3.0    1149
1.6     913
1.5     744
1.0     558
4.0     154
1.8     126
2.5      61
0.0      57
2.9      49
1.2      31
4.2      25
5.2      23
3.2       5
1.9       4
2.7       3
```

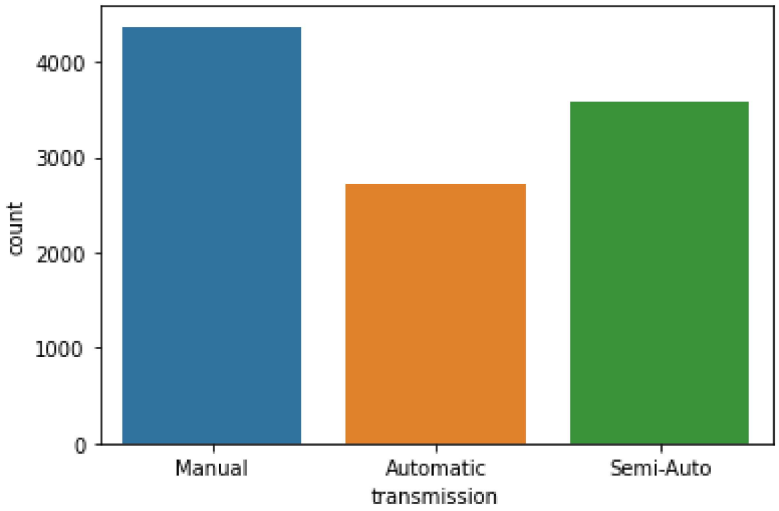
```
4.1       2
6.3       1
Name: engineSize, dtype: int64
```

In [11]:
```
car_audi['model'].value_counts()
```

Out[11]:
```
A3     1929
Q3     1417
A4     1381
A1     1347
A5      882
Q5      877
Q2      822
A6      748
Q7      397
TT      336
A7      122
A8      118
Q8       69
RS6      39
RS3      33
RS4      31
RS5      29
R8       28
S3       18
SQ5      16
S4       12
SQ7       8
S8        4
S5        3
RS7       1
A2        1
Name: model, dtype: int64
```

In [12]:
```
sns.countplot(x=car_audi["transmission"])
```

Out[12]:
```
<AxesSubplot:xlabel='transmission', ylabel='count'>
```



Above shows most of the cars on the dataset are with manual transmission, follow by semi-auto and least cars in automatic transmission

Let's check the price distribution among the models represented.

Hypothesis : price is very correlated with the model.

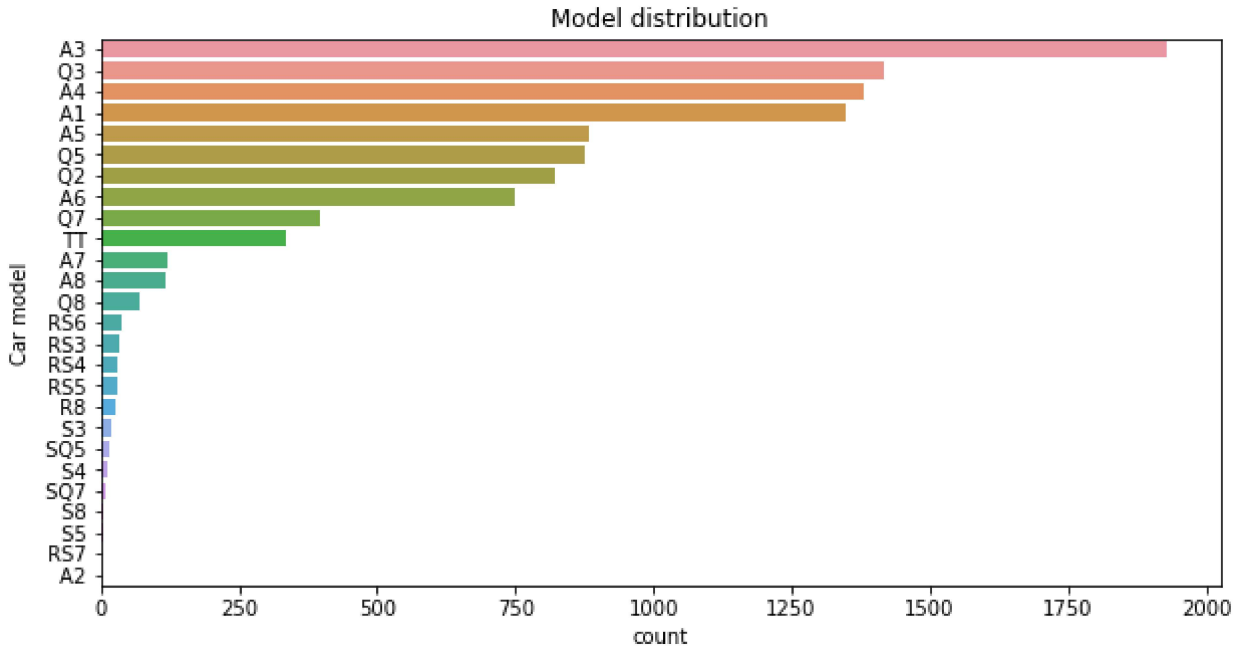
In [13]:
```
print(car_audi["model"].value_counts() / len(car_audi))
```

```
fig, ax = plt.subplots(figsize = (10,5))
sns.countplot(y = 'model', data = car_audi, order = car_audi['model'].value_counts().in
plt.ylabel('Car model')
plt.title('Model distribution')

