Hyundi Price Prediction

January 12, 2022

1 Preliminary project Explanation:

In this project, I am processing a dataset on the number of **Hyundai** cars sold in the UK in recent years. I collected this dataset from the **Kaggle** website, which you can find in the zip file. Fortunately, this dataset does not need to be cleaned as the author has already done that, but I will first try to check the **missing, duplicate and NA values** and correct them in the preprossing part. This dataset contains three object columns (model, transmission, fuel type) that need to be coded before processing the whole dataset. I will try to do this using libraries and algorithms like **OneHotEncoder** and **LabelEncoder**. In this project, I encounter the question and problem of which features of this dataset have the most impact on car prices. So I will try to find the best feature that has a stronger correlation with the price.

Moreover, I would like to predict the price of the cars using the **ML algorithms** that we can use for this kind of dataset. Generally, price prediction datasets are classified as **supervised datasets**, so my project falls into this category. I will use **regression models** such as **Linier Regression, Desicion Tree and Randomforrest** in this project. I will split the dataset into a **training (70%) and test (30%) set**, scale the dataset using the **standard scaler** and **minmax scaler**, find the **Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R2 value, and Cross-validation value**, and compare the models using these 4 error types by using statistic result and plots. Finally, after comparing the results of each model, I can propose the best model with the highest accuracy for predicting the price for this dataset.

I encounterd with some chalanges in my project such as finding some unusable values that could potentially destroy the integrity of the dataset and affect the results of the models. I also had to choose the best function for encoding the object columns between **LabelEncoder** and **OneHotEncoder**, In this case, I ran the project separately with both functions written to the project and compared both results and found that **OneHotEncoder** was better for encoding the object columns due to the number of models I have in the model columns (I wrote the LabelEncoder code as a comment in a chunk). Also, I had a same problem when I wanted to choose the better function for scaling the project. So, I ran the project with both the **standardscaler** and the MinMaxScaler and finally found that due to the large gap between min and max values in some columns, selecting the **MinMaxScaler** makes more sense for the project. Also, I tried to visualize the result of the model error by using some graphs to show the difference between the errors in each model, and I think this part of my project were the strong aspects of my analysis.

so my project objectives summarized to this three steps :

To perform EDA to understand the underlying trends

Clean the data and prepare it for regression models

Fit different regression models and compare their performance at predicting the price of the cars

2 Dataset

dataset is about the number of **Hyundai cars** sold in UK during the last yeras.The cleaned data set contains information of price, transmission, mileage, fuel type, road tax, miles per gallon (mpg), and engine size.inaddition contains **4860** rows that we encounter with them during the project.

Model:model of car

Year:the year that cars made

Price:price of the car

Transmission:type of gear

Milage:how many miles the car went(1 mile = 1,609344 km)

Fueltpe: fuel type

Tax(£):tax

Mpg :miles per galon (i galon = 3,78541178 liters)

Enginsize: size of engin (liters)

3 Importing the packages needed for the analysis

4

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

from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪LeaveOneOut
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import validation_curve
from sklearn.metrics import mean_squared_error , r2_score ,accuracy_score
from sklearn.ensemble import RandomForestRegressor
```

```
plt.style.use('fivethirtyeight')
import warnings
warnings.filterwarnings('ignore')
```

5 Preprocessing and Dataset Observation

First of all we start with reading the data set by using (pd.read_scv) to prepare the dataset for our project:

[2]:
```
cars = pd.read_csv("hyundi.csv", header = 0, sep =",")
```

As I said before, this data set is clear , but we should try to convince ourselves that the data set is completely clear, so let us try to find the possibility of null and duplicate values in our data set.

[3]:
```
cars.isnull().any()
```

[3]:
```
model           False
year            False
price           False
transmission    False
mileage         False
fuelType        False
tax(£)          False
mpg             False
engineSize      False
dtype: bool
```

Good news ! after run the chunk above we find that the dataset does not contains any Null values.

Now try to find the duplicate values :

[4]:
```
cars = cars.drop_duplicates(keep='first').reset_index(drop=True)
cars.shape
```

[4]:
```
(4774, 9)
```

Ooops, after run the chunk above we find that the dataset contains **86 duplicate values** which was dropped from the dataset immediately.

The info() function is used to print a concise summary of a DataFrame. This method prints information about a DataFrame including the index dtype and column dtypes, non-null values and memory usage which are important for us to use them in the project

[5]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4774 entries, 0 to 4773
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
```

```
---  ------        --------------  -----
 0   model         4774 non-null   object
 1   year          4774 non-null   int64
 2   price         4774 non-null   int64
 3   transmission  4774 non-null   object
 4   mileage       4774 non-null   int64
 5   fuelType      4774 non-null   object
 6   tax(£)        4774 non-null   int64
 7   mpg           4774 non-null   float64
 8   engineSize    4774 non-null   float64
dtypes: float64(2), int64(4), object(3)
memory usage: 335.8+ KB
```

Pandas head() method is used to return top n (5 by default) rows of a data frame

[6]:
```
cars.head()
```

[6]:
```
    model  year  price transmission  mileage fuelType  tax(£)   mpg  \
0     I20  2017   7999       Manual    17307   Petrol     145  58.9
1  Tucson  2016  14499    Automatic    25233   Diesel     235  43.5
2  Tucson  2016  11399       Manual    37877   Diesel      30  61.7
3     I10  2016   6499       Manual    23789   Petrol      20  60.1
4    IX35  2015  10199       Manual    33177   Diesel     160  51.4

   engineSize
0         1.2
1         2.0
2         1.7
3         1.0
4         2.0
```

Pandas tail() method is used to return last n (5 by default) rows of a data frame

[7]:
```
cars.tail()
```

[7]:
```
       model  year  price transmission  mileage fuelType  tax(£)   mpg  \
4769     I30  2016   8680       Manual    25906   Diesel       0  78.4
4770     I40  2015   7830       Manual    59508   Diesel      30  65.7
4771     I10  2017   6830       Manual    13810   Petrol      20  60.1
4772  Tucson  2018  13994       Manual    23313   Petrol     145  44.8
4773  Tucson  2016  15999    Automatic    11472   Diesel     125  57.6

      engineSize
4769         1.6
4770         1.7
4771         1.0
4772         1.6
4773         1.7
```

In the next three columns below, we will try to find the unique elements and the number of unique elements in the three object columns (model, transmission, and fuel type). in addition we provide a plot for each columns afterwards

[8]:
```python
cars["model"].unique()
```

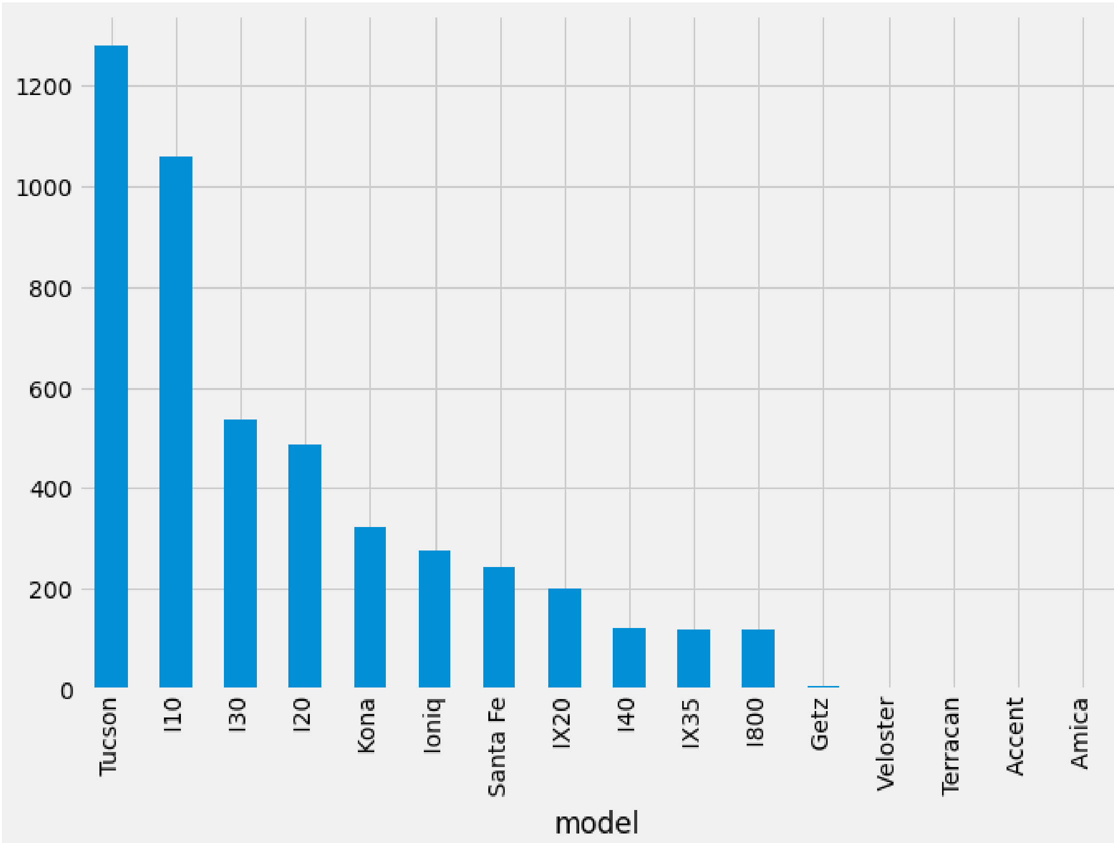
[8]:
```
array([' I20', ' Tucson', ' I10', ' IX35', ' I30', ' I40', ' Ioniq',
       ' Kona', ' Veloster', ' I800', ' IX20', ' Santa Fe', ' Accent',
       ' Terracan', ' Getz', ' Amica'], dtype=object)
```

[9]:
```python
cars["model"].value_counts()
```

[9]:
```
 Tucson      1280
 I10         1061
 I30          535
 I20          487
 Kona         322
 Ioniq        275
 Santa Fe     244
 IX20         202
 I40          120
 IX35         118
 I800         117
 Getz           6
 Veloster       3
 Terracan       2
 Accent         1
 Amica          1
Name: model, dtype: int64
```

[10]:
```python
plt.figure(figsize=(10, 7))
cars.value_counts(cars["model"]).plot.bar()
```

[10]:
```
<AxesSubplot:xlabel='model'>
```



[11]:
```
cars["transmission"].unique()
```

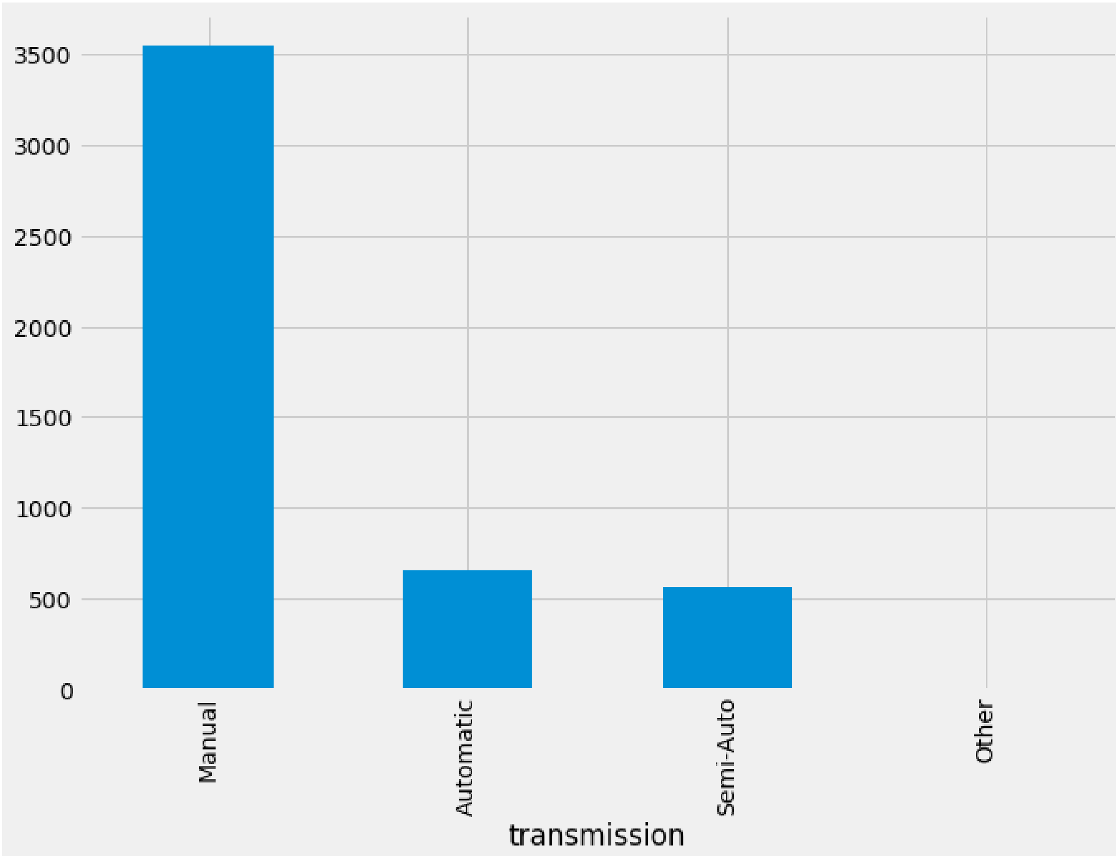
[11]:
```
array(['Manual', 'Automatic', 'Semi-Auto', 'Other'], dtype=object)
```

[12]:
```
cars["transmission"].value_counts()
```

[12]:
```
Manual       3546
Automatic     658
Semi-Auto     568
Other           2
Name: transmission, dtype: int64
```

[13]:
```
plt.figure(figsize=(10, 7))
cars.value_counts(cars["transmission"]).plot.bar()
```

[13]:
```
<AxesSubplot:xlabel='transmission'>
```



[14]:
```
cars["fuelType"].unique()
```

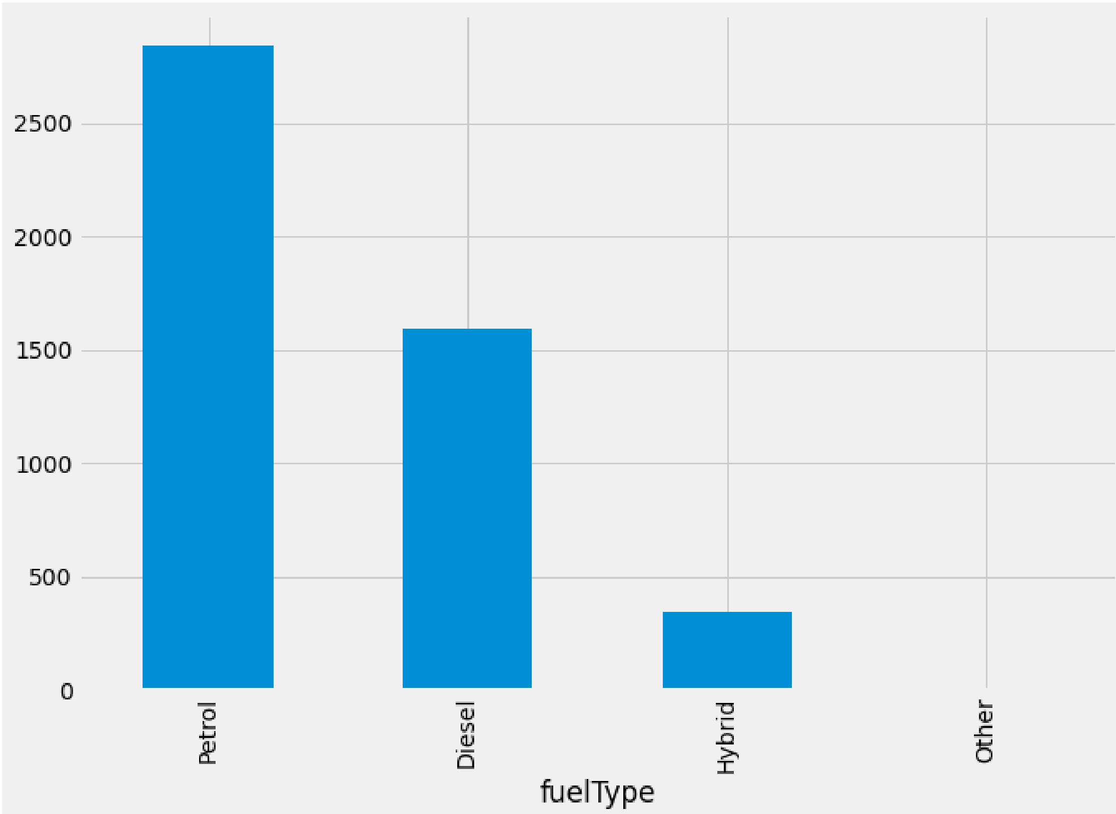
[14]:
```
array(['Petrol', 'Diesel', 'Hybrid', 'Other'], dtype=object)
```

[15]:
```
cars["fuelType"].value_counts()
```

[15]:
```
Petrol    2838
Diesel    1595
Hybrid     340
Other        1
Name: fuelType, dtype: int64
```

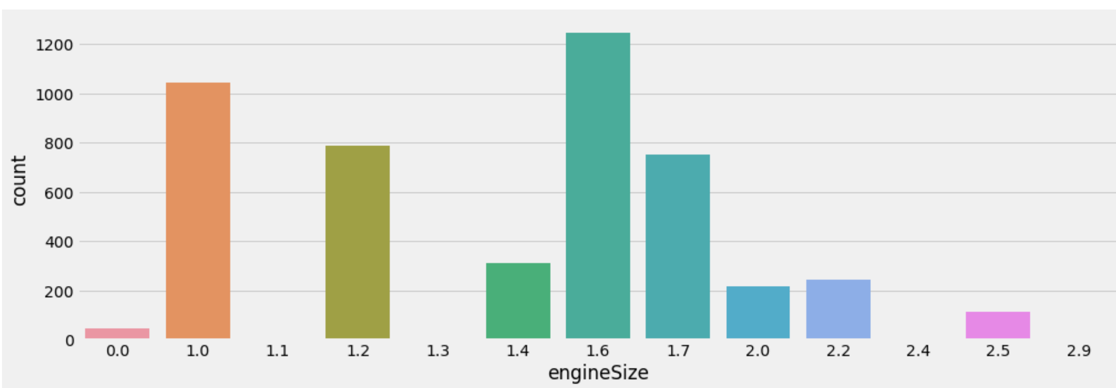
[16]:
```
plt.figure(figsize=(10, 7))
cars.value_counts(cars["fuelType"]).plot.bar()
```

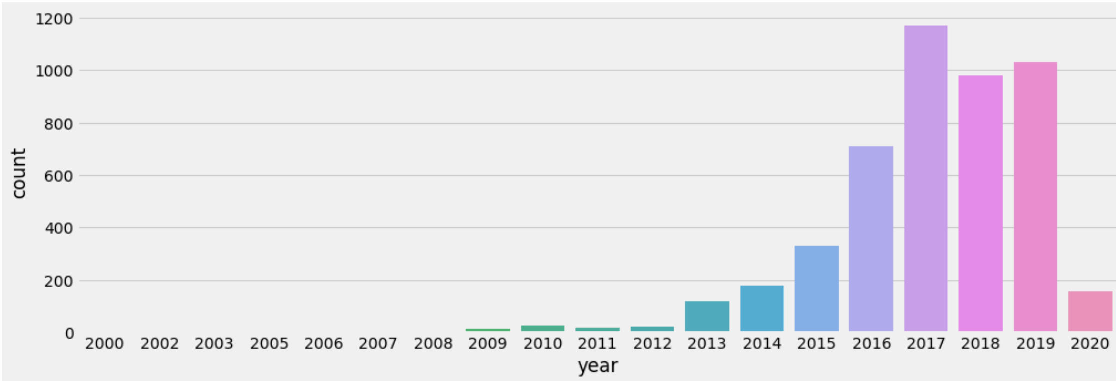
[16]:
```
<AxesSubplot:xlabel='fuelType'>
```



The next two charts show the number of cars with different engine sizes and the number of cars sold in different years

[17]:
```
list = [ 'engineSize', 'year']
for i in list:
    plt.figure(figsize=(15, 5))
    sns.countplot(cars[i])
    plt.show()
```





The **describe()** function computes a summary of statistics pertaining to the DataFrame columns. This function gives the mean, std and IQR values. And, function excludes the character columns and given summary about numeric columns.