#sns.countplot(y = car_audi["model"])
```

```
A3     0.180821
Q3     0.132827
A4     0.129453
A1     0.126265
A5     0.082677
Q5     0.082208
Q2     0.077053
A6     0.070116
Q7     0.037214
TT     0.031496
A7     0.011436
A8     0.011061
Q8     0.006468
RS6    0.003656
RS3    0.003093
RS4    0.002906
RS5    0.002718
R8     0.002625
S3     0.001687
SQ5    0.001500
S4     0.001125
SQ7    0.000750
S8     0.000375
S5     0.000281
RS7    0.000094
A2     0.000094
Name: model, dtype: float64
```

Out[13]: Text(0.5, 1.0, 'Model distribution')



Above dataset shows Top 3 cars are 'A3', 'Q3' and 'A4'/'A1' that contributed 57% of all the AUDI cars, with all other cars contributing to 43%

'A3' 0.180821

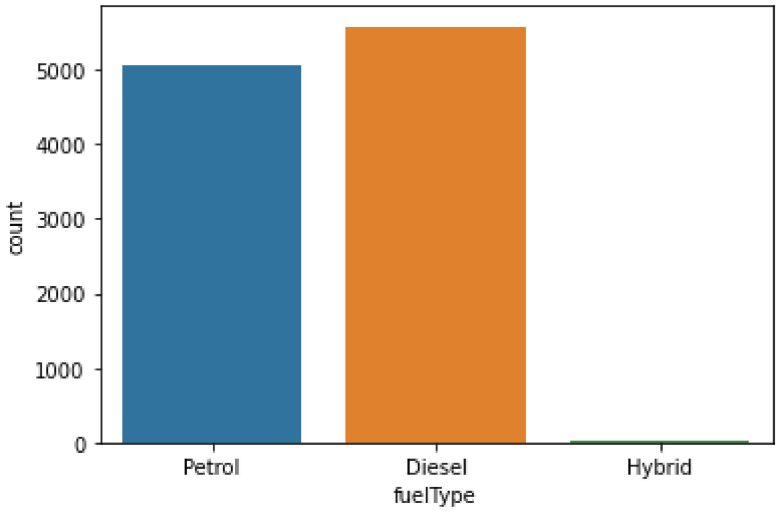
'Q3' 0.132827

'A4' 0.129453

'A1' 0.126265

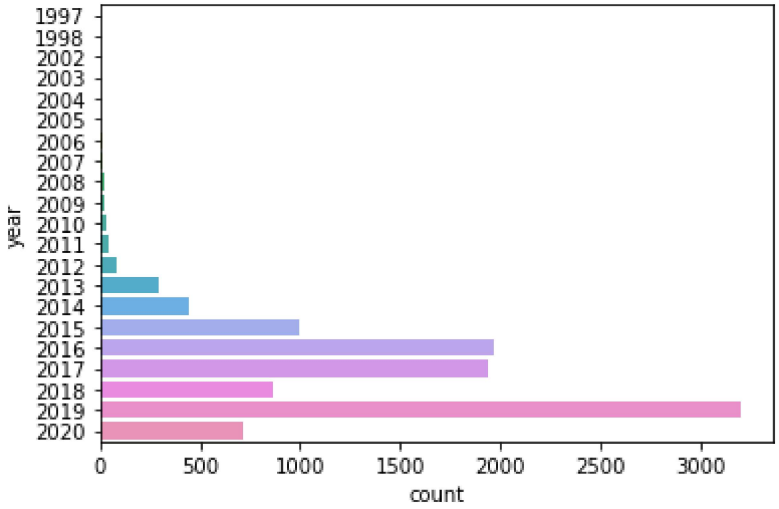
In [14]:
```
sns.countplot(x=car_audi["fuelType"])
```

Out[14]: <AxesSubplot:xlabel='fuelType', ylabel='count'>



In [15]:
```
sns.countplot(y = car_audi["year"])
```

Out[15]: <AxesSubplot:xlabel='count', ylabel='year'>



Step 2: EDA & Data Preprocessing

Cleaning Data (dataset Quality)

In [16]:
```
car_audi.describe()
```

Out[16]:

	year	price	mileage	tax	mpg	engineSize
count	10668.000000	10668.000000	10668.000000	10668.000000	10668.000000	10668.000000
mean	2017.100675	22896.685039	24827.244001	126.011436	50.770022	1.930709

	year	price	mileage	tax	mpg	engineSize
std	2.167494	11714.841888	23505.257205	67.170294	12.949782	0.602957
min	1997.000000	1490.000000	1.000000	0.000000	18.900000	0.000000
25%	2016.000000	15130.750000	5968.750000	125.000000	40.900000	1.500000
50%	2017.000000	20200.000000	19000.000000	145.000000	49.600000	2.000000
75%	2019.000000	27990.000000	36464.500000	145.000000	58.900000	2.000000
max	2020.000000	145000.000000	323000.000000	580.000000	188.300000	6.300000

In [17]:
```
#To show only feature that have non-zero missing values
# df_na[df_na!=0]

car_audi_na = car_audi.isna().sum()
car_audi_na
```

Out[17]:
```
model           0
year            0
price           0
transmission    0
mileage         0
fuelType        0
tax             0
mpg             0
engineSize      0
dtype: int64
```

In [18]:
```
# limit to categorical data using df.select_dtypes()
# count number of items on each unique categories feature

car_audi_cat = car_audi.select_dtypes(include=['object'])  #data label
car_audi_cat.nunique()
```

Out[18]:
```
model           26
transmission     3
fuelType         3
dtype: int64
```

In [19]:
```
# limit to numerical data using df.select_dtypes()
# count number of items on each unique numberical feature