[18]:
```
cars.describe()
```

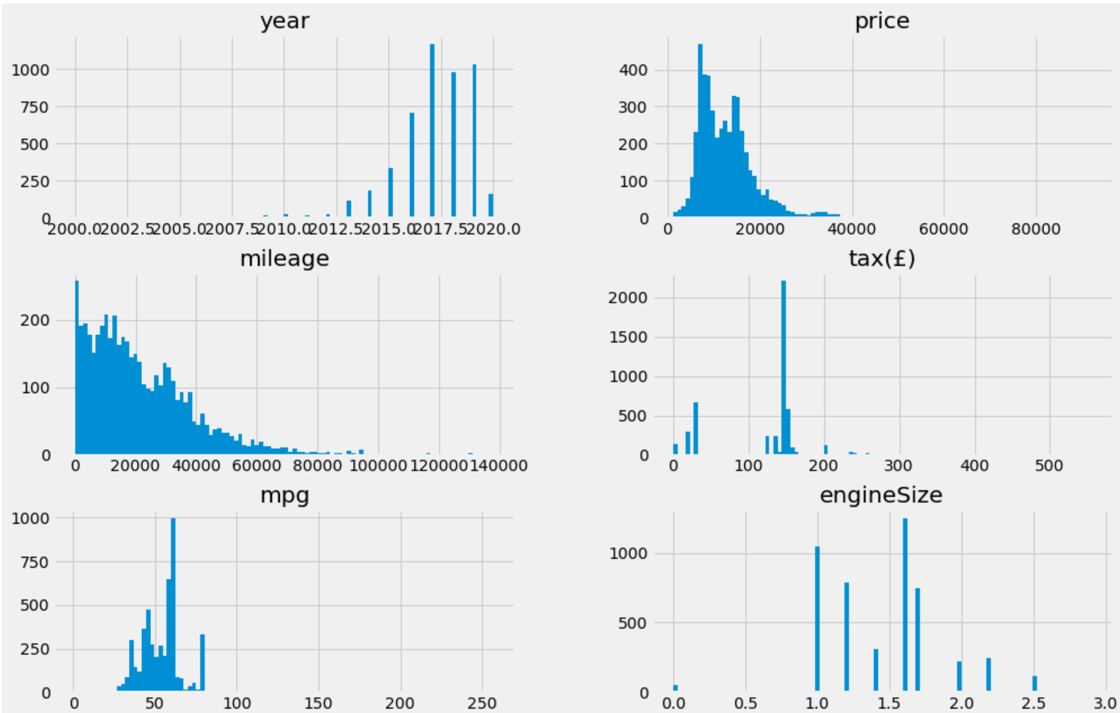
[18]:
```
              year         price        mileage      tax(£)         mpg  \
count  4774.000000   4774.000000    4774.000000  4774.000000  4774.000000
mean   2017.092166  12727.809384   21658.914537   121.187683    53.837956
std       1.921323   5976.925227   17618.489657    58.135472    12.740499
min    2000.000000   1200.000000       1.000000     0.000000     1.100000
25%    2016.000000   8000.000000    8542.500000   125.000000    44.800000
50%    2017.000000  11992.500000   17627.000000   145.000000    55.400000
75%    2018.000000  15695.000000   31067.500000   145.000000    60.100000
max    2020.000000  92000.000000  138000.000000   555.000000   256.800000

        engineSize
count  4774.000000
mean      1.460285
std       0.401858
min       0.000000
25%       1.200000
50%       1.600000
75%       1.700000
max       2.900000
```

Create histograms to display the values of the numerical columns

[19]:
```
cars.hist(bins = 100, figsize = (15,10))
```

[19]:
```
array([[<AxesSubplot:title={'center':'year'}>,
        <AxesSubplot:title={'center':'price'}>],
       [<AxesSubplot:title={'center':'mileage'}>,
        <AxesSubplot:title={'center':'tax(£)'}>],
```

```
[<AxesSubplot:title={'center':'mpg'}>,
 <AxesSubplot:title={'center':'engineSize'}>]], dtype=object)
```



Change of name from tax(£) to tax due to simplify use it in the project.

```
[20]: cars.rename(columns= {'tax(£)': 'tax'}, inplace = True)
cars_edit = cars.copy()
cars_edit
```

```
[20]:        model  year  price transmission  mileage fuelType  tax   mpg  \
0          I20  2017   7999       Manual    17307   Petrol  145  58.9
1       Tucson  2016  14499    Automatic    25233   Diesel  235  43.5
2       Tucson  2016  11399       Manual    37877   Diesel   30  61.7
3          I10  2016   6499       Manual    23789   Petrol   20  60.1
4         IX35  2015  10199       Manual    33177   Diesel  160  51.4
…          …    …       …            …        …       …    …    …
4769       I30  2016   8680       Manual    25906   Diesel    0  78.4
4770       I40  2015   7830       Manual    59508   Diesel   30  65.7
4771       I10  2017   6830       Manual    13810   Petrol   20  60.1
4772    Tucson  2018  13994       Manual    23313   Petrol  145  44.8
4773    Tucson  2016  15999    Automatic    11472   Diesel  125  57.6

      engineSize
0            1.2
```

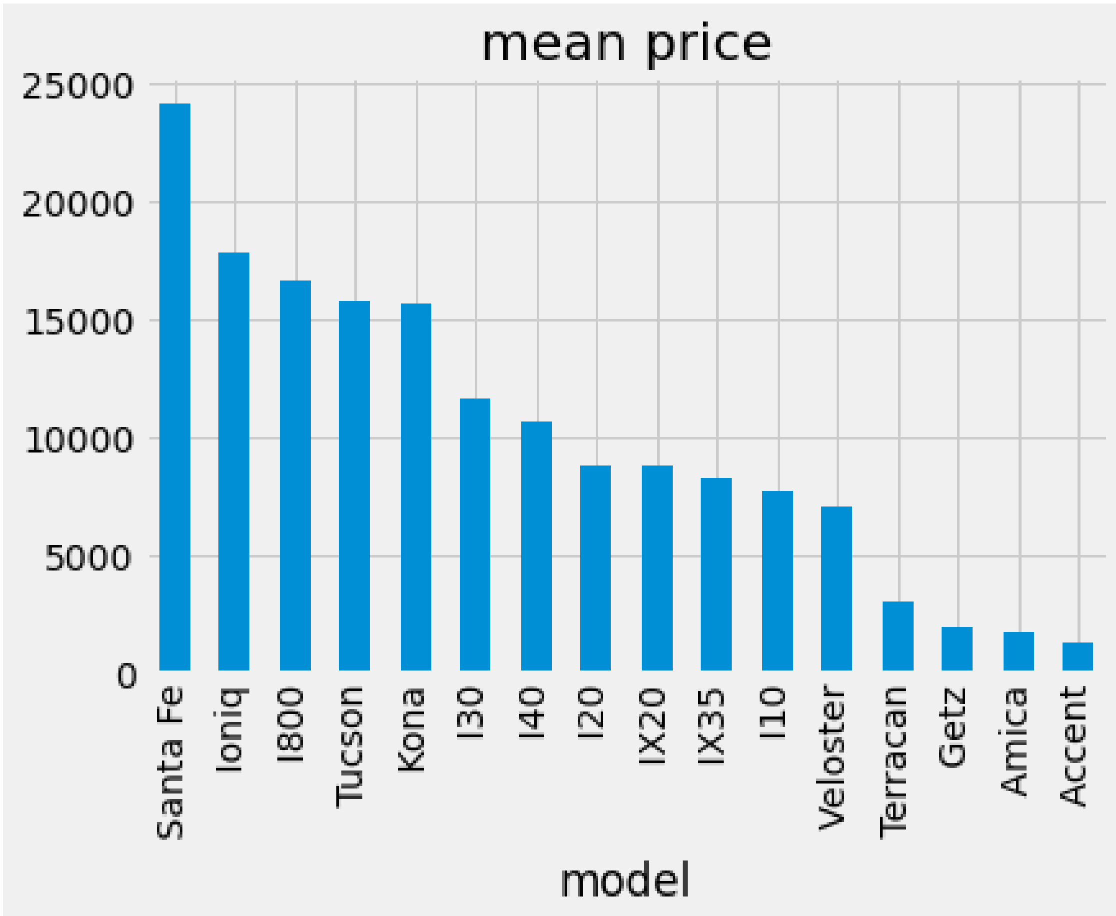
```
1            2.0
2            1.7
3            1.0
4            2.0
…          …
4769         1.6
4770         1.7
4771         1.0
4772         1.6
4773         1.7

[4774 rows x 9 columns]
```

During the initial observation of the dataset, I found the following line referring to the Hyundei model I10, but its price is about 10 times higher than the average price of this model according to the whole dataset, so I determined that this price is not real and replaced it with the rational price to maintain the integrity of the dataset

[21]:
```python
cars[cars["price"] == 92000]
```

[21]:
```
     model  year  price transmission  mileage fuelType  tax   mpg  engineSize
4165   I10  2017  92000    Automatic    35460   Petrol  150  47.9         1.2
```

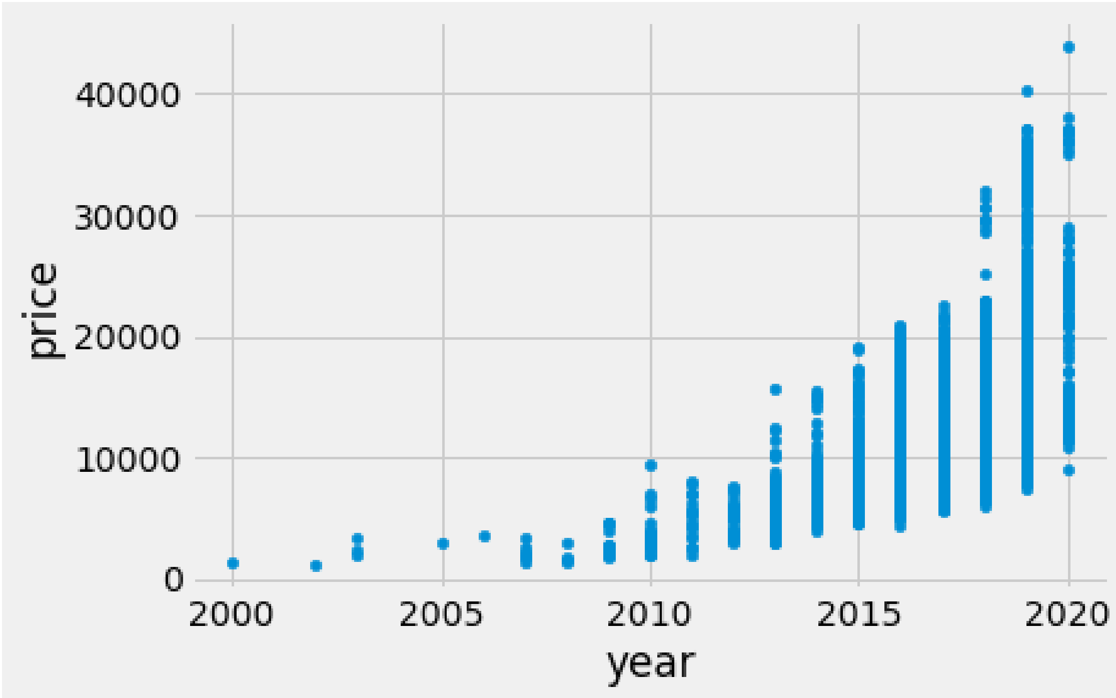
[22]:
```python
# we can see that there is a maximum value which is compeletly different with␣
↪other values
# i checked it and found this value is a human mistake when they wanted to fill␣
↪the dataset
# this value equals 92000 which i consider it as 9200 in my data to Maintain␣
↪the dataset's integrity
model = cars_edit.groupby(['model']).mean()['price'].
↪sort_values(ascending=False)
model.plot(kind='bar',title = 'mean price')
cars_edit.loc[cars_edit.price > 80000, 'price']= 9200
```



Now we would like to determine the correlation of the numeric columns with the price column to get a better insight into these columns: We use a scatter plot to show these correlations

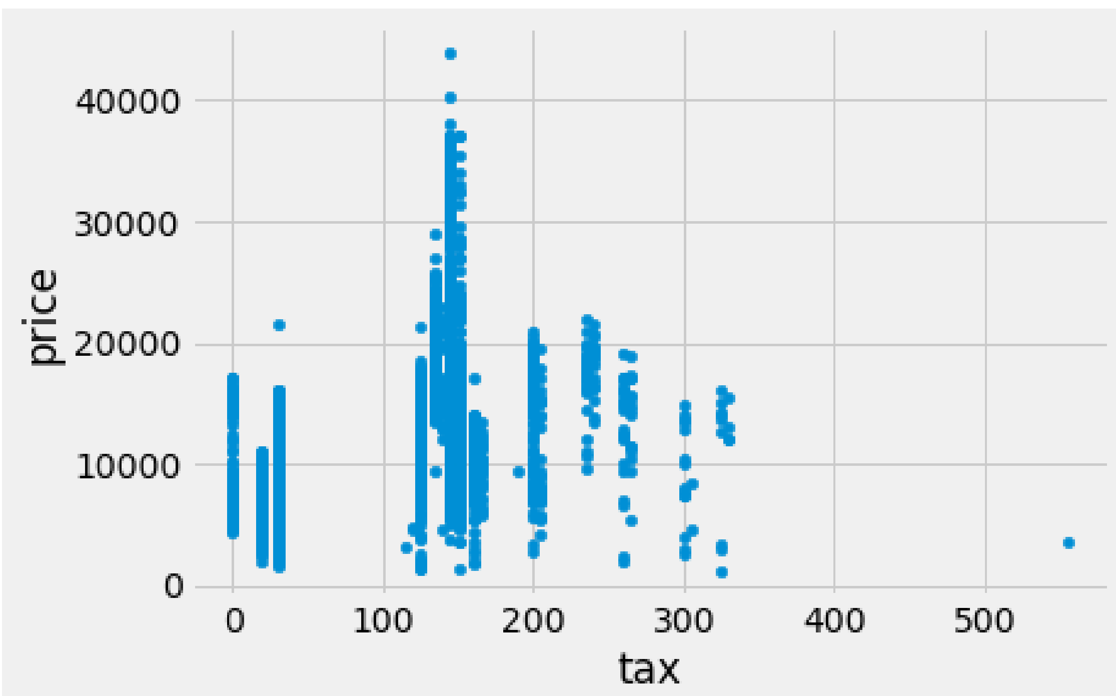
[23]:
```
cars_edit.plot.scatter(x = "year", y = "price")
```

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



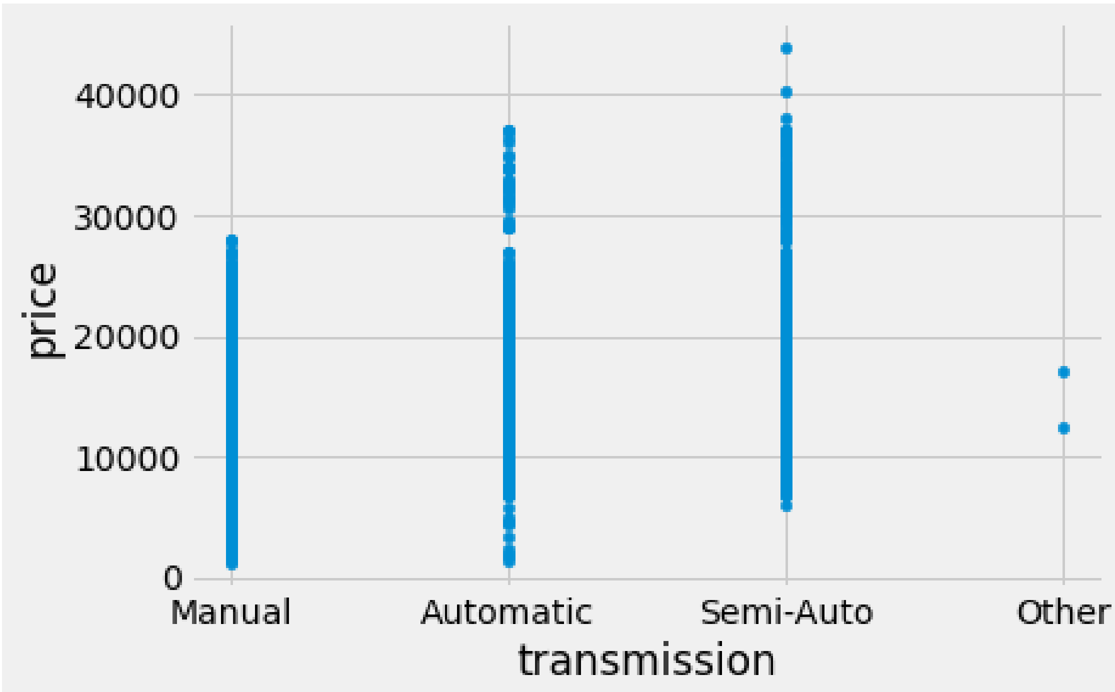
[24]:
```
cars_edit.plot.scatter(x = "tax", y = "price")
```