car_audi_num = car_audi.select_dtypes(include = ['number'])
car_audi_num.nunique()
```

Out[19]:
```
year            21
price         3260
mileage       7725
tax             37
mpg            104
engineSize      19
dtype: int64
```

In [20]:
```
car_audi['model'].unique()
```

Out[20]:
```
array([' A1', ' A6', ' A4', ' A3', ' Q3', ' Q5', ' A5', ' S4', ' Q2',
       ' A7', ' TT', ' Q7', ' RS6', ' RS3', ' A8', ' Q8', ' RS4', ' RS5',
       ' R8', ' SQ5', ' S8', ' SQ7', ' S3', ' S5', ' A2', ' RS7'],
      dtype=object)
```

In [21]:
```
car_audi['year'].unique()
```

Out[21]:
```
array([2017, 2016, 2019, 2015, 2014, 2018, 2013, 2020, 2004, 2009, 2012,
       2010, 2007, 2011, 2008, 2003, 2005, 2002, 2006, 1998, 1997],
      dtype=int64)
```

In [22]:
```
car_audi['price'].unique()
```

Out[22]:
```
array([12500, 16500, 11000, ..., 21291, 12380,  3750], dtype=int64)
```

In [23]:
```
car_audi['mileage'].unique()
```

Out[23]:
```
array([15735, 36203, 29946, ...,  4018,  1978,  8646], dtype=int64)
```

In [24]:
```
car_audi['fuelType'].unique()
```

Out[24]:
```
array(['Petrol', 'Diesel', 'Hybrid'], dtype=object)
```

In [25]:
```
car_audi['fuelType'].isna()
```

Out[25]:
```
0        False
1        False
2        False
3        False
4        False
         ...
10663    False
10664    False
10665    False
10666    False
10667    False
Name: fuelType, Length: 10668, dtype: bool
```

In [26]:
```
# look at correlations in the numerical independent variables, as well as the dependent
car_audi_num.corr()
```

Out[26]:

	year	price	mileage	tax	mpg	engineSize
year	1.000000	0.592581	-0.789667	0.093066	-0.351281	-0.031582
price	0.592581	1.000000	-0.535357	0.356157	-0.600334	0.591262
mileage	-0.789667	-0.535357	1.000000	-0.166547	0.395103	0.070710
tax	0.093066	0.356157	-0.166547	1.000000	-0.635909	0.393075
mpg	-0.351281	-0.600334	0.395103	-0.635909	1.000000	-0.365621
engineSize	-0.031582	0.591262	0.070710	0.393075	-0.365621	1.000000

In [27]:
```
car_audi.groupby(by='price').count().sort_values('price', ascending=True).head(10)
```

Out[27]:

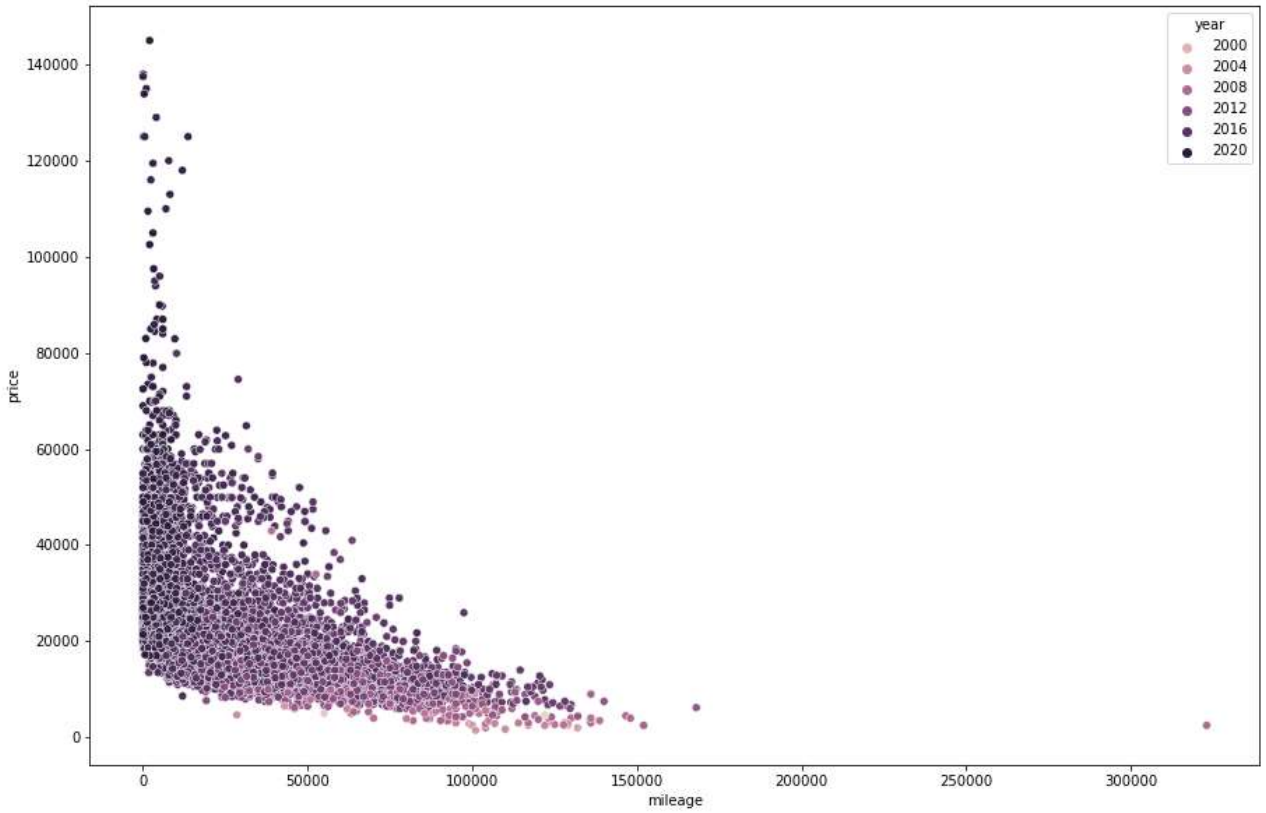
price	model	year	transmission	mileage	fuelType	tax	mpg	engineSize
1490	1	1	1	1	1	1	1	1
1699	1	1	1	1	1	1	1	1

price	model	year	transmission	mileage	fuelType	tax	mpg	engineSize
1975	1	1	1	1	1	1	1	1
1990	1	1	1	1	1	1	1	1
2490	3	3	3	3	3	3	3	3
2495	3	3	3	3	3	3	3	3
2600	1	1	1	1	1	1	1	1
2675	1	1	1	1	1	1	1	1
2795	1	1	1	1	1	1	1	1
2876	1	1	1	1	1	1	1	1

Visualizing Data

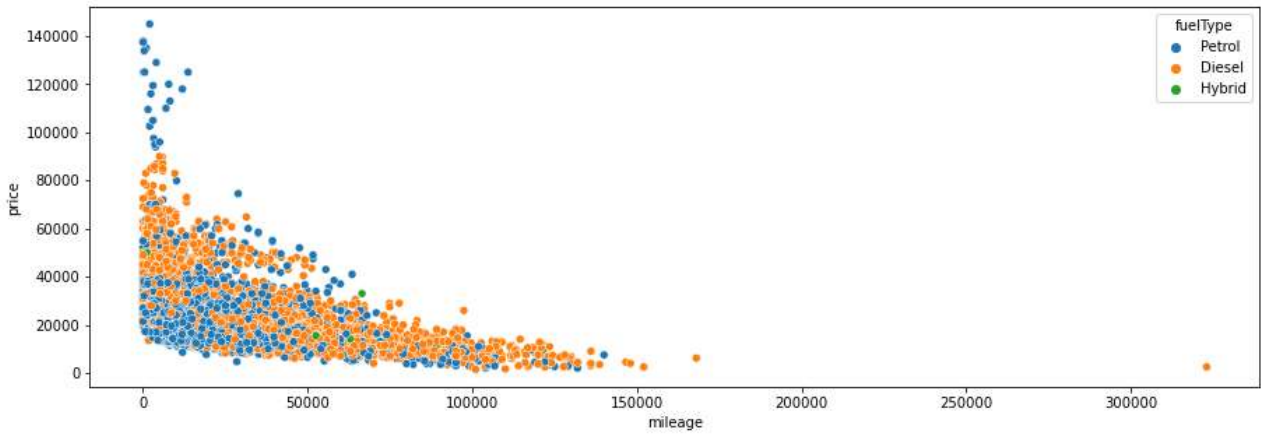
In [28]:
```
plt.figure(figsize=(15,10),facecolor='w')
sns.scatterplot(x=car_audi["mileage"], y=car_audi["price"], hue = car_audi["year"])
```

Out[28]: <AxesSubplot:xlabel='mileage', ylabel='price'>



In [29]:
```
plt.figure(figsize=(15,5),facecolor='w')
sns.scatterplot(x=car_audi["mileage"], y=car_audi["price"], hue = car_audi["fuelType"])
```

Out[29]: <AxesSubplot:xlabel='mileage', ylabel='price'>



In [30]:
```
# correlation matrix heatmap visualization
sns.set(style="white")