[24]: <AxesSubplot:xlabel='tax', ylabel='price'>



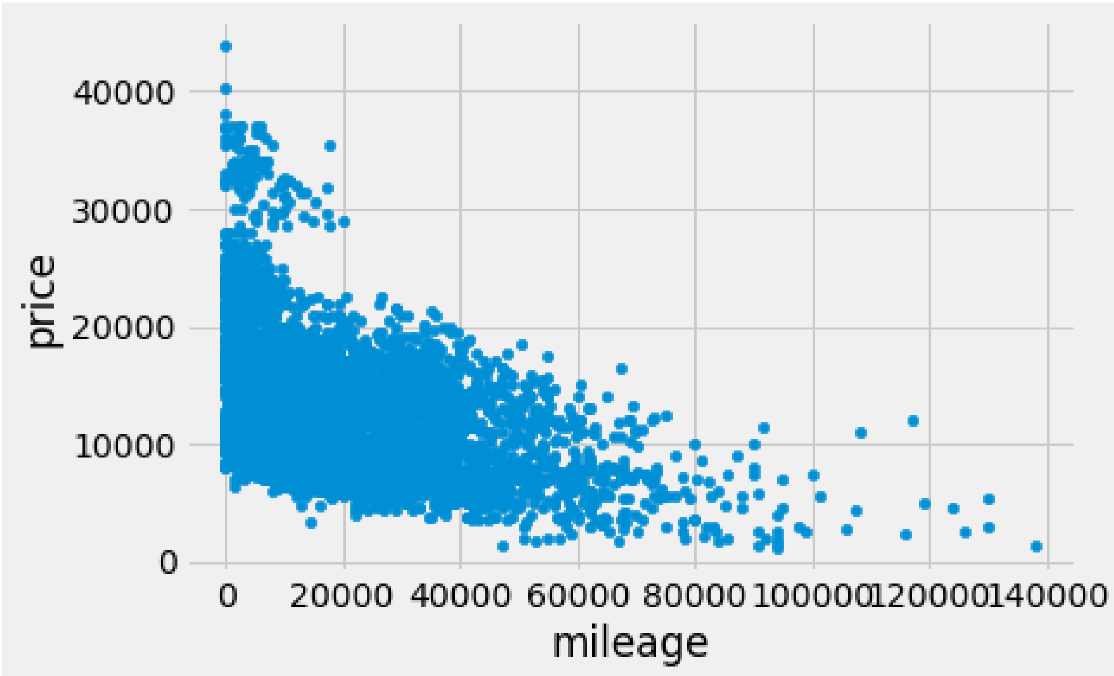
[25]:
```
cars_edit.plot.scatter(x = "transmission", y = "price")
```

[25]: <AxesSubplot:xlabel='transmission', ylabel='price'>



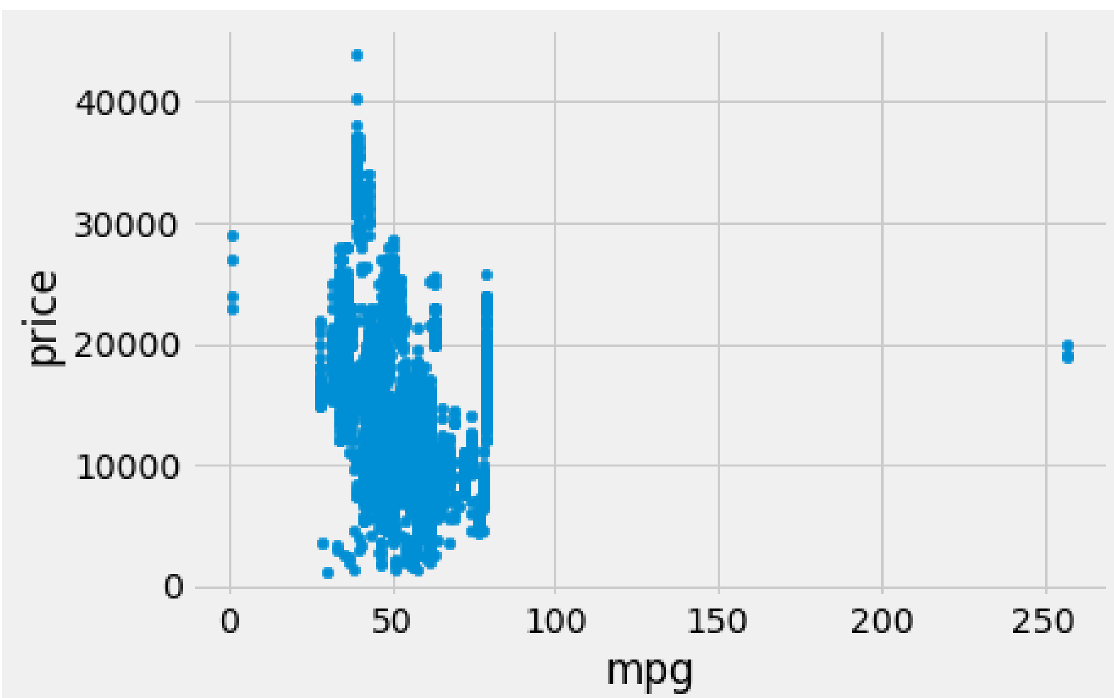
[26]:
```
cars_edit.plot.scatter(x = "mileage", y = "price")
```

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



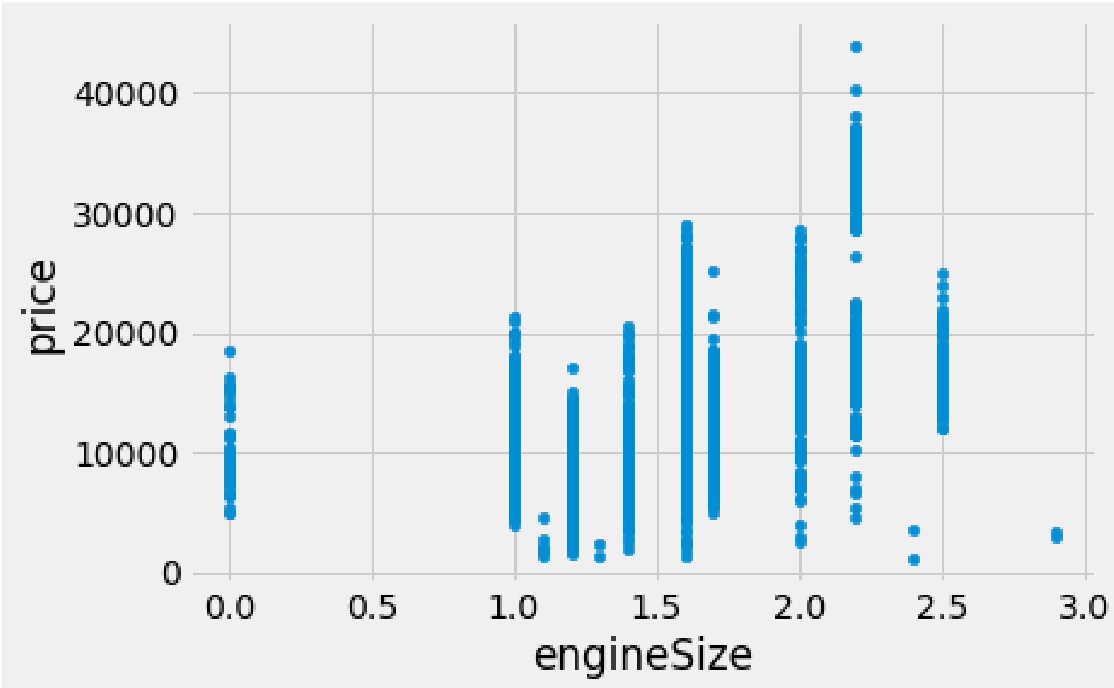

```
[27]: cars_edit.plot.scatter(x = "mpg", y = "price")
```

```
[27]: <AxesSubplot:xlabel='mpg', ylabel='price'>
```



[28]:
```
cars_edit.plot.scatter(x = "engineSize", y = "price")
```

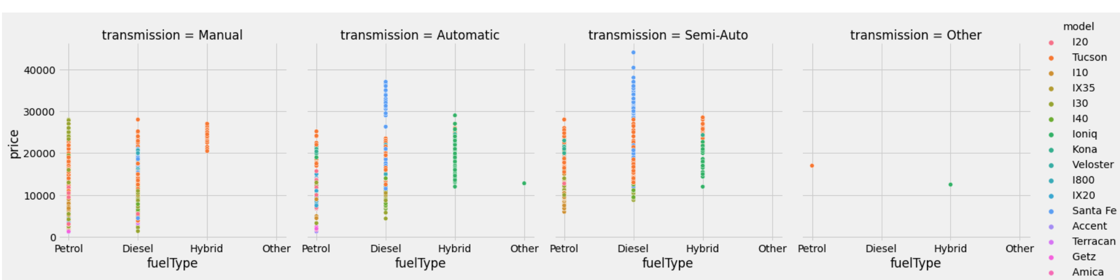
[28]:
```
<AxesSubplot:xlabel='engineSize', ylabel='price'>
```



This is a figure-level function for visualizing statistical relationships between columns to find the rational relationship between them

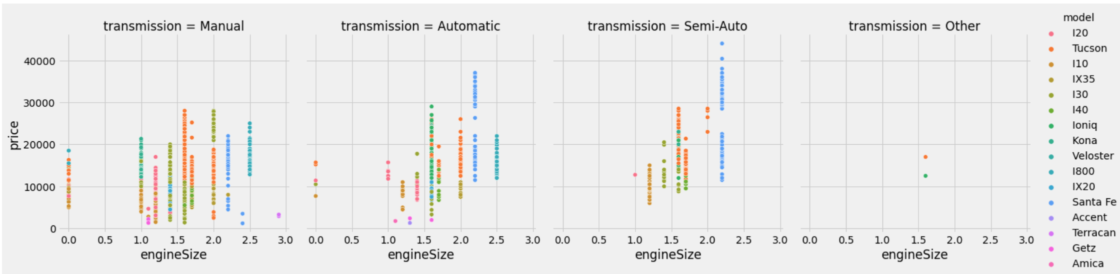
[29]:
```
sns.relplot(data=cars_edit, x="fuelType", y="price", hue="model",col␣
↪="transmission" )
```

[29]:
```
<seaborn.axisgrid.FacetGrid at 0x7fcbb8c8af70>
```



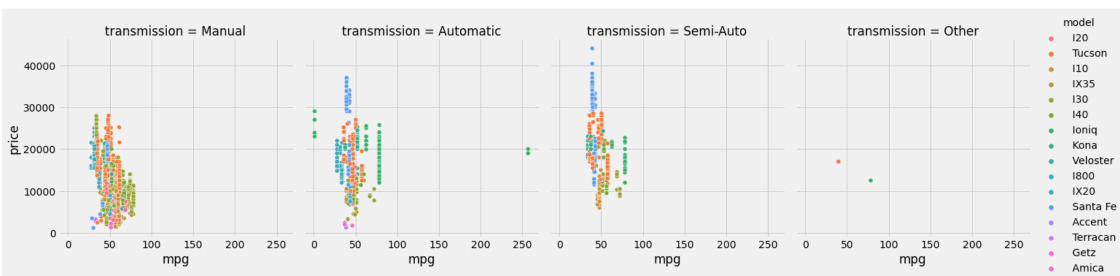
[30]:
```
sns.relplot(data=cars_edit, x="engineSize", y="price", hue="model",col␣
↪="transmission" )
```

[30]: <seaborn.axisgrid.FacetGrid at 0x7fcbb878c310>



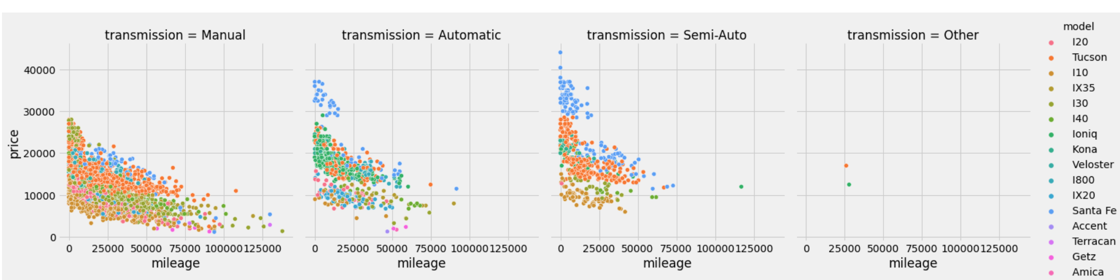
[31]:
```
sns.relplot(data=cars_edit, x="mpg", y="price", hue="model",col ="transmission"
↪)
```

[31]: <seaborn.axisgrid.FacetGrid at 0x7fcbb8c8ad30>



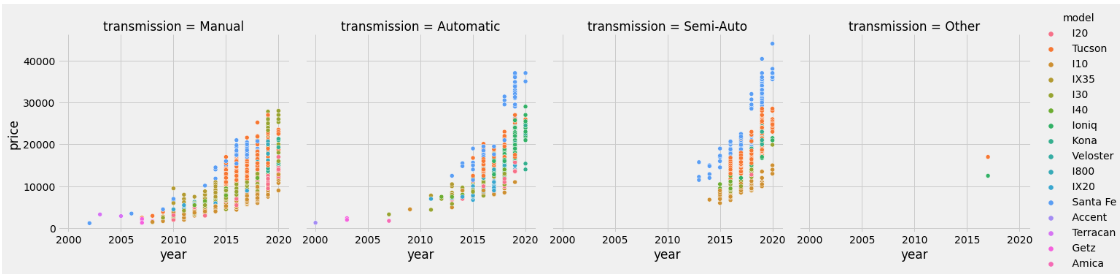
[32]:
```
sns.relplot(data=cars_edit, x="mileage", y="price", hue="model",col
↪="transmission" )
```

[32]: <seaborn.axisgrid.FacetGrid at 0x7fcbb6d86e50>



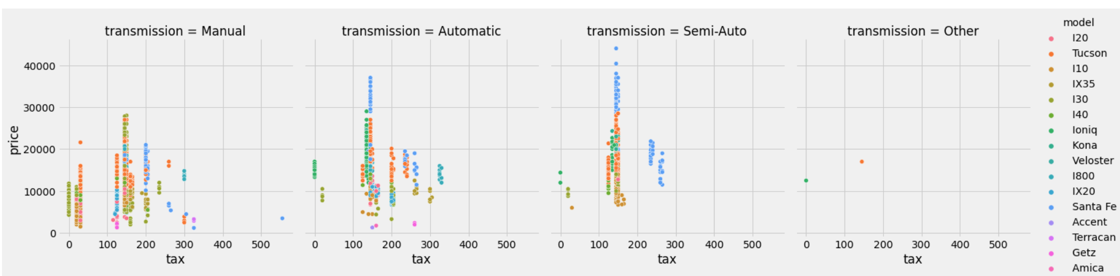
[33]:
```
sns.relplot(data=cars_edit, x="year", y="price", hue="model",col
↪="transmission" )
```

[33]: <seaborn.axisgrid.FacetGrid at 0x7fcbb6a967c0>



[34]:
```python
sns.relplot(data=cars_edit, x="tax", y="price", hue="model",col ="transmission")
```

[34]: <seaborn.axisgrid.FacetGrid at 0x7fcbb68ff490>



[35]:
```python
cars.corr()["price"].sort_values()
```

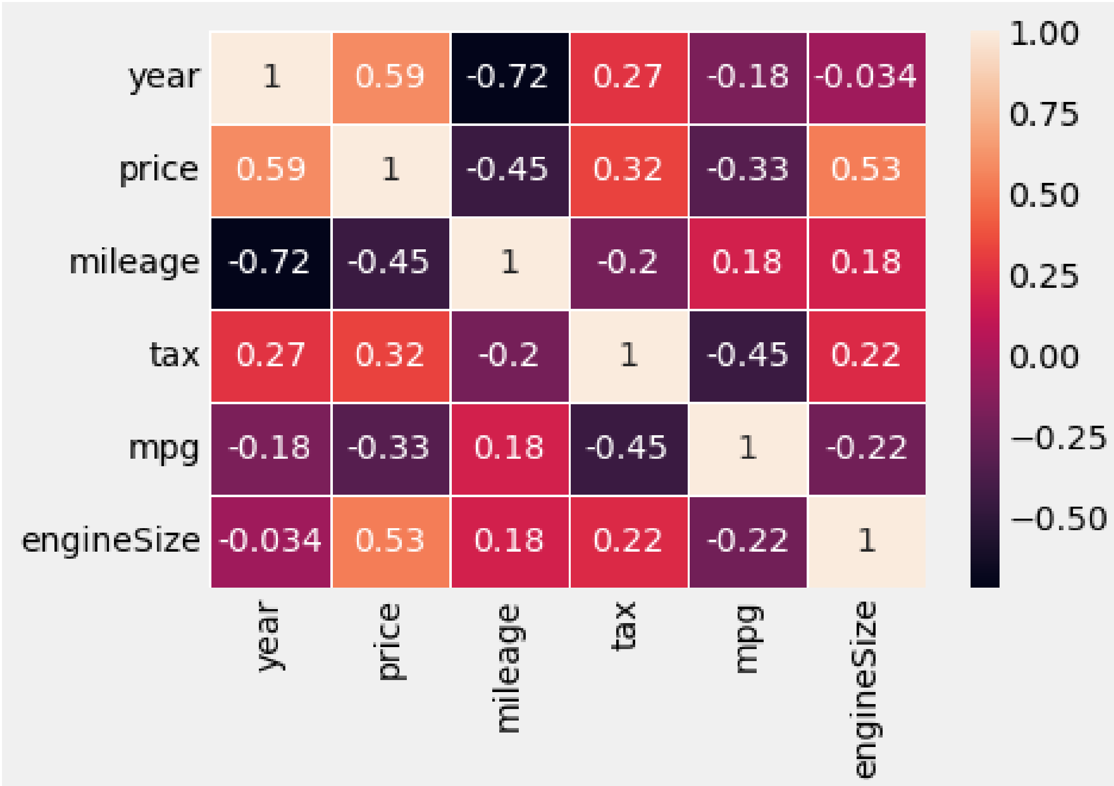
[35]:
```
mileage      -0.443754
mpg          -0.323742
tax           0.318448
engineSize    0.521832
year          0.575325
price         1.000000
Name: price, dtype: float64
```

Now we use a heat map plot to show the correlations between the columns and find out which columns are more strongly correlated with price.

From this heatmap, we see that year, engine, and tax have a stronger positive correlation with price, while mileage and mileage have a negative correlation with the price column.