# Generate a mask for the upper triangle
matrix = np.triu(car_audi.corr())


# Plot the heatmap
_ = sns.heatmap(car_audi.corr(), mask=matrix, annot=True, annot_kws={"size": 12}, squar
                cmap='coolwarm' , vmin=-1, vmax=1, fmt='.2f')  # annot=True display cor
```

To determining important features that have strong relationship with the target by identifying high correlation values (both positives and negatives)

In [31]:
```
car_audi['price'].nlargest()
```

Out[31]:
```
4783    145000
2255    137995
4179    137500
3367    135000
5459    133900
Name: price, dtype: int64
```

In [32]:
```
car_audi['price'].nsmallest()
```

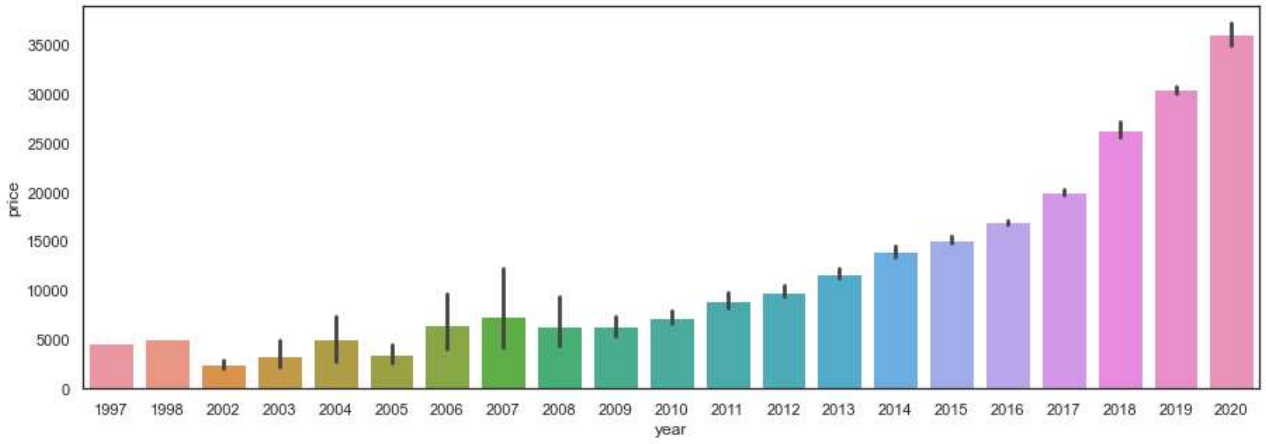
Out[32]:
```
10588    1490
```