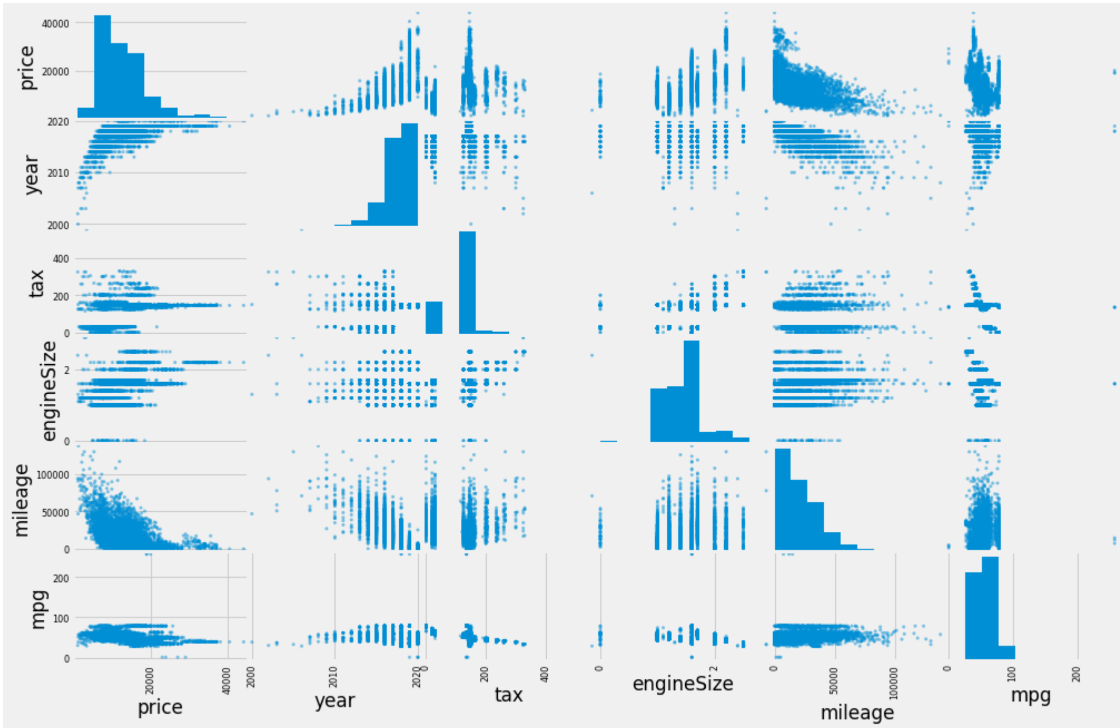
[36]:
```python
sns.heatmap(cars_edit.corr(), annot=True, linewidths=1)
```

[36]: <AxesSubplot:>



this is a grid of scatter plots to visualize bivariate relationships between combinations of variables. Each scatter plot in the matrix visualizes the relationship between a pair of variables, allowing many relationships to be explored in one chart.

[37]:
```
features = ["price","year","tax","engineSize","mileage","mpg"]
scatter_matrix(cars_edit[features], figsize = (15,10))
plt.show()
```



6 Summary of Dataset observation and visualization

We can mention to some below point as a summary of our Dataset observation and relation and corrolation between the columns:

The more the year increases, the more car prices increase. From this we can easily conclude that the year has a positive correlation with the price tag, which we can find in the previous heatmap .

Most cars in the data use a manual transmission, the price of which is below 30000. Also, cars with a semi-automatic transmission are more expensive than other models. Also, most cars in this dataset have engines with displacement between 1 and 2.5 liters.

According to this data, most of the cars have driven less than 75000 miles and the cars with manual transmission have driven the most. Also, the cars with the high price tag have driven less than other cars.

7 preparing Data for regression models

After cleaning and exploration data analysis, the next step is to prepare the data for input into a regression model.

First, we divide the data set into two parts (numeric data and categorical data).

We need to convert all categorical variables into numerical values by **OnehotEncoder** or **LabelEncoder**. We also need to standardize the predictors with **standardScaler** or **MinMaxScaler**, which is essential for regularized regression.

Finally, we merge both data back together. Now we have a suitable data set to use in the next steps.

[38]:
```
df = cars_edit.copy()
df_num = df.drop(["model","transmission","fuelType"], axis = 1 )
df_num
```

[38]:
```
      year  price  mileage  tax   mpg  engineSize
0     2017   7999    17307  145  58.9         1.2
1     2016  14499    25233  235  43.5         2.0
2     2016  11399    37877   30  61.7         1.7
3     2016   6499    23789   20  60.1         1.0
4     2015  10199    33177  160  51.4         2.0
…     …      …         …    …    …            …
4769  2016   8680    25906    0  78.4         1.6
4770  2015   7830    59508   30  65.7         1.7
4771  2017   6830    13810   20  60.1         1.0
4772  2018  13994    23313  145  44.8         1.6
4773  2016  15999    11472  125  57.6         1.7

[4774 rows x 6 columns]
```

[39]:
```
df_num.describe()
```

[39]:
```
              year         price        mileage          tax          mpg  \
count  4774.000000   4774.000000    4774.000000  4774.000000  4774.000000
mean   2017.092166  12710.465438   21658.914537   121.187683    53.837956
std       1.921323   5865.948565   17618.489657    58.135472    12.740499
min    2000.000000   1200.000000       1.000000     0.000000     1.100000
25%    2016.000000   8000.000000    8542.500000   125.000000    44.800000
50%    2017.000000  11990.500000   17627.000000   145.000000    55.400000
75%    2018.000000  15694.250000   31067.500000   145.000000    60.100000
max    2020.000000  43995.000000  138000.000000   555.000000   256.800000

        engineSize
count  4774.000000
mean      1.460285
std       0.401858
min       0.000000
25%       1.200000
50%       1.600000
75%       1.700000
max       2.900000
```

in this section, we can use both the LabelEncoder() and OneHotEncoder() functions to convert all categorical variables to numeric values.

I have written both encoder functions below, but due to the large number of models (16 models), I prefer OneHotEncoder to encode the categorical columns.

[40]:
```
#encoder = LabelEncoder()
#df["model"] = encoder.fit_transform(df["model"])
#df["transmission"] = encoder.fit_transform(df["transmission"])
#df["fuelType"] = encoder.fit_transform(df["fuelType"])
#df_clean = df.copy()
```

[41]:
```
df_onehot = OneHotEncoder(sparse= False)
df_onehot_encoded = df_onehot.
 ↪fit_transform(df[["model","transmission","fuelType"]])
column_name = df_onehot.get_feature_names(["model","transmission","fuelType"])
df_onehot_final = pd.DataFrame(df_onehot_encoded, columns= column_name)
df_onehot_final.head()
```

[41]:
```
   model_ Accent  model_ Amica  model_ Getz  model_ I10  model_ I20  \
0            0.0           0.0          0.0         0.0         1.0
1            0.0           0.0          0.0         0.0         0.0
2            0.0           0.0          0.0         0.0         0.0
3            0.0           0.0          0.0         1.0         0.0
4            0.0           0.0          0.0         0.0         0.0

   model_ I30  model_ I40  model_ I800  model_ IX20  model_ IX35  …  \
0         0.0         0.0          0.0          0.0          0.0  …
1         0.0         0.0          0.0          0.0          0.0  …
2         0.0         0.0          0.0          0.0          0.0  …
3         0.0         0.0          0.0          0.0          0.0  …
4         0.0         0.0          0.0          0.0          1.0  …

   model_ Tucson  model_ Veloster  transmission_Automatic  \
0            0.0              0.0                     0.0
1            1.0              0.0                     1.0
2            1.0              0.0                     0.0
3            0.0              0.0                     0.0
4            0.0              0.0                     0.0

   transmission_Manual  transmission_Other  transmission_Semi-Auto  \
0                  1.0                 0.0                     0.0
1                  0.0                 0.0                     0.0
2                  1.0                 0.0                     0.0
3                  1.0                 0.0                     0.0
4                  1.0                 0.0                     0.0

   fuelType_Diesel  fuelType_Hybrid  fuelType_Other  fuelType_Petrol
0              0.0              0.0             0.0              1.0
1              1.0              0.0             0.0              0.0
```

```
2              1.0              0.0             0.0              0.0
3              0.0              0.0             0.0              1.0
4              1.0              0.0             0.0              0.0

[5 rows x 24 columns]
```

We can do this by using pd.get_dummies(). This function encodes the labels just as OneHotEncoder does.

df_dummies= pd.get_dummies(df)

Now we merge two data into one clean data:

[42]:
```
df_clean = pd.concat([df_num,df_onehot_final], axis=1)
df_clean
```

[42]:
```
      year  price  mileage  tax   mpg  engineSize  model_ Accent  \
0     2017   7999    17307  145  58.9         1.2            0.0
1     2016  14499    25233  235  43.5         2.0            0.0
2     2016  11399    37877   30  61.7         1.7            0.0
3     2016   6499    23789   20  60.1         1.0            0.0
4     2015  10199    33177  160  51.4         2.0            0.0
…     …     …        …   …    …           …              …
4769  2016   8680    25906    0  78.4         1.6            0.0
4770  2015   7830    59508   30  65.7         1.7            0.0
4771  2017   6830    13810   20  60.1         1.0            0.0
4772  2018  13994    23313  145  44.8         1.6            0.0
4773  2016  15999    11472  125  57.6         1.7            0.0

      model_ Amica  model_ Getz  model_ I10  …  model_ Tucson  \
0              0.0          0.0         0.0  …            0.0
1              0.0          0.0         0.0  …            1.0
2              0.0          0.0         0.0  …            1.0
3              0.0          0.0         1.0  …            0.0
4              0.0          0.0         0.0  …            0.0
…              …            …          …  …            …
4769           0.0          0.0         0.0  …            0.0
4770           0.0          0.0         0.0  …            0.0
4771           0.0          0.0         1.0  …            0.0
4772           0.0          0.0         0.0  …            1.0
4773           0.0          0.0         0.0  …            1.0

      model_ Veloster  transmission_Automatic  transmission_Manual  \
0                 0.0                     0.0                  1.0
1                 0.0                     1.0                  0.0
2                 0.0                     0.0                  1.0
3                 0.0                     0.0                  1.0
4                 0.0                     0.0                  1.0
```

```
…                   …                       …                    …
4769              0.0                     0.0                  1.0
4770              0.0                     0.0                  1.0
4771              0.0                     0.0                  1.0
4772              0.0                     0.0                  1.0
4773              0.0                     1.0                  0.0

      transmission_Other  transmission_Semi-Auto  fuelType_Diesel  \
0                    0.0                     0.0              0.0
1                    0.0                     0.0              1.0
2                    0.0                     0.0              1.0
3                    0.0                     0.0              0.0
4                    0.0                     0.0              1.0
…                      …                       …                …
4769                 0.0                     0.0              1.0
4770                 0.0                     0.0              1.0
4771                 0.0                     0.0              0.0
4772                 0.0                     0.0              0.0
4773                 0.0                     0.0              1.0

      fuelType_Hybrid  fuelType_Other  fuelType_Petrol
0                 0.0             0.0              1.0
1                 0.0             0.0              0.0
2                 0.0             0.0              0.0
3                 0.0             0.0              1.0
4                 0.0             0.0              0.0
…                   …               …                …
4769              0.0             0.0              0.0
4770              0.0             0.0              0.0
4771              0.0             0.0              1.0
4772              0.0             0.0              1.0
4773              0.0             0.0              0.0

[4774 rows x 30 columns]
```

[43]:
```
df_clean.describe()
```

[43]:
```
              year         price        mileage          tax          mpg  \
count  4774.000000   4774.000000    4774.000000  4774.000000  4774.000000
mean   2017.092166  12710.465438   21658.914537   121.187683    53.837956
std       1.921323   5865.948565   17618.489657    58.135472    12.740499
min    2000.000000   1200.000000       1.000000     0.000000     1.100000
25%    2016.000000   8000.000000    8542.500000   125.000000    44.800000
50%    2017.000000  11990.500000   17627.000000   145.000000    55.400000
75%    2018.000000  15694.250000   31067.500000   145.000000    60.100000
max    2020.000000  43995.000000  138000.000000   555.000000   256.800000
```

```
       engineSize  model_ Accent  model_ Amica  model_ Getz   model_ I10  \
count  4774.000000    4774.000000   4774.000000  4774.000000  4774.000000
mean      1.460285       0.000209      0.000209     0.001257     0.222245
std       0.401858       0.014473      0.014473     0.035433     0.415799
min       0.000000       0.000000      0.000000     0.000000     0.000000
25%       1.200000       0.000000      0.000000     0.000000     0.000000
50%       1.600000       0.000000      0.000000     0.000000     0.000000
75%       1.700000       0.000000      0.000000     0.000000     0.000000
max       2.900000       1.000000      1.000000     1.000000     1.000000

       …  model_ Tucson  model_ Veloster  transmission_Automatic  \
count  …    4774.000000      4774.000000             4774.000000
mean   …       0.268119         0.000628                0.137830
std    …       0.443026         0.025063                0.344757
min    …       0.000000         0.000000                0.000000
25%    …       0.000000         0.000000                0.000000
50%    …       0.000000         0.000000                0.000000
75%    …       1.000000         0.000000                0.000000
max    …       1.000000         1.000000                1.000000

       transmission_Manual  transmission_Other  transmission_Semi-Auto  \
count          4774.000000         4774.000000             4774.000000
mean              0.742773            0.000419                0.118978
std               0.437151            0.020466                0.323796
min               0.000000            0.000000                0.000000
25%               0.000000            0.000000                0.000000
50%               1.000000            0.000000                0.000000
75%               1.000000            0.000000                0.000000
max               1.000000            1.000000                1.000000

       fuelType_Diesel  fuelType_Hybrid  fuelType_Other  fuelType_Petrol
count      4774.000000      4774.000000     4774.000000      4774.000000
mean          0.334101         0.071219        0.000209         0.594470
std           0.471725         0.257217        0.014473         0.491046
min           0.000000         0.000000        0.000000         0.000000
25%           0.000000         0.000000        0.000000         0.000000
50%           0.000000         0.000000        0.000000         1.000000
75%           1.000000         0.000000        0.000000         1.000000
max           1.000000         1.000000        1.000000         1.000000

[8 rows x 30 columns]
```

For scaling the dataset as the last step, we have two scaling functions (StandardScaler and MinMaxScaler). I have incorporated both functions into the scaling of the dataset.

After the previous description, I noticed that the distance between the minimum and maximum values in some columns is larger than usual. Therefore, I decided to scale the data set with MinMaxScaler. As you can see, after scaling the data with MinMaxScaler, our variables are in the range between 0 and 1.

[44]:
```
#scaler = StandardScaler()
scaler = MinMaxScaler()

data_scaled = scaler.fit_transform(df_clean)
data_scaled = pd.DataFrame(data_scaled, columns=df_clean.columns)


df_cars = data_scaled.copy()
df_cars
```

[44]:
```
      year     price   mileage       tax       mpg  engineSize  model_ Accent  \
0     0.85  0.158874  0.125407  0.261261  0.226046    0.413793            0.0
1     0.80  0.310761  0.182842  0.423423  0.165819    0.689655            0.0
2     0.80  0.238322  0.274466  0.054054  0.236996    0.586207            0.0
3     0.80  0.123823  0.172378  0.036036  0.230739    0.344828            0.0
4     0.75  0.210282  0.240408  0.288288  0.196715    0.689655            0.0
…      …         …         …         …         …           …              …
4769  0.80  0.174787  0.187719  0.000000  0.302307    0.551724            0.0
4770  0.75  0.154925  0.431213  0.054054  0.252640    0.586207            0.0
4771  0.85  0.131557  0.100066  0.036036  0.230739    0.344828            0.0
4772  0.90  0.298960  0.168929  0.261261  0.170903    0.551724            0.0
4773  0.80  0.345811  0.083124  0.225225  0.220962    0.586207            0.0

      model_ Amica  model_ Getz  model_ I10  …  model_ Tucson  \
0              0.0          0.0         0.0  …            0.0
1              0.0          0.0         0.0  …            1.0
2              0.0          0.0         0.0  …            1.0
3              0.0          0.0         1.0  …            0.0
4              0.0          0.0         0.0  …            0.0
…                …            …         …  …              …
4769           0.0          0.0         0.0  …            0.0
4770           0.0          0.0         0.0  …            0.0
4771           0.0          0.0         1.0  …            0.0
4772           0.0          0.0         0.0  …            1.0
4773           0.0          0.0         0.0  …            1.0

      model_ Veloster  transmission_Automatic  transmission_Manual  \
0                 0.0                     0.0                  1.0
1                 0.0                     1.0                  0.0
2                 0.0                     0.0                  1.0
3                 0.0                     0.0                  1.0
4                 0.0                     0.0                  1.0
…                   …                       …                    …
4769              0.0                     0.0                  1.0
4770              0.0                     0.0                  1.0
```

```
4771            0.0                     0.0                   1.0
4772            0.0                     0.0                   1.0
4773            0.0                     1.0                   0.0

      transmission_Other  transmission_Semi-Auto  fuelType_Diesel  \
0                    0.0                     0.0              0.0
1                    0.0                     0.0              1.0
2                    0.0                     0.0              1.0
3                    0.0                     0.0              0.0
4                    0.0                     0.0              1.0
…                 …                     …              …
4769                 0.0                     0.0              1.0
4770                 0.0                     0.0              1.0
4771                 0.0                     0.0              0.0
4772                 0.0                     0.0              0.0
4773                 0.0                     0.0              1.0

      fuelType_Hybrid  fuelType_Other  fuelType_Petrol
0                 0.0             0.0              1.0
1                 0.0             0.0              0.0
2                 0.0             0.0              0.0
3                 0.0             0.0              1.0
4                 0.0             0.0              0.0
…              …            …             …
4769              0.0             0.0              0.0
4770              0.0             0.0              0.0
4771              0.0             0.0              1.0
4772              0.0             0.0              1.0
4773              0.0             0.0              0.0

[4774 rows x 30 columns]
```

[45]:
```
df_cars.describe()
```

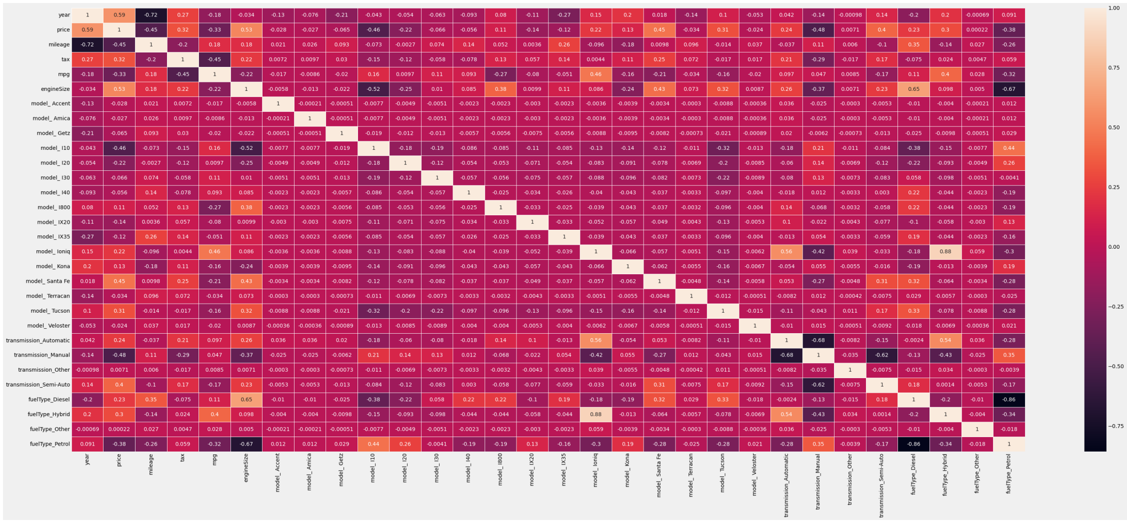
[45]:
```
              year        price      mileage          tax          mpg  \
count  4774.000000  4774.000000  4774.000000  4774.000000  4774.000000
mean      0.854608     0.268968     0.156943     0.218356     0.206249
std       0.096066     0.137071     0.127671     0.104749     0.049826
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.800000     0.158897     0.061895     0.225225     0.170903
50%       0.850000     0.252144     0.127726     0.261261     0.212358
75%       0.900000     0.338690     0.225121     0.261261     0.230739
max       1.000000     1.000000     1.000000     1.000000     1.000000

        engineSize  model_ Accent  model_ Amica  model_ Getz  model_ I10  \
count  4774.000000    4774.000000   4774.000000  4774.000000  4774.000000
mean      0.503547       0.000209      0.000209     0.001257     0.222245
```

```
std      0.138572    0.014473    0.014473    0.035433    0.415799
min      0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.413793    0.000000    0.000000    0.000000    0.000000
50%      0.551724    0.000000    0.000000    0.000000    0.000000
75%      0.586207    0.000000    0.000000    0.000000    0.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000

      …  model_ Tucson  model_ Veloster  transmission_Automatic  \
count …    4774.000000      4774.000000             4774.000000
mean  …       0.268119         0.000628                0.137830
std   …       0.443026         0.025063                0.344757
min   …       0.000000         0.000000                0.000000
25%   …       0.000000         0.000000                0.000000
50%   …       0.000000         0.000000                0.000000
75%   …       1.000000         0.000000                0.000000
max   …       1.000000         1.000000                1.000000

       transmission_Manual  transmission_Other  transmission_Semi-Auto  \
count          4774.000000         4774.000000             4774.000000
mean              0.742773            0.000419                0.118978
std               0.437151            0.020466                0.323796
min               0.000000            0.000000                0.000000
25%               0.000000            0.000000                0.000000
50%               1.000000            0.000000                0.000000
75%               1.000000            0.000000                0.000000
max               1.000000            1.000000                1.000000

       fuelType_Diesel  fuelType_Hybrid  fuelType_Other  fuelType_Petrol
count      4774.000000      4774.000000     4774.000000      4774.000000
mean          0.334101         0.071219        0.000209         0.594470
std           0.471725         0.257217        0.014473         0.491046
min           0.000000         0.000000        0.000000         0.000000
25%           0.000000         0.000000        0.000000         0.000000
50%           0.000000         0.000000        0.000000         1.000000
75%           1.000000         0.000000        0.000000         1.000000
max           1.000000         1.000000        1.000000         1.000000

[8 rows x 30 columns]
```

This is a correlation heatmap showing the correlations between all columns after coding and scaling the data. In the next steps, I will show which columns are more strongly correlated with price.

[95]:
```
sns.heatmap(df_cars.corr(), annot=True, linewidths=1)
```

[95]:
```
<AxesSubplot:>
```



select the X and Y to use them in the next steps

Our target in this project is the price column, so we consider it as Y and other columns are automatically considered X. But in the next step, when I compare the regression algorithms for predicting the price, I will try to check them with two different X (first I will try with the current X values and then I will use another X value that contains the columns that are less correlated with the price according to the heatmap)

[47]:
```
x = df_cars.drop(columns= 'price')
y= df_cars.price
```

According to the previous heatmap, we noticed that after coding the data, some new columns have a relationship with the price that we did not know in the first heatmap. With the following code we can find out which columns have a stronger connection with the price

[48]:
```
k=4
selector = SelectKBest(f_regression,k=k)
selector.fit(x,y)
best_feats = selector.get_support(indices=True)
x_best =x.iloc[:,best_feats]
x_best.head()
```

[48]:

	year	engineSize	model_ I10	transmission_Manual
0	0.85	0.413793	0.0	1.0
1	0.80	0.689655	0.0	0.0
2	0.80	0.586207	0.0	1.0
3	0.80	0.344828	1.0	1.0
4	0.75	0.689655	0.0	1.0

[49]:
```
y.head()
```

[49]:
```
0    0.158874
1    0.310761
2    0.238322
3    0.123823
4    0.210282
Name: price, dtype: float64
```

[50]:
```
x.head()
```

[50]:
```
   year   mileage       tax       mpg  engineSize  model_ Accent  \
0  0.85  0.125407  0.261261  0.226046    0.413793            0.0
1  0.80  0.182842  0.423423  0.165819    0.689655            0.0
2  0.80  0.274466  0.054054  0.236996    0.586207            0.0
3  0.80  0.172378  0.036036  0.230739    0.344828            0.0
4  0.75  0.240408  0.288288  0.196715    0.689655            0.0

   model_ Amica  model_ Getz  model_ I10  model_ I20  …  model_ Tucson  \
0           0.0          0.0         0.0         1.0  …            0.0
1           0.0          0.0         0.0         0.0  …            1.0
2           0.0          0.0         0.0         0.0  …            1.0
3           0.0          0.0         1.0         0.0  …            0.0
4           0.0          0.0         0.0         0.0  …            0.0

   model_ Veloster  transmission_Automatic  transmission_Manual  \
0              0.0                     0.0                  1.0
1              0.0                     1.0                  0.0
2              0.0                     0.0                  1.0
3              0.0                     0.0                  1.0
4              0.0                     0.0                  1.0

   transmission_Other  transmission_Semi-Auto  fuelType_Diesel  \
0                 0.0                     0.0              0.0
1                 0.0                     0.0              1.0
2                 0.0                     0.0              1.0
3                 0.0                     0.0              0.0
4                 0.0                     0.0              1.0

   fuelType_Hybrid  fuelType_Other  fuelType_Petrol
0              0.0             0.0              1.0
1              0.0             0.0              0.0
2              0.0             0.0              0.0
3              0.0             0.0              1.0
4              0.0             0.0              0.0

[5 rows x 29 columns]
```

finally, we split our data set into train set and test set at the final step of this part. >Train/Test is a method to measure the accuracy of your model.so we split the the data set into two sets: a training set and a testing set. 70% for training, and 30% for testing.with train set we will train the model then we will test our model with the test values.

Random state consider as 42 in this project

[51]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,␣
 ↪random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3341, 29)
(1433, 29)
(3341,)
(1433,)
```

8 Fit different regression models and compare their performance at predicting the price of the cars

In this part, we started by training different 3 ML regression models **(Linier Regression, Desicion Tree and Random Forest)** to find out which algorithm provides better accuracy for our price prediction.

First, we start by training the models by evaluating the training data collected in the previous step. Then, I show the first **20** predicted values in each model and compare them to the actual values which we previously trained the model with them .The graph I created after predicting the values shows the extent to which the model was successful in predicting the price, because it shows the extent to which the model was successful in fitting the actual values.

then,i explian the average squared difference between the estimated values and the actual value by using**Mean Square Error(MSE)** and **Root Mean Squared Error (RMSE)** - The MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.

then i will show measure that represents the proportion of the variance for a dependent variable(price) according to other variables in aeach regression model**R2** which show the prediction efficency for each model(The R2 result is between 0 and 1, showing that the models whose result is closer to 1 have better efficiency in predicting the price for our project).

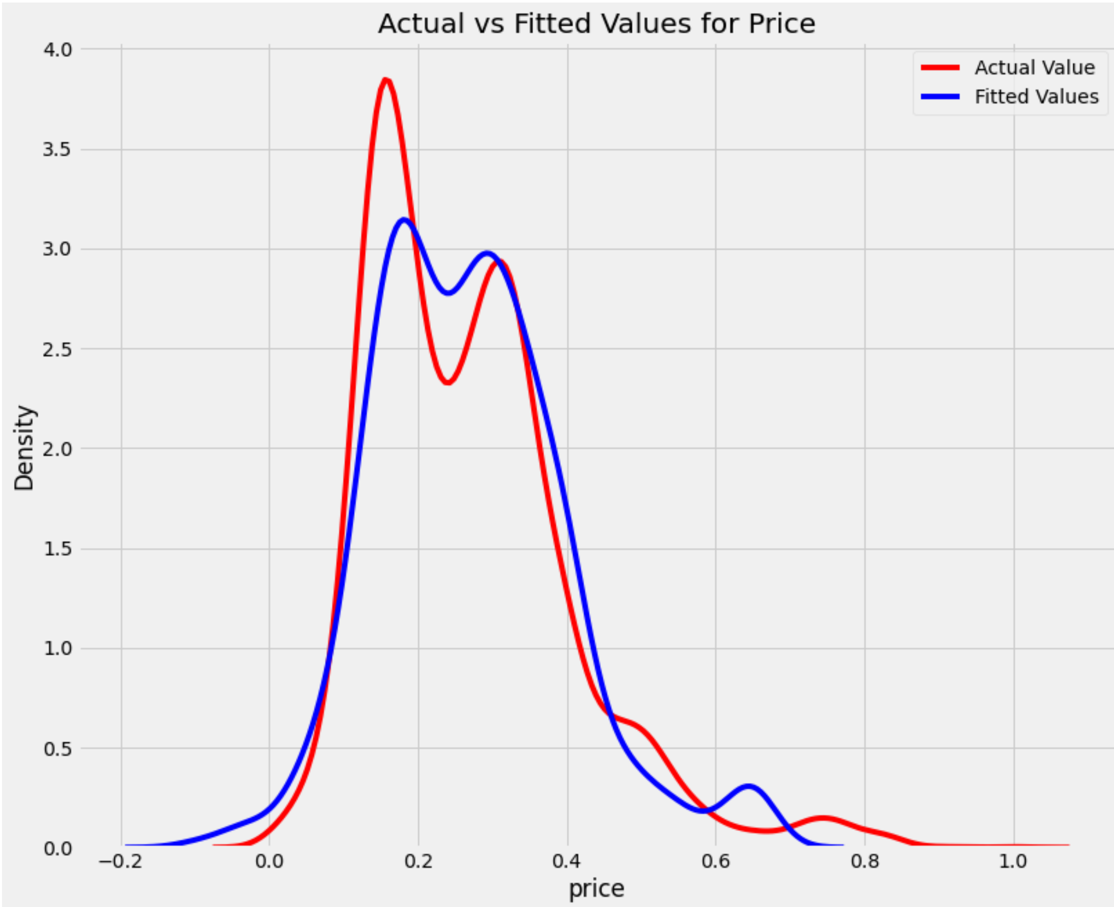
Evaluating the **MSE** and **R2** error by validating the data is the next step of my project, where I use the **cross-validation method**. In this method, the data is divided into 7 equal parts depending on the number of CV chosen (in this project there are 7) and each part is evaluated separately. The results show the more accurate R2 and MSE value, which indicates which model is better for price prediction.

Finally, I have defined the **learning curves function** and use it for each model to plot the trend of training and validation errors as a function of increasing number of samples in our data. This plot allows us to see when the errors become stable,

9 Linear Regression Model

[52]:
```
linModel = LinearRegression()
linModel.fit(x_train.values, y_train.values)
y_predLin = linModel.predict(x_test.values)
print("actual value", "             predicted value")
i = 1
while i < 20:
    print(y_test.values[i],"      ",y_predLin[i])
    i += 1

plt.figure(figsize=(12, 10))
ax = sns.distplot(y, hist=False, color="r", label="Actual Value")
sns.distplot(y_predLin, hist=False, color="b", label="Fitted Values" , ax=ax)
plt.title('Actual vs Fitted Values for Price')
plt.legend()
plt.show()
```

```
actual value             predicted value
0.23437317443626593        0.287353515625
0.06998481130973246        0.046875
0.415819605094053        0.39453125
0.3726837247341979        0.37060546875
0.2754761070218483        0.284912109375
0.25236593059936907        0.309326171875
0.18226428321065544        0.21044921875
0.22895198037153874        0.22412109375
0.23016707559294308        0.260986328125
0.7195934104451455        0.652587890625
0.18226428321065544        0.2109375
0.35494800794485337        0.403076171875
0.2983993457179577        0.322509765625
0.19850449818904076        0.203125
0.03502745647856058        -0.076904296875
0.12379950928846829        0.15234375
0.12641663745764692        0.168701171875
0.5069517466993808        0.389404296875
0.4392802897534759        0.447021484375
```



[53]:
```
mseLin = mean_squared_error(y_test.values, y_predLin)
rmselin = np.sqrt(mean_squared_error(y_test.values, y_predLin))
print('The Mean Squared Error = {0:.4f}'.format(mseLin))
print('Root Mean Squared Error :{0:.4f}'.format(rmselin))
```

```
The Mean Squared Error = 0.0026
Root Mean Squared Error :0.0505
```

[54]:
```
r2_Lin = r2_score(y_test.values, y_predLin)
print('Goodness of Fit: {0:.2f}'.format(r2_Lin))
```

```
Goodness of Fit: 0.87
```

[55]:
```
plt.figure(figsize = (10,6))
plt.scatter(y_test, y_predLin, label='Linear Regression')
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
plt.legend(loc='upper left');
plt.grid()
```

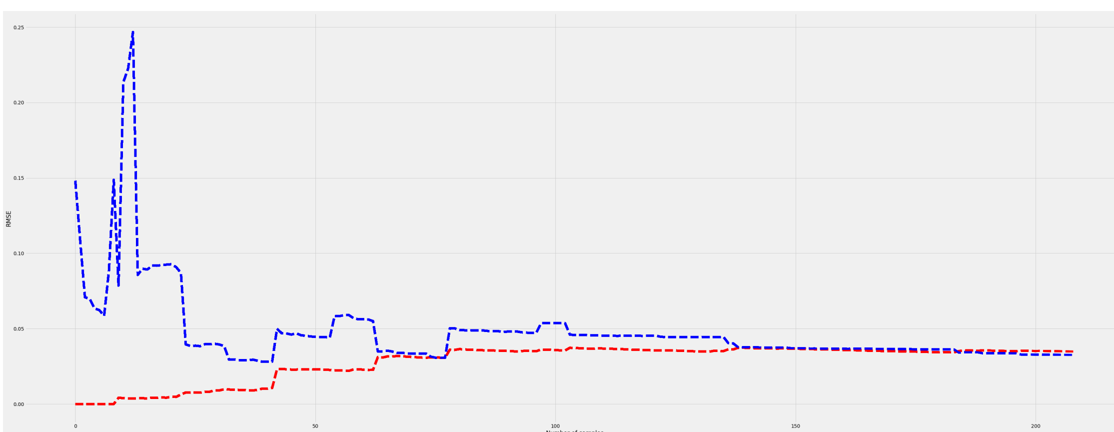
This figure shows predicted and actual prices. As we see, some prices predicted have negative values, which is not possible. In Linear regression some of feature have negative coeefficient values and cause these negative preicted values.

[56]:
```
#cross validation
scores = cross_val_score(linModel, x_train, y_train,␣
 ↪scoring="neg_mean_squared_error", cv=7)
linModel_rmse_scores = np.sqrt(-scores)

linModel_r2_scores = cross_val_score(linModel, x_train, y_train, scoring="r2",␣
 ↪cv=7)



def show_scores(scores, model):
    print("      ",model,"    ")
    print("scores:",scores)
    print("scores_mean :", scores.mean())


show_scores(linModel_rmse_scores,linModel)
print("-"*10)
show_scores(linModel_r2_scores,linModel)
```

```
      LinearRegression()
scores: [4.44449231e-02 1.79290800e+09 4.82397383e-02 2.26668832e+10
 4.80201187e-02 5.07742344e-02 4.40284600e-02]
```

```
scores_mean : 3494255888.171599
----------
       LinearRegression()
scores: [ 8.92175209e-01 -2.05100961e+20  8.76011613e-01 -2.73377745e+22
  8.67825460e-01  8.67853878e-01  9.01092071e-01]
scores_mean : -3.934696497436076e+21
```

[57]:
```python
def plot_learning_curves (model, X, y):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
    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.rcParams["figure.figsize"] = (50, 20)
        plt.plot(np.sqrt(train_errors), 'r--', linewidth=6, label='train')
        plt.plot(np.sqrt(val_errors), 'b--', linewidth=6, label='val')
        plt.ylabel('RMSE')
        plt.xlabel('Number of samples')




plot_learning_curves(linModel, x[:300], y[:300])
```

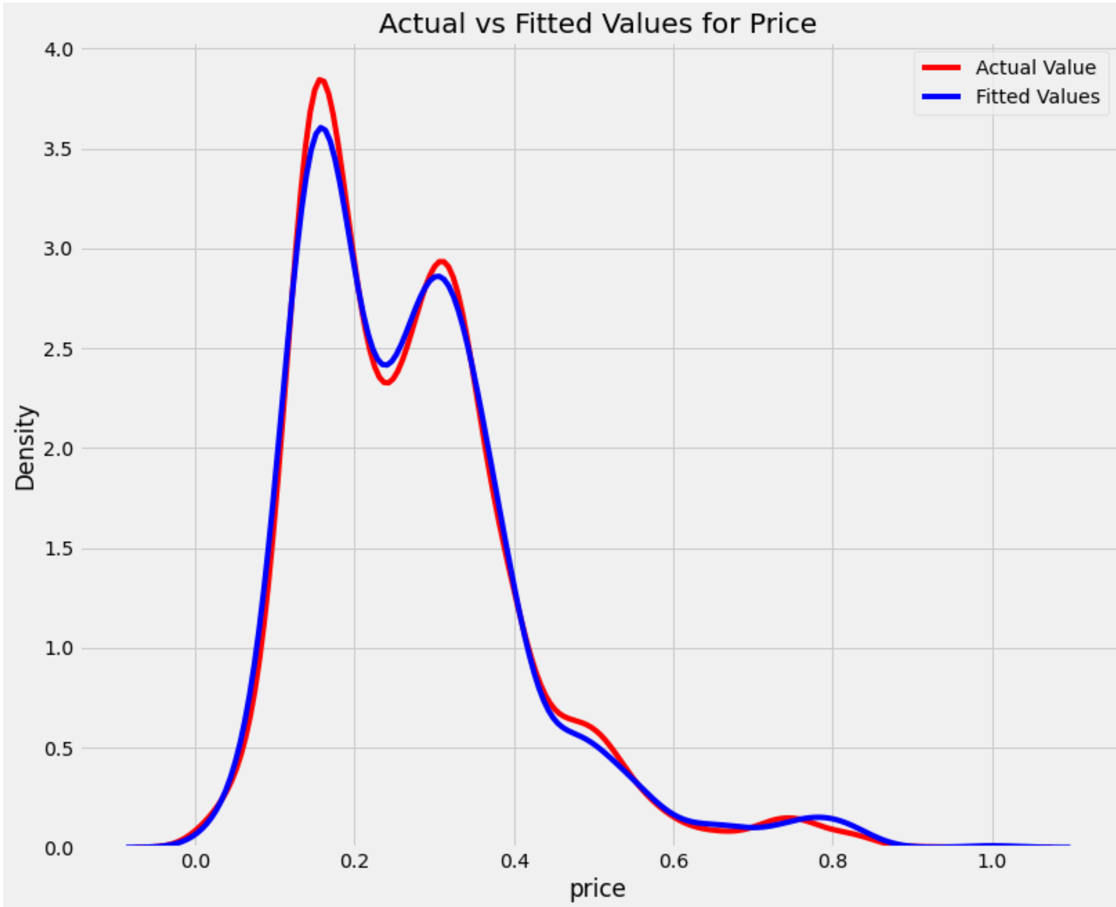


So, learning curves tell us that the RMSE stabilizes as long as the number of samples grow in volume.

10 DecisionTree Model

[58]:
```
DecisionTree_Model = DecisionTreeRegressor()
DecisionTree_Model.fit(x_train.values, y_train.values)
y_dt_pred = DecisionTree_Model.predict(x_test.values)
print("actual value", "             predicted value")
i = 1
while i < 20:
    print(y_test.values[i],"       ",y_dt_pred[i])
    i += 1

plt.figure(figsize=(12, 10))
ax = sns.distplot(y, hist=False, color="r", label="Actual Value")
sns.distplot(y_dt_pred, hist=False, color="b", label="Fitted Values" , ax=ax)
plt.title('Actual vs Fitted Values for Price')
plt.legend()
plt.show()
```