```
10552    1699
7795     1975
10108    1990
7404     2490
Name: price, dtype: int64
```

View the Average Price below

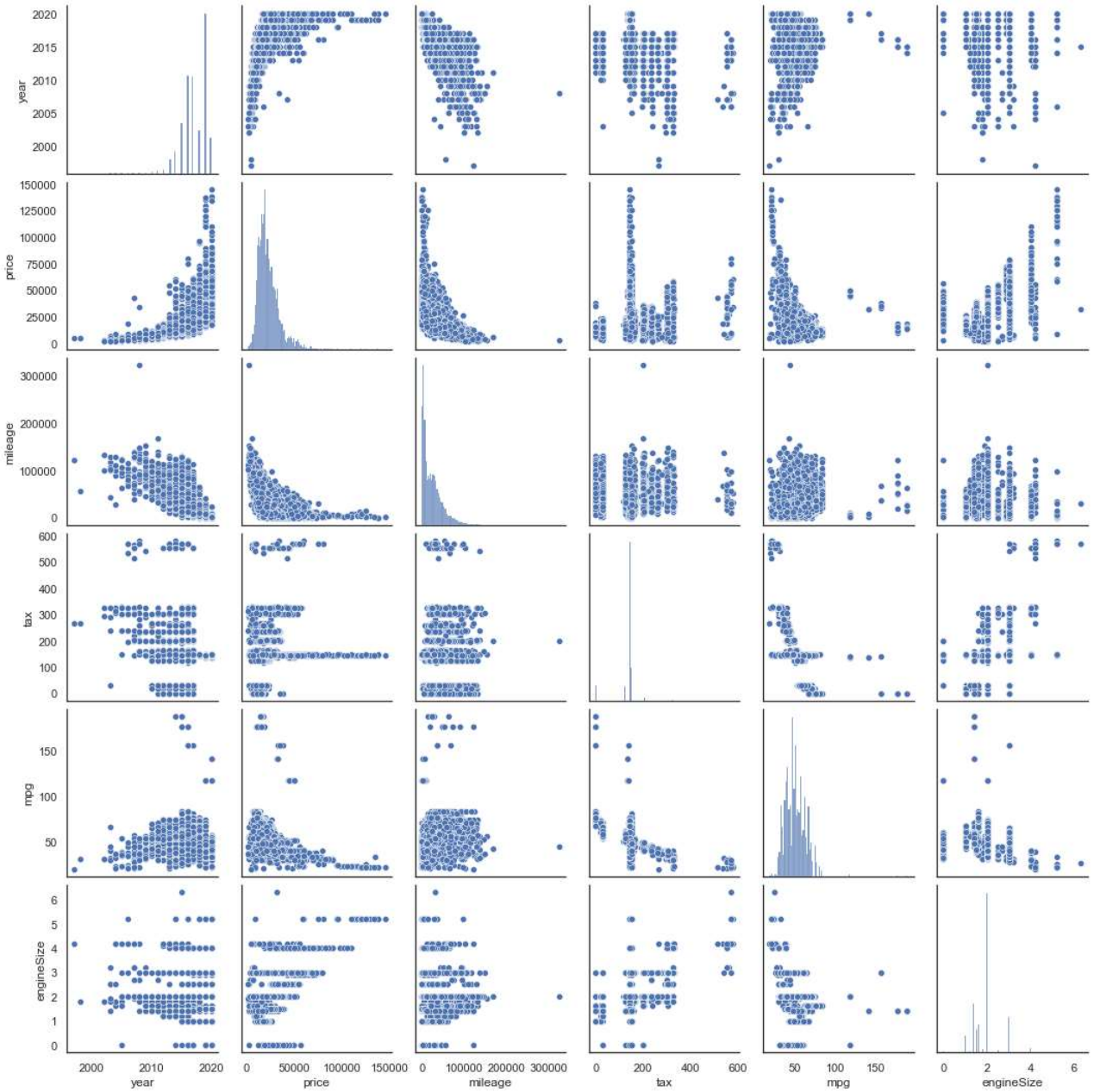
In [33]:
```
plt.figure(figsize=(15,5),facecolor='w')
sns.barplot(x = car_audi["year"], y = car_audi["price"])
```

Out[33]:
```
<AxesSubplot:xlabel='year', ylabel='price'>
```



In [34]:
```
sns.pairplot(car_audi)
```

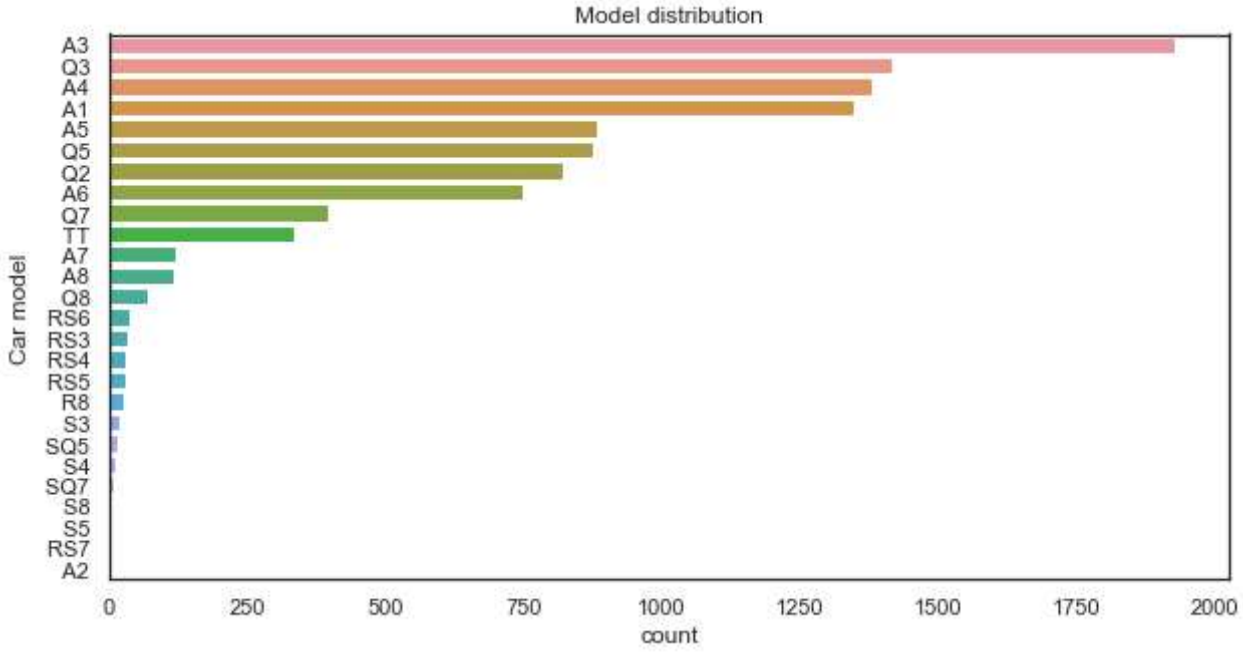
Out[34]:
```
<seaborn.axisgrid.PairGrid at 0x26be4bc5ee0>
```



Let's check the price distribution among the models represented. Hypothesis: price is very correlated with the model

In [35]:
```
fig, ax = plt.subplots(figsize = (10,5))
sns.countplot(y = 'model', data = car_audi, order = car_audi['model'].value_counts().in
plt.ylabel('Car model')
plt.title('Model distribution')
```

Out[35]:
```
Text(0.5, 1.0, 'Model distribution')
```



In [36]:
```
car_audi.describe()
```

Out[36]:

	year	price	mileage	tax	mpg	engineSize
count	10668.000000	10668.000000	10668.000000	10668.000000	10668.000000	10668.000000
mean	2017.100675	22896.685039	24827.244001	126.011436	50.770022	1.930709
std	2.167494	11714.841888	23505.257205	67.170294	12.949782	0.602957
min	1997.000000	1490.000000	1.000000	0.000000	18.900000	0.000000
25%	2016.000000	15130.750000	5968.750000	125.000000	40.900000	1.500000
50%	2017.000000	20200.000000	19000.000000	145.000000	49.600000	2.000000
75%	2019.000000	27990.000000	36464.500000	145.000000	58.900000	2.000000
max	2020.000000	145000.000000	323000.000000	580.000000	188.300000	6.300000

In [37]:
```
car_audi[car_audi['price']<=1490]    #view the min
```

Out[37]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
10588	A3	2003	1490	Automatic	101000	Diesel	240	44.5	1.9

In [38]:
```
car_audi[car_audi['price']>=145000]  #view max = maybe the outliers
```

Out[38]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
4783	R8	2020	145000	Semi-Auto	2000	Petrol	145	21.1	5.2

Save the cleaned dataset to a new file

In [39]:
```
backup=car_audi.copy()    # backup dataset
```

In [40]:
```
car_audi.to_csv('Cleaned car_audi.') # file for building website using Python
```

Step 3: Preparing data for training and testing

In [41]:
```
car_audi
```

Out[41]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	A1	2017	12500	Manual	15735	Petrol	150	55.4	1.4
1	A6	2016	16500	Automatic	36203	Diesel	20	64.2	2.0
2	A1	2016	11000	Manual	29946	Petrol	30	55.4	1.4
3	A4	2017	16800	Automatic	25952	Diesel	145	67.3	2.0
4	A3	2019	17300	Manual	1998	Petrol	145	49.6	1.0
...	...	...	...	...	...	...	...	...	...
10663	A3	2020	16999	Manual	4018	Petrol	145	49.6	1.0
10664	A3	2020	16999	Manual	1978	Petrol	150	49.6	1.0
10665	A3	2020	17199	Manual	609	Petrol	150	49.6	1.0
10666	Q3	2017	19499	Automatic	8646	Petrol	150	47.9	1.4
10667	Q3	2016	15999	Manual	11855	Petrol	150	47.9	1.4

10668 rows × 9 columns

car_audi = car_audi.drop(columns = 'mpg') # to drop the float column ONLY IF NECESSARY

car_audi = car_audi.drop(columns = 'tax') # to drop the tax column ONLY IF NECESSARY

In [42]:
```
car_audi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10668 entries, 0 to 10667
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   model         10668 non-null  object 
 1   year          10668 non-null  int64  
 2   price         10668 non-null  int64  
 3   transmission  10668 non-null  object 
 4   mileage       10668 non-null  int64  
 5   fuelType      10668 non-null  object 
 6   tax           10668 non-null  int64  
 7   mpg           10668 non-null  float64
 8   engineSize    10668 non-null  float64
dtypes: float64(2), int64(4), object(3)
memory usage: 750.2+ KB
```

Step 4 - Creating Machine Learning Training Model

Method 1 : Support Vector Machine (SVM) - Regression problem (SVR)

Split the data into training and testing dataset

In [43]:
```
feature_cols=['year','mileage', 'tax', 'mpg', 'engineSize']
```

In [44]:
```
X=car_audi[feature_cols]
y=car_audi.price
```

In [45]:
```
# Splitting data into training and testing sets

from sklearn.model_selection import train_test_split
X_train, X_test,y_train, y_test=train_test_split(X,y, test_size=0.2) #20% test size
```

In [46]:
```
from sklearn import svm
sv=svm.SVR()
sv.fit(X_train,y_train)
```

Out[46]:
```
SVR()
```

In [47]:
```
pred=sv.predict(X_test)    # y_pred
pred
```

Out[47]:
```
array([19170.81725806, 18631.19292991, 21538.90182514, ...,
       21230.73127666, 18932.49365421, 21490.97289078])
```

In [48]:
```
from sklearn.metrics import accuracy_score
```

In [49]:
```
s=sv.score(X,y)    #Return the coefficient of determination R^2 of the prediction.
s
```

Out[49]:
```
0.04291439313592116
```

In [50]:
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# The mean squared error & mean absolute error
print('Mean squared error: {:.2f}'.format(mean_squared_error(y_test, pred)))    #y_pred
print('Mean absolute error: {:.2f}'.format(mean_absolute_error(y_test, pred)))  #y_pred


# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: {:.2f}'.format(r2_score(y_test, pred)))   #y_pred
```

```
Mean squared error: 132034150.03
Mean absolute error: 7578.15
Coefficient of determination: 0.04
```

Method 2 : Random Forest Regressor - Regression problem

In [51]:
```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
```

In [52]:
```
regressor.fit(X_train,y_train)
```

Out[52]:
```
RandomForestRegressor(random_state=0)
```

In [53]:
```
p=regressor.predict(X_test)
```

In [54]:
```
p
```

Out[54]:
```
array([13337.96      , 15807.72      , 27936.22333333, ...,
       19016.83      , 14631.94      , 23377.89      ])
```

In [55]:
```
s=regressor.score(X,y)  #Return the coefficient of determination R^2 of the prediction.
```

In [56]:
```
print('accuracy score',s)
```

```
accuracy score 0.9780029579219299
```

Evaluation on Random Forest Regressor model

In [57]:
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

In [58]:
```
r2_score(y_test, pred)
```

Out[58]:
```
0.035326809338045884
```

R 2 can take values from 0 to 1. A value of 1 indicates that the regression predictions perfectly fit the data.

In [59]:
```
# from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# The mean squared error & mean absolute error
print('Mean squared error: {:.2f}'.format(mean_squared_error(y_test, pred)))    #y_pred
print('Mean absolute error: {:.2f}'.format(mean_absolute_error(y_test, pred)))  #y_pred


# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: {:.2f}'.format(r2_score(y_test, pred)))   #y_pred
```

```
Mean squared error: 132034150.03
Mean absolute error: 7578.15
Coefficient of determination: 0.04
```

R-Squared (R^2 or the coefficient of determination) is a statistical measure in a regression model that determines the proportion of variance in the dependent variable that can be explained by the independent variable. In other words, r-squared shows how well the data fit the regression model (the goodness of fit).

R2 can take values from 0 to 1. A value of 1 indicates that the regression predictions perfectly fit the data. Results look like the model is not a good fit for car price prediction.

R-squared can take any values between 0 to 1, therefore it look like the Rando

Although the statistical measure provides some useful insights regarding the regression model, the user should not rely only on the measure in the assessment of a statistical model. The figure does not disclose information about the causation relationship between the independent and dependent variables.

In addition, it does not indicate the correctness of the regression model. Therefore, the user should always draw conclusions about the model by analyzing r-squared together with the other variables in a statistical model.

A high r-squared is good for the regression model however, sometimes a high r-squared can indicate the problems with the regression model.

A low r-squared figure is generally a bad sign for predictive models. However, in some cases, a good model may show a small value.

There is no universal rule on how to incorporate the statistical measure in assessing a model. The context of the experiment or forecast is extremely important and, in different scenarios, the insights from the metric can vary.

Method 3 : Linear Regression method

Split the data into training and testing dataset

In [60]:
```
car_audi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10668 entries, 0 to 10667
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   model         10668 non-null  object 
 1   year          10668 non-null  int64  
 2   price         10668 non-null  int64  
 3   transmission  10668 non-null  object 
 4   mileage       10668 non-null  int64  
 5   fuelType      10668 non-null  object 
 6   tax           10668 non-null  int64  
 7   mpg           10668 non-null  float64
 8   engineSize    10668 non-null  float64
dtypes: float64(2), int64(4), object(3)
memory usage: 750.2+ KB
```

In [61]:
```
ohe = pd.get_dummies(car_audi)
ohe.head()
```

Out[61]:

	year	price	mileage	tax	mpg	engineSize	model_ A1	model_ A2	model_ A3	model_ A4	...	model_ S8	model_ SQ5
0	2017	12500	15735	150	55.4	1.4	1	0	0	0	...	0	0
1	2016	16500	36203	20	64.2	2.0	0	0	0	0	...	0	0
2	2016	11000	29946	30	55.4	1.4	1	0	0	0	...	0	0
3	2017	16800	25952	145	67.3	2.0	0	0	0	1	...	0	0
4	2019	17300	1998	145	49.6	1.0	0	0	1	0	...	0	0

5 rows × 38 columns

In [62]:
```
X = ohe.drop(['price'], axis=1)
y = ohe['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
```

In [63]:
```
pipe = Pipeline([('scaler', StandardScaler()), ('LinReg', LinearRegression())])
```

In [64]:
```
pipe.fit(X_train, y_train)
```

Out[64]:
```
Pipeline(steps=[('scaler', StandardScaler()), ('LinReg', LinearRegression())])
```

In [65]:
```
pipe.score(X_train, y_train)
```

Out[65]: 0.889303418782393

In [66]:
```
y_pred = pipe.predict(X_test)
```

In [67]:
```
from sklearn.metrics import r2_score
r2_lin = r2_score(y_test, y_pred)
```

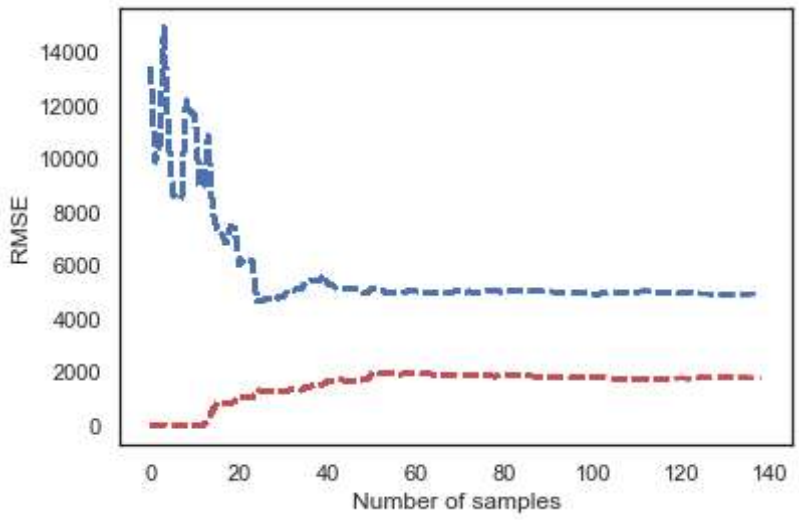
Let's plot learning curves (depending on the number of samples in the set). The RMSE should tend to converge closer to the max limit of samples.