```
actual value              predicted value
0.23437317443626593          0.22187171398527866
0.06998481130973246          0.10725552050473187
0.415819605094053          0.37466993807687815
0.3726837247341979          0.3692020095805585
0.2754761070218483          0.26103516765977336
0.25236593059936907          0.29884332281808623
0.18226428321065544          0.17053394088094403
0.22895198037153874          0.2522490945203879
0.23016707559294308          0.2523191961677766
0.7195934104451455          0.8364294894263349
0.18226428321065544          0.17455310199789695
0.35494800794485337          0.3714452622969973
0.2983993457179577          0.28694940997780116
0.19850449818904076          0.20551466292791212
0.03502745647856058          0.03960743077462321
0.12379950928846829          0.12379950928846829
0.12641663745764692          0.13155742493281924
0.5069517466993808          0.5326556840752424
0.4392802897534759          0.48592125248276663
```



[59]:
```
mse_dt = mean_squared_error(y_test.values, y_dt_pred)
rmse_dt = np.sqrt(mean_squared_error(y_test.values, y_dt_pred))
print('The Mean Squared Error = {0:.4f}'.format(mse_dt))
print('Root Mean Squared Error :{0:.4f}'.format(rmse_dt))
```

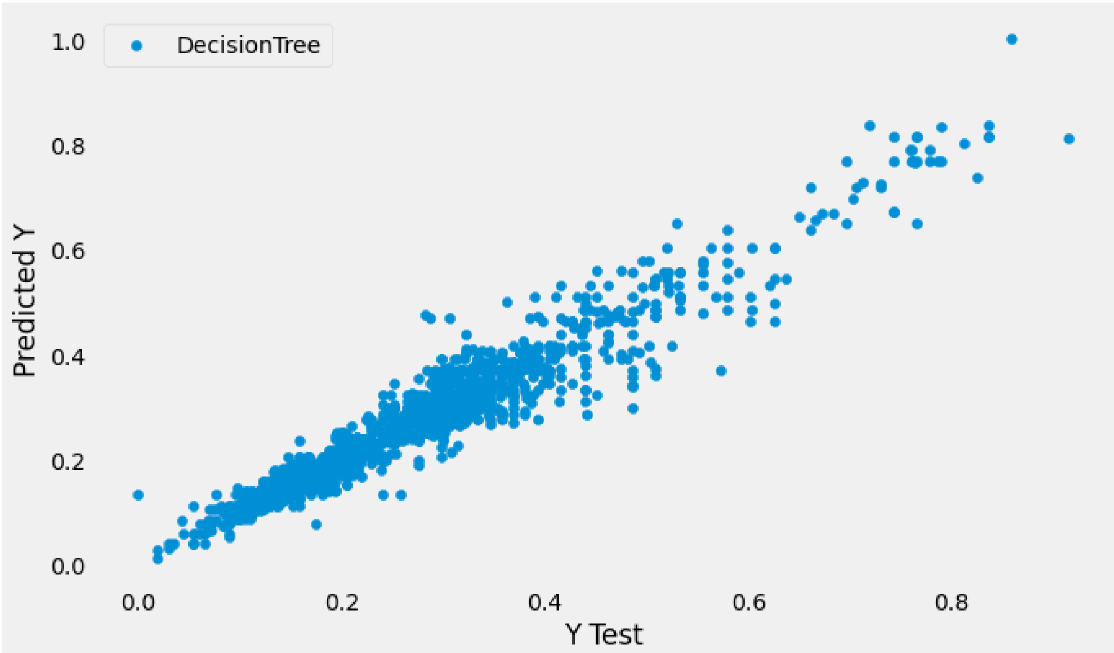
```
The Mean Squared Error = 0.0014
Root Mean Squared Error :0.0368
```

[60]:
```
r2_dt = r2_score(y_test.values, y_dt_pred)
print('Goodness of Fit: {0:.2f}'.format(r2_dt))
```

```
Goodness of Fit: 0.93
```

[61]:
```
plt.figure(figsize = (10,6))
plt.scatter(y_test, y_dt_pred, label='DecisionTree')
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
plt.legend(loc='upper left')
plt.legend()
```

```
plt.grid()
```



[62]:

```
scores = cross_val_score(DecisionTree_Model, x_train, y_train,
    scoring="neg_mean_squared_error", cv=10)
DecisionTree_rmse_scores = np.sqrt(-scores)

DecisionTree_r2_scores = cross_val_score(DecisionTree_Model, x_train, y_train,
    scoring="r2", cv=10)



def show_scores(scores, model):
    print("       ",model,"    ")
    print("scores:",scores)
    print("scores_mean", scores.mean())


show_scores(DecisionTree_rmse_scores,DecisionTree_Model)
print("-"*10)
show_scores(DecisionTree_r2_scores,DecisionTree_Model)
```

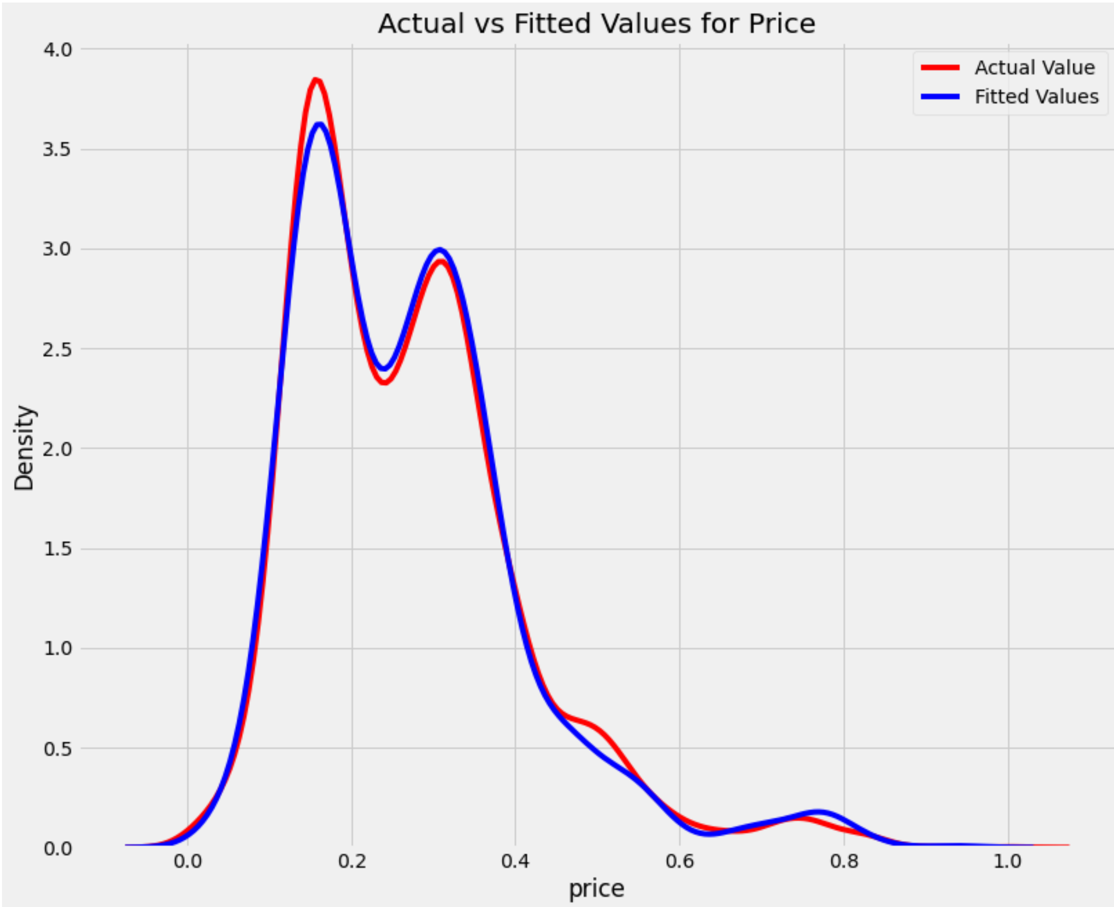
```
      DecisionTreeRegressor()
scores: [0.03448634 0.03705589 0.04201725 0.03826896 0.03531262 0.03523443
 0.03495813 0.03712369 0.03363404 0.03565915]
scores_mean 0.03637505203859618
----------
```

```
       DecisionTreeRegressor()
scores: [0.92959873 0.91954739 0.90204475 0.91987078 0.93179007 0.93084466
 0.93306296 0.93278463 0.93681262 0.93545713]
scores_mean 0.9271813706522553
```

11 Random Forest Model

[63]:
```
randomForestModel = RandomForestRegressor()
randomForestModel.fit(x_train.values, y_train.values)
y_rf_pred = randomForestModel.predict(x_test.values)
print("actual value", "predicted value")
i = 1
while i < 20:
    print(y_test.values[i],"       ",y_rf_pred[i])
    i += 1
plt.figure(figsize=(12, 10))
ax = sns.distplot(y, hist=False, color="r", label="Actual Value")
sns.distplot(y_rf_pred, hist=False, color="b", label="Fitted Values" , ax=ax)
plt.title('Actual vs Fitted Values for Price')
plt.legend()
plt.show()
```

```
actual value predicted value
0.23437317443626593         0.24179577053394102
0.06998481130973246         0.10933239864470147
0.415819605094053         0.38431592475756476
0.3726837247341979         0.3633746933052926
0.275476107021848З         0.26371281691786397
0.25236593059936907         0.3022902208201895
0.18226428321065544         0.1753599719593412
0.22895198037153874         0.23352167309265093
0.23016707559294308         0.2528344432760835
0.7195934104451455         0.761835494800795
0.18226428321065544         0.17928262647505566
0.35494800794485337         0.4206157261362307
0.2983993457179577         0.3181865482727734
0.19850449818904076         0.19443276083654631
0.03502745647856058         0.03760953382404486
0.12379950928846829         0.1312943100829539
0.12641663745764692         0.13071386844257504
0.5069517466993808         0.48528659890174075
0.4392802897534759         0.4442152120574835
```



[64]:
```
mse_rf = mean_squared_error(y_test.values, y_rf_pred)
rmse_rf = np.sqrt(mean_squared_error(y_test.values, y_rf_pred))
print('The Mean Squared Error = {0:.4f}'.format(mse_rf))
print('Root Mean Squared Error :{0:.4f}'.format(rmse_rf))
```

```
The Mean Squared Error = 0.0009
Root Mean Squared Error :0.0300
```

[65]:
```
r2_rf = r2_score(y_test.values, y_rf_pred)
print('Goodness of Fit: {0:.2f}'.format(r2_rf))
```

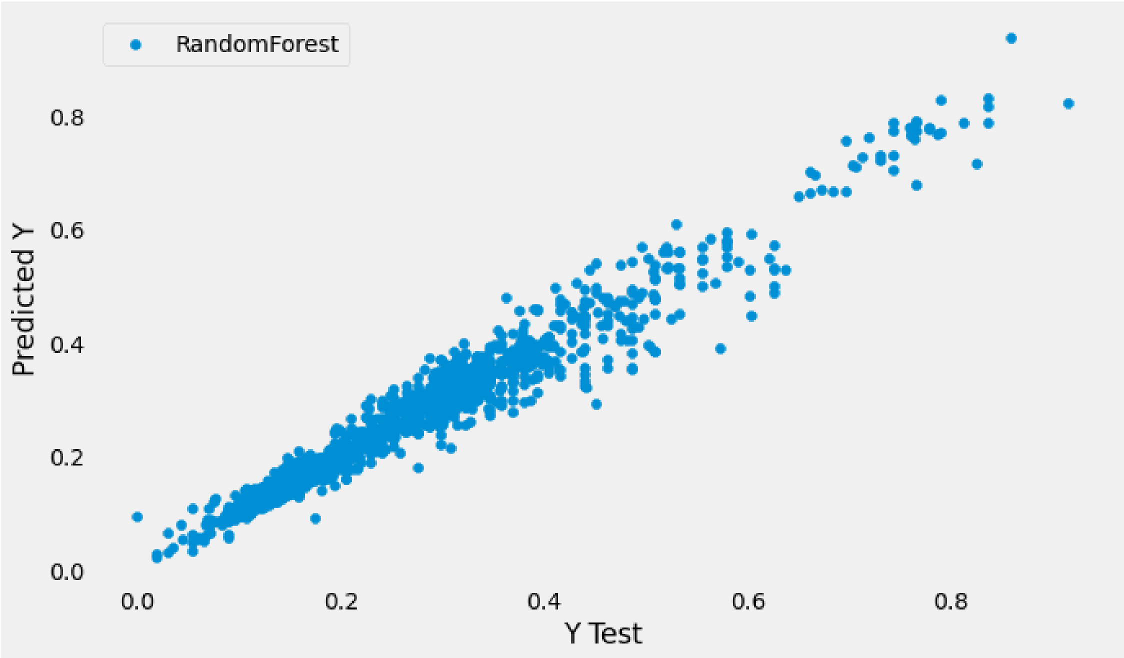
```
Goodness of Fit: 0.95
```

[66]:
```
scores_randomforrest = cross_val_score(randomForestModel, x_train, y_train,␣
↪scoring="neg_mean_squared_error", cv=10)
randomForestModel_rmse_scores = np.sqrt(-scores_randomforrest)

randomForestModel_r2_scores = cross_val_score(randomForestModel, x_train,␣
↪y_train, scoring="r2", cv=10)
```

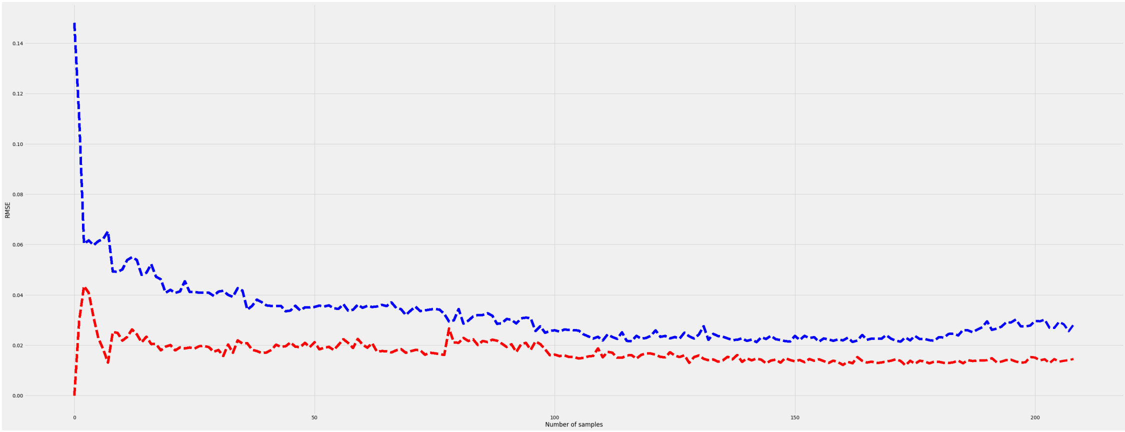
```
def show_scores(scores, model):
    print("      ",model,"    ")
    print("scores:",scores)
    print("scores_mean", scores.mean())


show_scores(randomForestModel_rmse_scores,randomForestModel)
print("-"*10)
show_scores(randomForestModel_r2_scores,randomForestModel)
```