Root mean squared error or RMSE is a measure of the difference between actual values and predicted values of a machine learning model like Linear Regression. Root mean squared error is a measure of how well the machine learning model can perform. The lower the RMSE, the better the model.

In [68]:
```
from sklearn.metrics import mean_squared_error

def plot_learning_curves (model, X, y):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3)
    train_errors, val_errors = [], []
    for m in range(1, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_pred = model.predict(X_train[:m])
        y_val_pred = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train_pred, y_train[:m]))
        val_errors.append(mean_squared_error(y_val_pred, y_val))
        plt.plot(np.sqrt(train_errors), 'r--', linewidth=2, label='train')
        plt.plot(np.sqrt(val_errors), 'b--', linewidth=2, label='val')
        plt.ylabel('RMSE')
        plt.xlabel('Number of samples')

linReg = LinearRegression()
plot_learning_curves(linReg, X[:200], y[:200])
```



The learning curves tell us that the RMSE stabilizes as long as the number of samples grow in volume.

In [69]:
```
# Splitting data into training and testing sets
from sklearn.model_selection import train_test_split
```

```
X_train, X_test,y_train, y_test=train_test_split(X,y, test_size=0.2) #20% test size
```

In [70]:
```
# To EXTRACT the Feature & label
X = car_audi.drop(columns = 'price') # Features = every columns except PRICE
y = car_audi['price']                # Regression problem, target is PRICE = Label
```

In [71]:
```
X
```

Out[71]:

	model	year	transmission	mileage	fuelType	tax	mpg	engineSize
0	A1	2017	Manual	15735	Petrol	150	55.4	1.4
1	A6	2016	Automatic	36203	Diesel	20	64.2	2.0
2	A1	2016	Manual	29946	Petrol	30	55.4	1.4
3	A4	2017	Automatic	25952	Diesel	145	67.3	2.0
4	A3	2019	Manual	1998	Petrol	145	49.6	1.0
...	...	...	...	...	...	...	...	...
10663	A3	2020	Manual	4018	Petrol	145	49.6	1.0
10664	A3	2020	Manual	1978	Petrol	150	49.6	1.0
10665	A3	2020	Manual	609	Petrol	150	49.6	1.0
10666	Q3	2017	Automatic	8646	Petrol	150	47.9	1.4
10667	Q3	2016	Manual	11855	Petrol	150	47.9	1.4

10668 rows × 8 columns

In [72]:
```
X.shape
```

Out[72]:
```
(10668, 8)
```

In [73]:
```
y.shape
```

Out[73]:
```
(10668,)
```

In [74]:
```
y
```

Out[74]:
```
0        12500
1        16500
2        11000
3        16800
4        17300
         ...
10663    16999
10664    16999
10665    17199
10666    19499
10667    15999
Name: price, Length: 10668, dtype: int64
```

In [75]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import OneHotEncoder
```

```
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
```

In [76]:
```
# Splitting data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test,y_train, y_test=train_test_split(X,y, test_size=0.2) #20% test size
```

One-hot encoding of categorical features:

We do this to avoid the misinterpretation of feature correlations by the ML algorithm.

In [77]:
```
ohe = OneHotEncoder()
ohe.fit(X)
#ohe.fit(X[['year','mileage', 'tax', 'mpg', 'engineSize']])
```

Out[77]:
```
OneHotEncoder()
```

In [78]:
```
ohe.categories_     # parameter categories
```

Out[78]:
```
[array([' A1', ' A2', ' A3', ' A4', ' A5', ' A6', ' A7', ' A8', ' Q2',
        ' Q3', ' Q5', ' Q7', ' Q8', ' R8', ' RS3', ' RS4', ' RS5', ' RS6',
        ' RS7', ' S3', ' S4', ' S5', ' S8', ' SQ5', ' SQ7', ' TT'],
       dtype=object),
 array([1997, 1998, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
        2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020],
       dtype=int64),
 array(['Automatic', 'Manual', 'Semi-Auto'], dtype=object),
 array([     1,      5,      7, ..., 152034, 168017, 323000], dtype=int64),
 array(['Diesel', 'Hybrid', 'Petrol'], dtype=object),
 array([  0,  20,  30, 115, 125, 135, 140, 145, 150, 155, 160, 165, 190,
        195, 200, 205, 220, 230, 235, 240, 260, 265, 280, 290, 295, 300,
        305, 315, 325, 330, 515, 535, 540, 555, 565, 570, 580], dtype=int64),
 array([ 18.9,  19.3,  19.6,  20.3,  21. ,  21.1,  21.4,  21.6,  21.9,
         22.1,  22.4,  22.6,  22.8,  23. ,  24.1,  24.8,  25. ,  26.4,
         26.9,  27.2,  27.4,  28.2,  28.5,  28.8,  29.1,  29.4,  29.7,
         30.1,  30.4,  30.7,  30.8,  31. ,  31.4,  31.7,  32.1,  32.5,
         32.8,  33.2,  33.3,  33.6,  34. ,  34.4,  34.5,  34.9,  35.3,
         35.8,  36.2,  36.7,  37.2,  37.6,  37.7,  38.1,  38.2,  38.7,
         39.2,  39.8,  40.4,  40.9,  41.5,  42.1,  42.2,  42.8,  43.4,
         43.5,  44.1,  44.5,  44.8,  45.6,  46.3,  47.1,  47.9,  48.7,
         49.6,  50. ,  50.4,  51.3,  51.4,  52.3,  53.3,  54.3,  55.4,
         56.5,  57.6,  57.7,  58.9,  60.1,  61.4,  62.8,  64.2,  65.7,
         67.3,  68.9,  70.6,  72.4,  74.3,  76.3,  76.4,  80.7,  83.1,
        117.7, 141.3, 156.9, 176.6, 188.3]),
 array([0. , 1. , 1.2, 1.4, 1.5, 1.6, 1.8, 1.9, 2. , 2.5, 2.7, 2.9, 3. ,
        3.2, 4. , 4.1, 4.2, 5.2, 6.3])]
```

In [79]:
```
car_audi.head()
```

Out[79]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	A1	2017	12500	Manual	15735	Petrol	150	55.4	1.4
1	A6	2016	16500	Automatic	36203	Diesel	20	64.2	2.0
2	A1	2016	11000	Manual	29946	Petrol	30	55.4	1.4
3	A4	2017	16800	Automatic	25952	Diesel	145	67.3	2.0
4	A3	2019	17300	Manual	1998	Petrol	145	49.6	1.0

Use transformer

In [80]:
```
column_trans = make_column_transformer((OneHotEncoder(categories=ohe.categories_), ['mo
```

In [81]:
```
lr=LinearRegression()
```

In [82]:
```
#Sequentially apply a list of transforms and a final estimator.