```
       RandomForestRegressor()
scores: [0.03051997 0.02923194 0.03482468 0.0306957  0.0275152  0.02844263
 0.02926808 0.03029258 0.02828441 0.02739067]
scores_mean 0.029646587802086226
----------
       RandomForestRegressor()
scores: [0.94777848 0.94878058 0.93289733 0.94855573 0.9578231  0.95342142
 0.95335814 0.95548759 0.95551445 0.96204986]
scores_mean 0.9515666683785078
```

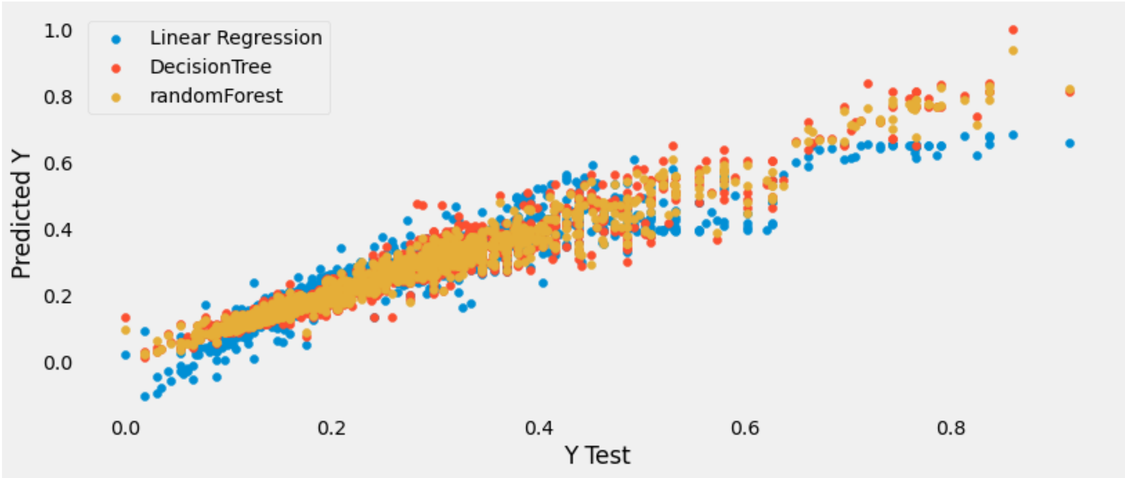
[67]:
```
plt.figure(figsize = (10,6))
plt.scatter(y_test, y_rf_pred, label='RandomForest')
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
plt.legend(loc='upper left');
plt.grid()
```



[68]:
```
plot_learning_curves(randomForestModel, x[:300], y[:300 ])
```



[69]:
```
plt.figure(figsize = (12,5))
plt.scatter(y_test, y_predLin, label='Linear Regression')
plt.scatter(y_test, y_dt_pred, label='DecisionTree')
plt.scatter(y_test, y_rf_pred, label='randomForest')
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
plt.legend(loc='upper left');
plt.grid()
```



The above plot shows the predicted values of the three models that show Randomforest model predicted values have a great positive corrolation with the actual values and have a less outlier values in comparison with other models result.

Also if we analyse the result below and compare it with this plot, we can see that the Random Forest model is better for predicting the price in this project, since it has a higher accuracy with the less errors ratio in compared to the other models.

[70]:
```
print("R2  score by using MultiLinear Regression:", r2_Lin)
print("R2  score by using Decision Tree         :", r2_dt)
print("R2  score by using Random Forest         :", r2_rf)
```

```
R2  score by using MultiLinear Regression: 0.871720933476039
R2  score by using Decision Tree         : 0.9319433950819141
R2  score by using Random Forest         : 0.9546236157474985
```

[71]:
```
print("RMSE  score by using MultiLinear Regression:", rmselin)
print("RMSE  score by using Decision Tree         :", rmse_dt)
print("RMSE  score by using Random Forest         :", rmse_rf)
```

```
RMSE  score by using MultiLinear Regression: 0.050519734420225976
RMSE  score by using Decision Tree         : 0.036797499101295324
RMSE  score by using Random Forest         : 0.030046791736597828
```

In this part, we repeat the previous step by changing the X values. In this part, we omit the columns with high correlation with price and consider the other columns as new X to continue our investigation and find out which model and columns set are better for predicting price in our project

[72]:
```
x1 = df_cars.drop(columns =["price","year","engineSize"])
y= df_cars.price
```

[73]:
```
k=4
selector = SelectKBest(f_regression,k=k)
selector.fit(x1,y)
best_feats = selector.get_support(indices=True)
x_best =x1.iloc[:,best_feats]
x_best.head()
```

[73]:

	mileage	model_ I10	model_ Santa Fe	transmission_Manual
0	0.125407	0.0	0.0	1.0
1	0.182842	0.0	0.0	0.0
2	0.274466	0.0	0.0	1.0
3	0.172378	1.0	0.0	1.0
4	0.240408	0.0	0.0	1.0

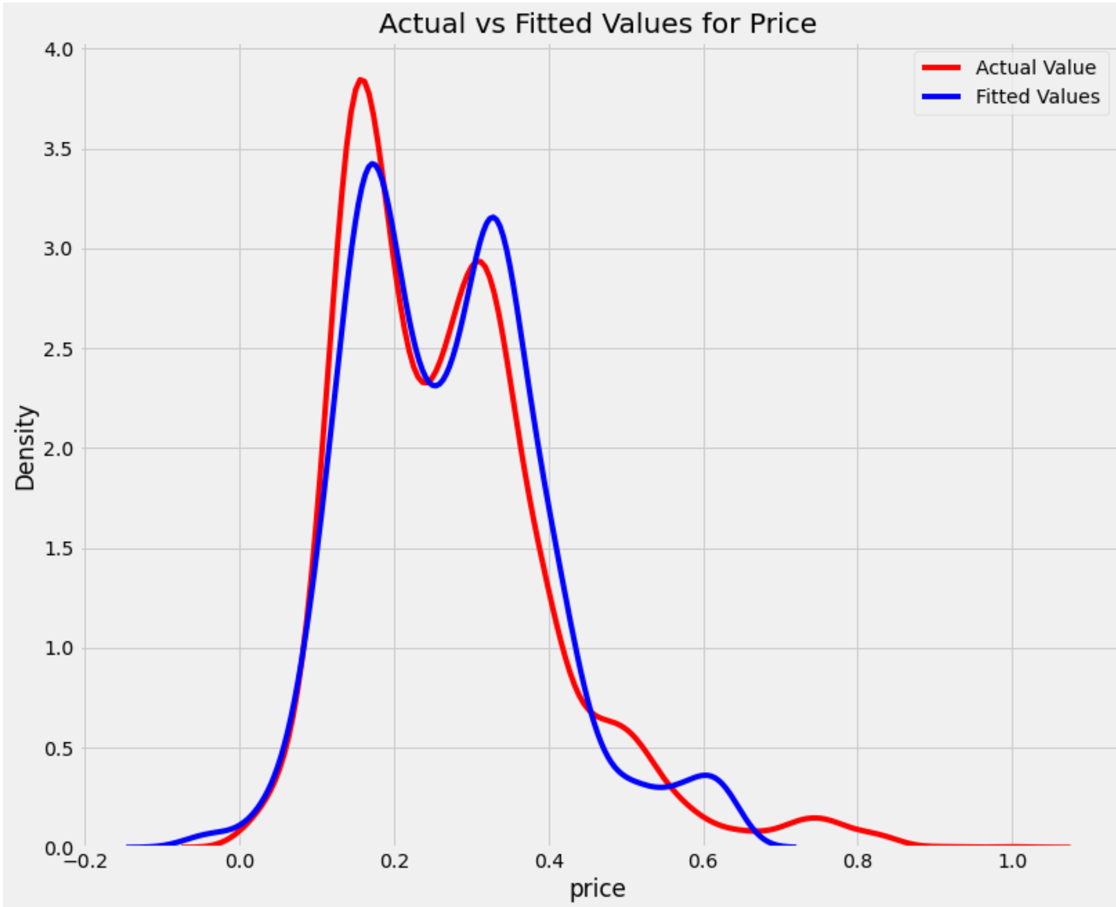
[74]:
```
x1_train, x1_test, y_train, y_test = train_test_split(x1, y, test_size=0.3,␣
 ↪random_state=42)
print(x1_train.shape)
print(x1_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3341, 27)
(1433, 27)
(3341,)
(1433,)
```

[75]:

```
linModel = LinearRegression()
linModel.fit(x1_train.values, y_train.values)
y_predLin = linModel.predict(x1_test.values)
print("actual value", "             predicted value")
i = 1
while i < 20:
    print(y_test.values[i],"      ",y_predLin[i])
    i += 1

plt.figure(figsize=(12, 10))
ax = sns.distplot(y, hist=False, color="r", label="Actual Value")
sns.distplot(y_predLin, hist=False, color="b", label="Fitted Values" , ax=ax)
plt.title('Actual vs Fitted Values for Price')
plt.legend()
plt.show()
```

```
actual value             predicted value
0.23437317443626593        0.33456761729811646
0.06998481130973246        0.13536609158596224
0.415819605094053        0.4031266696311403
0.3726837247341979        0.403450327315679
0.2754761070218483        0.2734425399120707
0.25236593059936907        0.24857643192393242
0.18226428321065544        0.2675371365468141
0.22895198037153874        0.24673410797831963
0.23016707559294308        0.2985614081685777
0.7195934104451455        0.6258589930528804
0.18226428321065544        0.17486586804529458
0.35494800794485337        0.39617395684036616
0.2983993457179577        0.3329695523042795
0.19850449818904076        0.19149288092856542
0.03502745647856058        -0.0039122414609583656
0.12379950928846829        0.13510189708568066
0.12641663745764692        0.16494621805429244
0.5069517466993808        0.3981674097691641
0.4392802897534759        0.44192501408940454
```



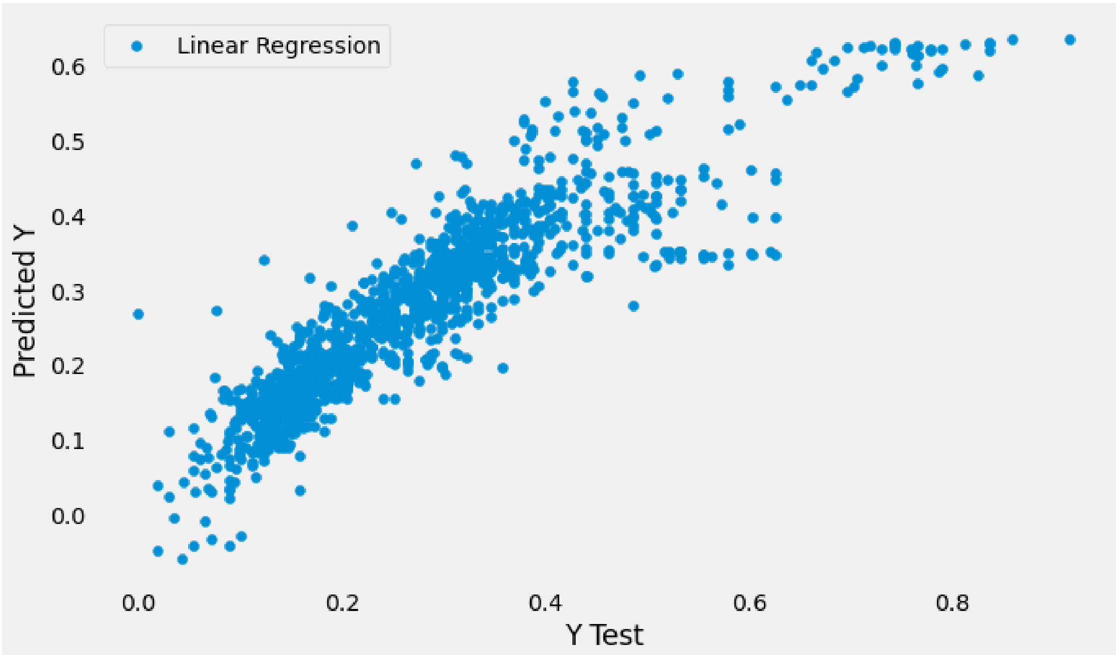
[76]:
```
mseLin = mean_squared_error(y_test.values, y_predLin)
rmselin = np.sqrt(mean_squared_error(y_test.values, y_predLin))
print('The Mean Squared Error = {0:.4f}'.format(mseLin))
print('Root Mean Squared Error :{0:.4f}'.format(rmselin))
```

```
The Mean Squared Error = 0.0035
Root Mean Squared Error :0.0595
```

[77]:
```
r2_Lin = r2_score(y_test.values, y_predLin)
print('Goodness of Fit: {0:.2f}'.format(r2_Lin))
```

```
Goodness of Fit: 0.82
```

[78]:
```
plt.figure(figsize = (10,6))
plt.scatter(y_test, y_predLin, label='Linear Regression')
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
plt.legend(loc='upper left');
plt.grid()
```

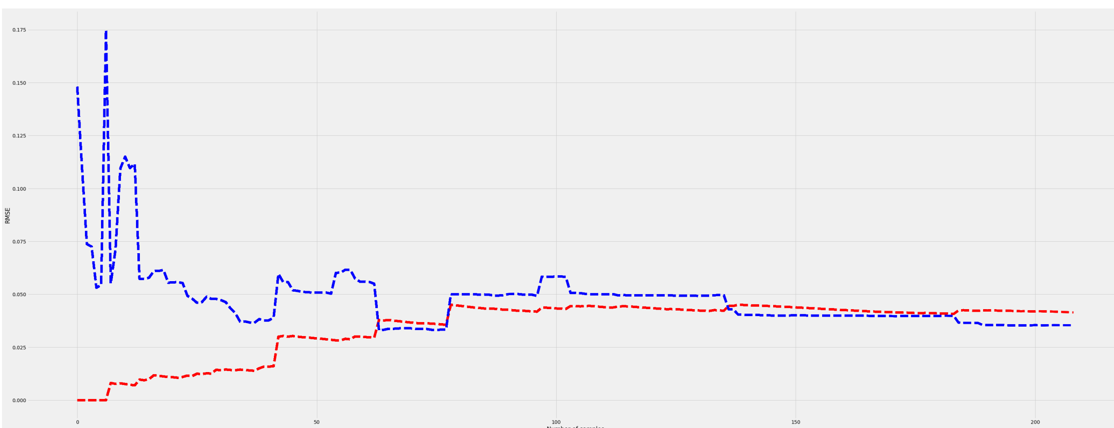


[79]:
```
#cross validation
scores = cross_val_score(linModel, x1_train, y_train,␣
 ↪scoring="neg_mean_squared_error", cv=10)
linModel_rmse_scores = np.sqrt(-scores)

linModel_r2_scores = cross_val_score(linModel, x1_train, y_train, scoring="r2",␣
 ↪cv=10)



def show_scores(scores, model):
    print("       ",model,"    ")
    print("scores:",scores)
    print("scores_mean :", scores.mean())


show_scores(linModel_rmse_scores,linModel)
print("-"*10)
show_scores(linModel_r2_scores,linModel)
```

```
       LinearRegression()
scores: [0.05044419 0.05246223 0.06103618 0.0565169  0.05393032 0.05612132
 0.05849508 0.06159356 0.05460134 0.05149799]
scores_mean : 0.05566991094250119
----------
       LinearRegression()
```

```
scores: [0.85579192 0.83505457 0.79464953 0.82317108 0.84095551 0.81529993
 0.81440906 0.81414578 0.83610793 0.86656255]
scores_mean : 0.8296147843789166
```

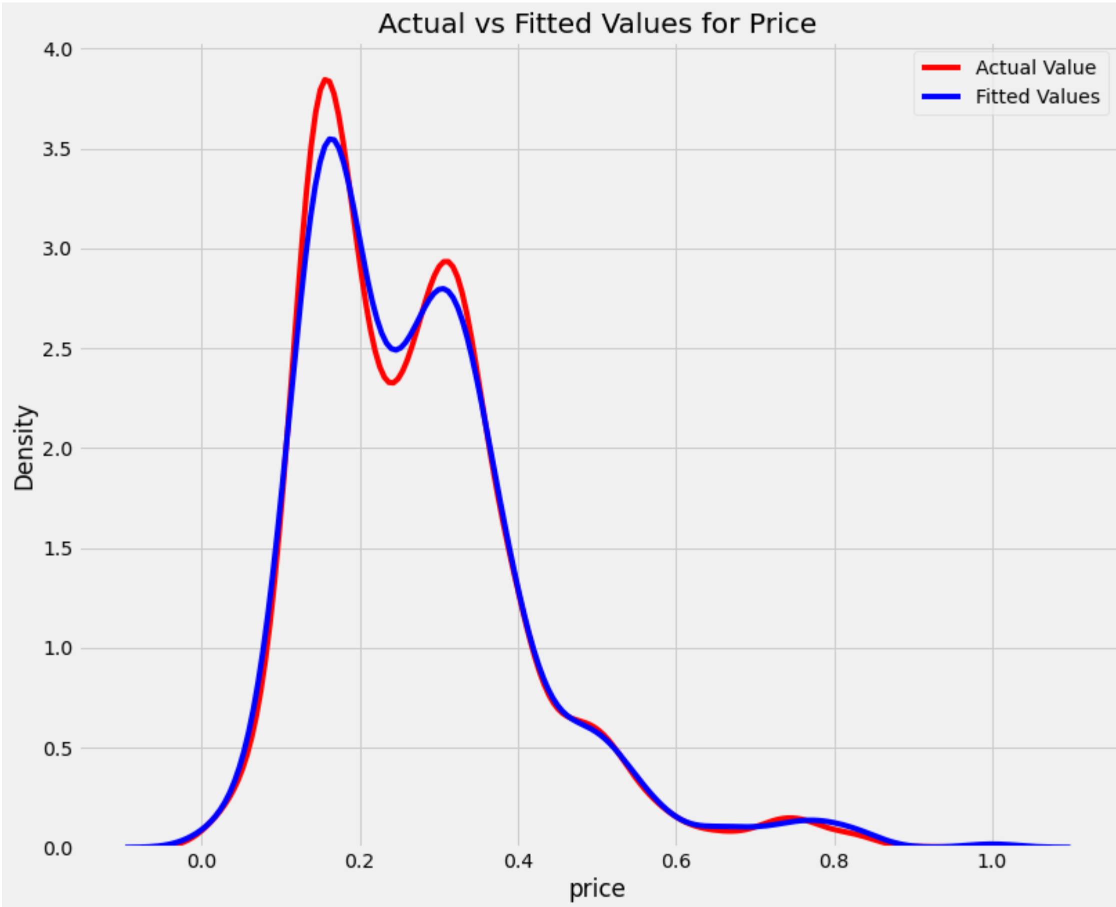
[80]:
```
def plot_learning_curves (model, X, y):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
    train_errors, val_errors = [], []
    for m in range(1, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_pred = model.predict(X_train[:m])
        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.rcParams["figure.figsize"] = (50, 20)
        plt.plot(np.sqrt(train_errors), 'r--', linewidth=6, label='train')
        plt.plot(np.sqrt(val_errors), 'b--', linewidth=6, label='val')
        plt.ylabel('RMSE')
        plt.xlabel('Number of samples')




plot_learning_curves(linModel, x1[:300], y[:300])
```



[81]:
```
DecisionTree_Model = DecisionTreeRegressor()
DecisionTree_Model.fit(x1_train.values, y_train.values)
y_dt_pred = DecisionTree_Model.predict(x1_test.values)
print("actual value", "             predicted value")
i = 1
while i < 20:
    print(y_test.values[i],"       ",y_dt_pred[i])
```

```
    i += 1

plt.figure(figsize=(12, 10))
ax = sns.distplot(y, hist=False, color="r", label="Actual Value")
sns.distplot(y_dt_pred, hist=False, color="b", label="Fitted Values" , ax=ax)
plt.title('Actual vs Fitted Values for Price')
plt.legend()
plt.show()
```

```
actual value             predicted value
0.23437317443626593         0.32242084355649026
0.06998481130973246         0.1003621918448417
0.415819605094053         0.5019277953031896
0.3726837247341979          0.48592125248276663
0.2754761070218483          0.2732094870896133
0.25236593059936907         0.29884332281808623
0.18226428321065544         0.1983876621100596
0.22895198037153874         0.19628461268839817
0.23016707559294308         0.2754994742376446
0.7195934104451455          0.8364294894263349
0.18226428321065544         0.17455310199789695
0.35494800794485337         0.5209720761771235
0.2983993457179577          0.28694940997780116
0.19850449818904076         0.20551466292791212
0.03502745647856058         0.0186937726369903
0.12379950928846829         0.12382287650426453
0.12641663745764692         0.11197569809557191
0.5069517466993808          0.5326556840752424
0.4392802897534759          0.48592125248276663
```



[82]:
```
mse_dt = mean_squared_error(y_test.values, y_dt_pred)
rmse_dt = np.sqrt(mean_squared_error(y_test.values, y_dt_pred))
print('The Mean Squared Error = {0:.4f}'.format(mse_dt))
print('Root Mean Squared Error :{0:.4f}'.format(rmse_dt))
```

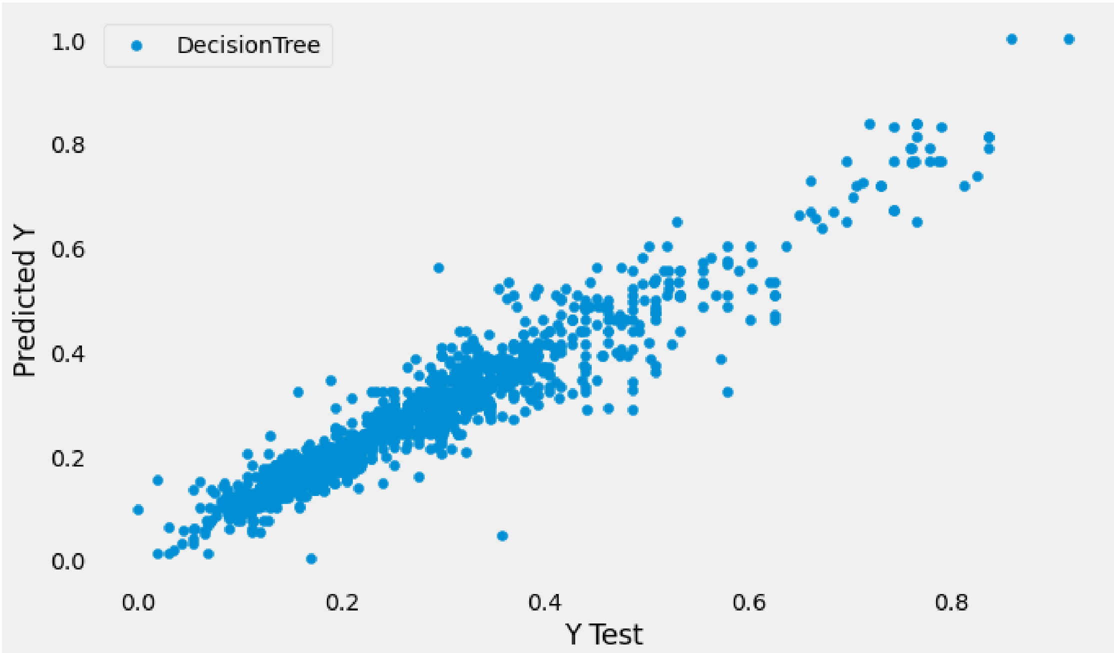
```
The Mean Squared Error = 0.0018
Root Mean Squared Error :0.0422
```

[83]:
```
r2_dt = r2_score(y_test.values, y_dt_pred)
print('Goodness of Fit: {0:.2f}'.format(r2_dt))
```

```
Goodness of Fit: 0.91
```

[84]:
```
plt.figure(figsize = (10,6))
plt.scatter(y_test, y_dt_pred, label='DecisionTree')
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
plt.legend(loc='upper left')
plt.legend()
```

```
plt.grid()
```



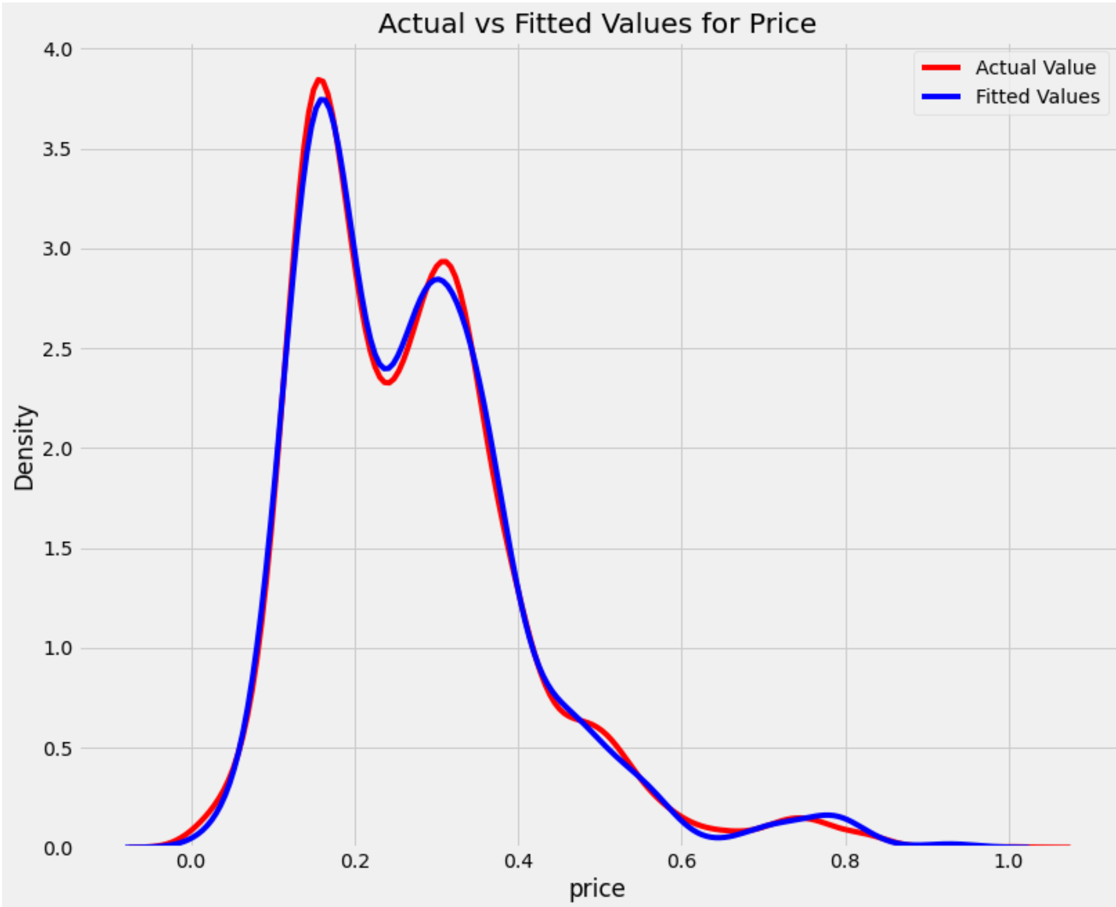
[85]:
```
scores = cross_val_score(DecisionTree_Model, x1_train, y_train,␣
 ↪scoring="neg_mean_squared_error", cv=10)
DecisionTree_rmse_scores = np.sqrt(-scores)

DecisionTree_r2_scores = cross_val_score(DecisionTree_Model, x1_train, y_train,␣
 ↪scoring="r2", cv=10)



def show_scores(scores, model):
    print("       ",model,"    ")
    print("scores:",scores)
    print("scores_mean", scores.mean())


show_scores(DecisionTree_rmse_scores,DecisionTree_Model)
print("-"*10)
show_scores(DecisionTree_r2_scores,DecisionTree_Model)
```

```
       DecisionTreeRegressor()
scores: [0.0423774  0.04254543 0.04886479 0.04438553 0.04061207 0.04085776
 0.03956199 0.03838584 0.04135941 0.03784534]
scores_mean 0.041679555911149316
----------
```

```
       DecisionTreeRegressor()
scores: [0.89514572 0.89137096 0.86564151 0.89069586 0.91161051 0.90460788
 0.91313647 0.92709941 0.90833178 0.93066687]
scores_mean 0.9038306959106196
```

[86]:
```
randomForestModel = RandomForestRegressor()
randomForestModel.fit(x1_train.values, y_train.values)
y_rf_pred = randomForestModel.predict(x1_test.values)
print("actual value", "predicted value")
i = 1
while i < 20:
    print(y_test.values[i],"       ",y_rf_pred[i])
    i += 1
plt.figure(figsize=(12, 10))
ax = sns.distplot(y, hist=False, color="r", label="Actual Value")
sns.distplot(y_rf_pred, hist=False, color="b", label="Fitted Values" , ax=ax)
plt.title('Actual vs Fitted Values for Price')
plt.legend()
plt.show()
```

```
actual value predicted value
0.23437317443626593         0.3323713050590024
0.06998481130973246         0.10954807804650077
0.415819605094053         0.4556097674962028
0.3726837247341979         0.45131907933169807
0.2754761070218483         0.2561189391284027
0.25236593059936907         0.2910033882462907
0.18226428321065544         0.19974319429839962
0.22895198037153874         0.18912186003037731
0.23016707559294308         0.27719336371071374
0.7195934104451455         0.7423297114148856
0.18226428321065544         0.17813716555672404
0.35494800794485337         0.4531398527865403
0.2983993457179577         0.3011928496319666
0.19850449818904076         0.201422128753359
0.03502745647856058         0.03990325972660355
0.12379950928846829         0.11990723215328908
0.12641663745764692         0.11180465007594341
0.5069517466993808         0.5094284379016243
0.4392802897534759         0.47723799509288495
```



[87]:
```
mse_rf = mean_squared_error(y_test.values, y_rf_pred)
rmse_rf = np.sqrt(mean_squared_error(y_test.values, y_rf_pred))
print('The Mean Squared Error = {0:.4f}'.format(mse_rf))
print('Root Mean Squared Error :{0:.4f}'.format(rmse_rf))
```

```
The Mean Squared Error = 0.0012
Root Mean Squared Error :0.0345
```

[88]:
```
r2_rf = r2_score(y_test.values, y_rf_pred)
print('Goodness of Fit: {0:.2f}'.format(r2_rf))
```

```
Goodness of Fit: 0.94
```

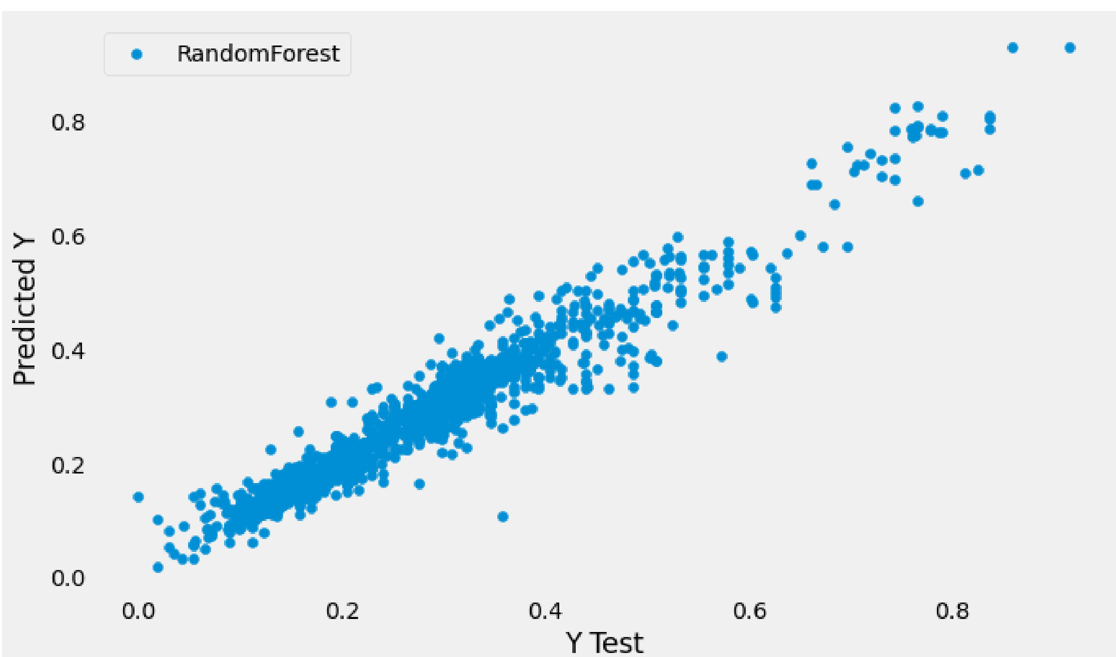
[89]:
```
scores_randomforrest = cross_val_score(randomForestModel, x1_train, y_train,␣
 ↪scoring="neg_mean_squared_error", cv=10)
randomForestModel_rmse_scores = np.sqrt(-scores_randomforrest)

randomForestModel_r2_scores = cross_val_score(randomForestModel, x1_train,␣
 ↪y_train, scoring="r2", cv=10)
```

```
def show_scores(scores, model):
    print("       ",model,"    ")
    print("scores:",scores)
    print("scores_mean", scores.mean())


show_scores(randomForestModel_rmse_scores,randomForestModel)
print("-"*10)
show_scores(randomForestModel_r2_scores,randomForestModel)
```

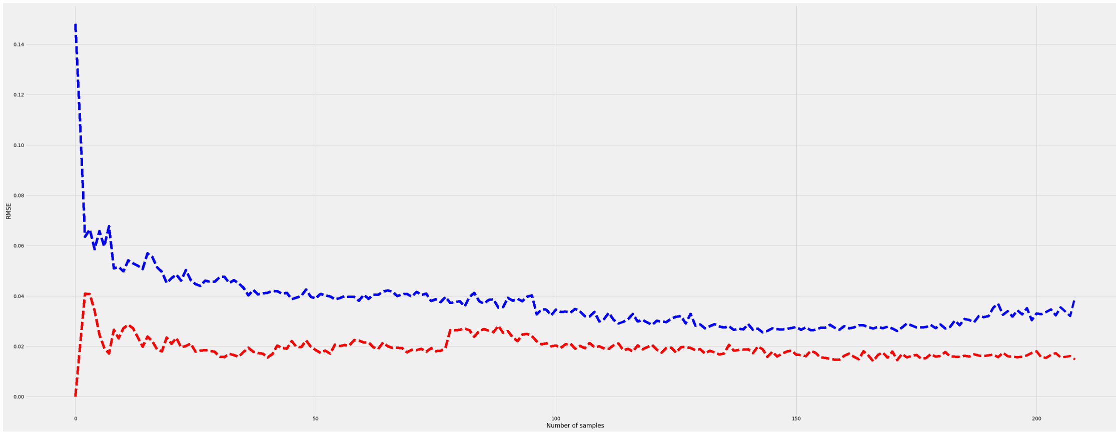
```
       RandomForestRegressor()
scores: [0.03488171 0.03422601 0.03764459 0.03539716 0.03212558 0.0314211
 0.03154996 0.03245999 0.03173164 0.0314562 ]
scores_mean 0.033289393157439416
----------
       RandomForestRegressor()
scores: [0.93150723 0.9300576  0.91968305 0.92960292 0.94302227 0.93973645
 0.9466437  0.94558255 0.94571408 0.95083895]
scores_mean 0.9382388808606391
```

[90]:

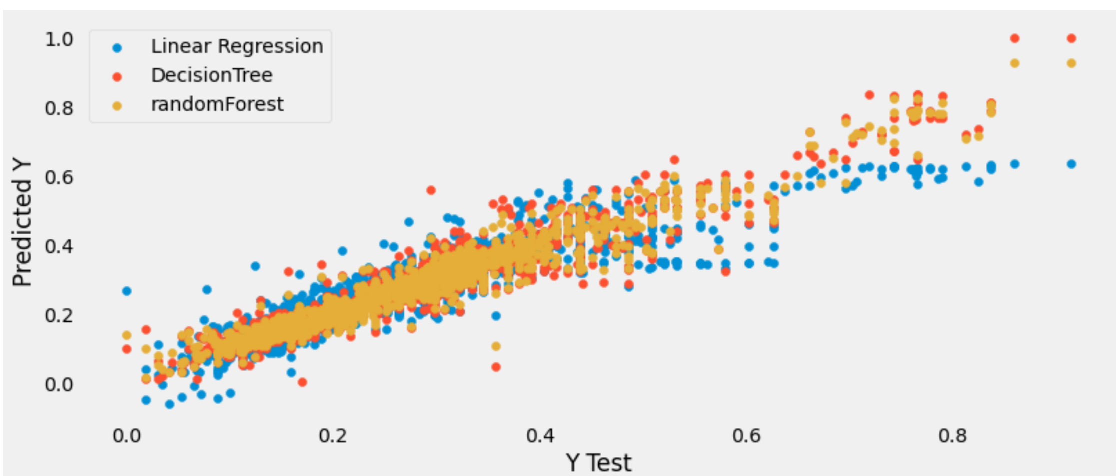
```
plt.figure(figsize = (10,6))
plt.scatter(y_test, y_rf_pred, label='RandomForest')
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
plt.legend(loc='upper left');
plt.grid()
```



[91]:
```python
plot_learning_curves(randomForestModel, x1[:300], y[:300 ])
```



[92]:
```python
plt.figure(figsize = (12,5))
plt.scatter(y_test, y_predLin, label='Linear Regression')
plt.scatter(y_test, y_dt_pred, label='DecisionTree')
plt.scatter(y_test, y_rf_pred, label='randomForest')
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
plt.legend(loc='upper left');
plt.grid()
```



[93]:
```python
print("R2  score by using MultiLinear Regression:", r2_Lin)
print("R2  score by using Decision Tree          :", r2_dt)
print("R2  score by using Random Forest          :", r2_rf)
```

```
R2  score by using MultiLinear Regression: 0.822348652930147
R2  score by using Decision Tree         : 0.9105998428335186
R2  score by using Random Forest         : 0.9402123812737575
```

[94]:
```
print("RMSE  score by using MultiLinear Regression:", rmselin)
print("RMSE  score by using Decision Tree         :", rmse_dt)
print("RMSE  score by using Random Forest         :", rmse_rf)
```

```
RMSE  score by using MultiLinear Regression: 0.05945212959771072
RMSE  score by using Decision Tree         : 0.042174729374081135
RMSE  score by using Random Forest         : 0.03448965034328435
```

By comparing the results we collected in each step, we can conclude that our first set of columns, which we consider as X due to contain thos clumns with the higher corrolation give us the better result and the **Random Forest model** provide the better result which is more accurate besides the lower errors ratio incomparision with other models's result,so it's predicted price are closer to the actual values in comparision with other models(the decision tree model also have a great accuracy -more than 90%- which can consider it as a good model but random forest accuracy is the best .

According to the results I obtained during the project, Random Forest has an accuracy of 95% for predicting the price in the data of the first selected columns, which decreases in the second selection and reaches 94%.We can notice the same trend in the result of Rmse errors in both steps. So we can consider the **Random Forest model** as the final model for price prediction in our project.