pipe=make_pipeline(column_trans,lr)
```

In [83]:
```
# Train the model
pipe.fit(X_train,y_train)
```

Out[83]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(categories=[array([' A
1', ' A2', ' A3', ' A4', ' A5', ' A6', ' A7', ' A8', ' Q2',
       ' Q3', ' Q5', ' Q7', ' Q8', ' R8', ' RS3', ' RS4', ' RS5', ' RS6',
       ' RS7', ' S3', ' S4', ' S5', ' S8', ' SQ5', ' SQ7', ' TT'],
      dtype=object),
                                                                              array([1997,
1998, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 200...
        49.6,  50. ,  50.4,  51.3,  51.4,  52.3,  53.3,  54.3,  55.4,
        56.5,  57.6,  57.7,  58.9,  60.1,  61.4,  62.8,  64.2,  65.7,
        67.3,  68.9,  70.6,  72.4,  74.3,  76.3,  76.4,  80.7,  83.1,
       117.7, 141.3, 156.9, 176.6, 188.3]),
                                                                              array([0. ,
1. , 1.2, 1.4, 1.5, 1.6, 1.8, 1.9, 2. , 2.5, 2.7, 2.9, 3. ,
       3.2, 4. , 4.1, 4.2, 5.2, 6.3])]),
                                                  ['model', 'year',
                                                   'transmission', 'mileage',
                                                   'fuelType', 'tax', 'mpg',
                                                   'engineSize'])])),
                ('linearregression', LinearRegression())])
```

In [84]:
```
pipe.score(X_train, y_train)
```

Out[84]:
```
0.9849589657830721
```

In [85]:
```
X_train.head()
```

Out[85]:

	model	year	transmission	mileage	fuelType	tax	mpg	engineSize
851	Q2	2019	Automatic	2402	Diesel	145	47.9	1.6
6834	A7	2019	Semi-Auto	7968	Diesel	145	38.7	3.0
7089	Q7	2017	Semi-Auto	15328	Diesel	300	39.2	4.0
2724	Q2	2017	Manual	44551	Petrol	125	50.4	1.4
6862	Q5	2016	Semi-Auto	39015	Diesel	205	47.1	2.0

In [86]:
```
y_pred = pipe.predict(X_test)
```

In [87]:
```
y_pred
```

Out[87]:
```
array([21390.95186074, 27141.94567819, 20034.66152579, ...,
       32560.04659975, 14297.18952104, 20358.81478714])
```

Step 5 - Compare 'Desired Output' vs 'Predicted Output'

In [88]:
```
compare_car_audi = pd.DataFrame({"Desired Output (price)": y_test,
                                 "Predicted Output (price)": y_pred})
```

In [89]:
```
compare_car_audi[:10]
```

Out[89]:

	Desired Output (price)	Predicted Output (price)
2746	21294	21390.951861
6875	28599	27141.945678
8338	19999	20034.661526
7416	5995	7958.761826
4653	33990	29433.397129
1628	19990	21116.963982
3858	17491	16701.259882
5895	24990	27924.385162
1821	11995	16254.270733
7741	23888	20000.417404

Step 6 - Evaluate the Linear Regression Model

In [90]:
```
r2_score(y_test,y_pred)
```

Out[90]:
```
0.8976693503511738
```

In [91]:
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# The coefficients
print('Coefficients: \n', lr.coef_)

# The mean squared error
print('Mean squared error: {:.2f}'.format(mean_squared_error(y_test, y_pred)))
print('Mean absolute error: {:.2f}'.format(mean_absolute_error(y_test, y_pred)))


# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: {:.2f}'.format(r2_score(y_test, y_pred)))
```

```
Coefficients:
 [ -8031.87361279 -10565.67811091  -5997.86588135 ...  -5313.08322033
  42277.14046277  12527.71784116]
Mean squared error: 13263080.33
Mean absolute error: 2586.51
Coefficient of determination: 0.90
```

In [92]:
```
# Evaluate the model's training score and test score
print("Regression model's training score = {:.2f}".format(pipe.score(X_train, y_train)))
print("Regression model's test score     = {:.2f}".format(pipe.score(X_test, y_test)))
```

```
Regression model's training score = 0.98
```

```
Regression model's test score     = 0.90
```

My own Car Price prediction using Linear Regression Model

In [93]:
```
#Create portable serialized representations of Python objects.
import pickle
```

In [94]:
```
#Write a pickled representation of obj to the open file object file.

pickle.dump(pipe,open('LinearRegressionModel.pkl', 'wb'))
```

In [95]:
```
pipe.predict(pd.DataFrame([[' A3', 2017, 'Manual', 15000, 'Petrol', 150, 55.4, 1.4]], c
```

Out[95]:
```
array([19655.93722307])
```

In [96]:
```
compare_car_audi = pd.DataFrame({"Desired Output (Price)": y_test,
                                 "Predicted Output (Price)": y_pred})
```

In [97]:
```
compare_car_audi[:10]
```

Out[97]:

	Desired Output (Price)	Predicted Output (Price)
2746	21294	21390.951861
6875	28599	27141.945678
8338	19999	20034.661526
7416	5995	7958.761826
4653	33990	29433.397129
1628	19990	21116.963982
3858	17491	16701.259882
5895	24990	27924.385162
1821	11995	16254.270733
7741	23888	20000.417404

Above modeling trained & tested successfully !!!

Conclusion :

Using Mutilpe Linear Regression model

R2 - Coefficient of determination = 0.9 (are much higher) MAE = 2586.51 (are much lower),

comparing to

Support Vector Machine model

R2 - Coefficient of determination = 0.04 (are much lower) MAE = 7578.15 (are much higher)

and

Random Forest model

R2 - Coefficient of determination = 0.04 (are much lower) MAE = 7578.15 (are much higher)

Therefore my analysis concluded that Linear Regression model is much accurate/better in predicting Audi car prices.

End of the notebook

In [ ]: