*The work We submitted as part of this assignment is original, and due credit is given to others where appropriate. We accept and acknowledge that We are **solely** responsible for if the assignment found to be plagiarized in any way, and we will be subject to School's Academic Integrity policy*

Group 10: Jiadai Yu(U63347232), Yuesen Zhang(U25148433), Xin Su(U07869307)

Date: 29th Aug 2022

Based on the correlation matrix, our group would suggest to include the price, year, mileage, engine size, and tax. The variables we did not include are mpg, transmission, and fuel type. Since the price is the factor the clients are most concerned about, our group focused on analyzing the regression analysis between the price and other variables.

From the matrix, we separated these variables into two types one is time-related, and the other is environment related. The time-related type has the variables year and mileage, and both of these two variables have a strong correlation with the price. Hence, these two variables, year and mileage, are chosen in the regression analysis. In addition, the environment-related type has the variables engine size, tax, mpg, transmission, and fuel type. Among these variables, the fuel type and transmission have the weakest correlation with the price, so we excluded them from the regression analysis. Moreover, both engine size and tax have a 0.41 correlation with the price, while the mpg has a -0.5 correlation with the tax. In other words, these three variables are dependent on each other. Therefore, we picked two of these three variables, engine size and tax, for our regression analysis.

Overall, our group suggests to include price, year, mileage, engine size, and tax in the regression analysis.

In this case, we keep all the explanatory variables the same, the only difference between Model 1 and Model 2 is the response variable. After linear regression and modeling, we obtain the models:

Model 1

price = 1.21292845e+03*year - 6.04409697e-02*mileage
+ 5.67107198e+03*engineSize + 6.50246287e+00*tax
- 2441011.0945755113

Model 2

log_price = 1.30621388e-01*year - 4.95493934e-06*mileage
+ 3.92918890e-01*engineSize + 2.47154619e-04*tax
- 254.5480523731459

We know that in Python, R^2 represents the fraction of variance of the actual value of the response variable captured by the regression model, which is bounded between 0 and 1. If the value of R^2 is greater, it means the response variable captured by the model is better-observed. In other words, if the value of R^2 is higher, the model fits better with the dataset. For Model 1, we get the value of R^2 is 0.7045. For Model2, we get the value of R^2 is 0.7856. According to the definition and the comparison of the data, we can conclude that Model2 is more well-observed.

In addition, we also obtain the Mean Squared Error(MSE). We know that by definition the MSE represents the error of the estimator or predictive model created based on the given set of observations in the sample. Thus it is used to measure the quality of the model based on the predictions made on the entire training dataset vis-a-vis the true label/output value. If the value of MSE is higher then the model has more errors. If the value of MSE is lower then the model has less error and more accuracy. For Model 1, we get the value of MSE is 6715130.61. For Model 2, we get the value of MSE is 0.03535. We observed that the MSE of Model 2 is significantly smaller than the value of Model 1, and very close to 0. Which indicates the outstanding accuracy of Model 2.

Lastly, we can also compare the actual price of the test data and the predicted output. We know that the predicted value should be close to the actual value, then we can conclude the model is the best fit for the dataset. For Model 1, We observed that the actual price1 is $12495 and the predicted value is $15337, which has a 22.75% of the difference. For Model 2, we observed that the actual price2 is $12495 and the predicted value is $ 14238.18, which has a 13.95% difference. Then we can see that Model 2 has less percentage difference, and we can conclude that Model 2 is the best fit.

We would propose the second model to our client as the final recommendation, which is the linear regression model with production year, number of miles the vehicle traveled, the car's engine size, and annual tax as explanatory variables, and the logarithm of price as the dependent variable. The reason is not only that the statistical results of this model are better than first model, it also has a stronger probabilistic theoretical support.

From the scatter plot of production year and mile age versus price, and the prediction of test data, we can tell that the distribution of the dependent variable price is not linear. The reason why the first regression model achieved a seemingly satisfactory result that R^2 = 0.7, is due to the relatively small range of the data. Within this small distance, the relationship between price and the independent variables is roughly linear. If the client would like to promote the model to predict a larger range of data, the predictions are likely to be inaccurate.

The logarithm of price, on the other hand, shows a significant linearity in the scatter plots with both the production year and mile age of each vehicle. In addition, the test data results yielded by the second model predictions are evenly distributed on both sides of the linear regression. A linear regression model such as the second one is suitable for generalization to a larger range of data and is also more likely to produce valid pridictions in data sets beyond those used in the current study.

For further discussion, we can justify the log-price model with probability theory. In terms of the random data distribution, year and mile age are variables that describe " the time until an event occurs " which fit the exponential distribution. The independent variables, year and mile age, have an exponential level influence on price. After logging both sides of the exponential distribution equation, there is a linear correlation between log price and independent variable.

In summary, the log-price model provides more reliable and valid statistical results, is more suitable for generalization to a larger range of data, and adheres better to the theoretical knowledge of probability theory. We would recommend this model to my clients.

In [1]:
```python
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import sklearn.metrics as metrics
import warnings

filename = r'D:\0-Summer Pre\boot camp\group project\Ford-Data-GroupProject.xlsx'
Ford = pd.read_excel(filename , header=0)
```

In [2]:
```python
#--------------------------------
#Basic Information about the data

print(Ford.head())
print(Ford.shape)
print(Ford.info())
```

```
     model  year transmission fuelType  engineSize  price  mileage  tax   mpg
0   Fiesta  2017    Automatic   Petrol         1.0  12000    15944  150  57.7
1    Focus  2018       Manual   Petrol         1.0  14000     9083  150  57.7
2    Focus  2017       Manual   Petrol         1.0  13000    12456  150  57.7
3   Fiesta  2019       Manual   Petrol         1.5  17500    10460  145  40.3
4   Fiesta  2019    Automatic   Petrol         1.0  16500     1482  145  48.7
(17966, 9)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17966 entries, 0 to 17965
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   model         17966 non-null  object
 1   year          17966 non-null  int64
 2   transmission  17966 non-null  object
 3   fuelType      17966 non-null  object
 4   engineSize    17966 non-null  float64
 5   price         17966 non-null  int64
 6   mileage       17966 non-null  int64
 7   tax           17966 non-null  int64
 8   mpg           17966 non-null  float64
dtypes: float64(2), int64(4), object(3)
memory usage: 1.2+ MB
None
```

In [3]:
```python
#drop the unrealistic year = 2060 as an input error

Ford.drop(Ford.index[Ford['year'] > 2022], inplace=True)
```

In [4]:
```python
#categorical variables:

for i in ['model','transmission','fuelType','tax']:
    print('prices vary across '+ i +' :')
    price_by_i = pd.DataFrame(Ford.groupby(i)['price'].mean())
    i_count = Ford[i].value_counts()
    price_by_i_count_srt = pd.concat([price_by_i,i_count] , axis=1).sort_values('price')
    print(price_by_i_count_srt)
    print('\n')
```

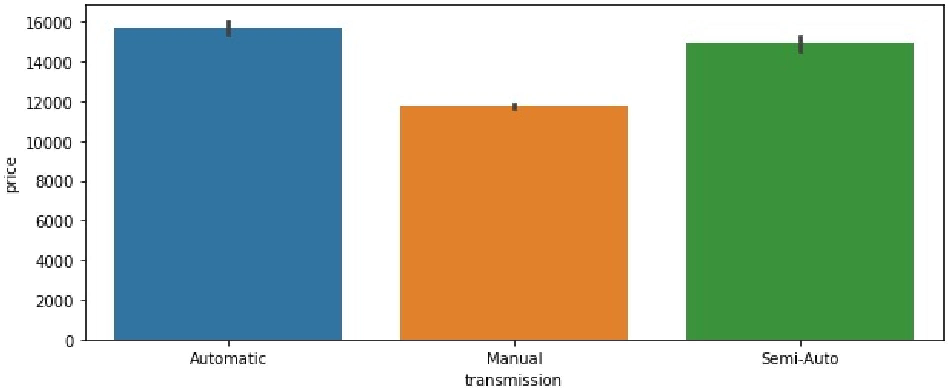
```
prices vary across model :
                               price  model
 Streetka                1924.500000      2
 Fusion                  2555.812500     16
 Escort                  3000.000000      1
 KA                      5186.125628    199
 B-MAX                   8287.526761    355
Focus                    8299.000000      1
 Ka+                     8707.856874    531
 C-MAX                   9914.567219    543
 Fiesta                 10196.862569   6556
 Grand C-MAX            10881.574899    247
 Mondeo                 12305.709125    526
 Transit Tourneo        12450.000000      1
 EcoSport               12499.268591   1143
 Focus                  13185.882956   4588
 Tourneo Connect        13805.818182     33
 Ranger                 14495.000000      1
 Grand Tourneo Connect  14874.915254     59
 Kuga                   15823.472360   2225
 S-MAX                  17720.226351    296
 Galaxy                 17841.872807    228
 Tourneo Custom         21165.985507     69
 Puma                   21447.250000     80
 Edge                   22810.500000    208
 Mustang                34631.263158     57


prices vary across transmission :
                  price  transmission
Manual     11792.264918         15518
Semi-Auto  14919.034039          1087
Automatic  15734.022794          1360


prices vary across fuelType :
                 price  fuelType
Petrol    11608.713582     12178
Diesel    13659.173724      5762
Other     13800.000000         1
Electric  15737.500000         2
Hybrid    22149.090909        22


prices vary across tax :
            price   tax
185   1295.000000     1
290   1850.000000     1
270   1999.000000     1
305   2897.500000     2
220   3628.333333     3
210   3875.000000     1
190   4000.000000     2
300   5333.272727    11
120   5445.000000     2
195   5620.833333     6
265   7216.466667    15
155   7329.666667     3
230   7345.000000     2
555   7507.250000     4
30    8144.397249  1236
22    8299.000000     1
0     8722.670228  2153
325   9427.166667     6
205   9445.389831    59
20    9494.711570  1210
165   9802.672566   113
125  10678.743426  1407
110  10700.000000     1
115  10830.000000     3
200  10883.809524   252
260  10954.365854    41
160  10979.494413   358
240  12005.156250    32
150  13245.809156  1944
330  13747.500000     2
235  13837.788618   123
145  14257.397809  8944
140  21404.125000     8
135  26743.800000    10
570  27327.166667     6
580  27907.500000     2
```

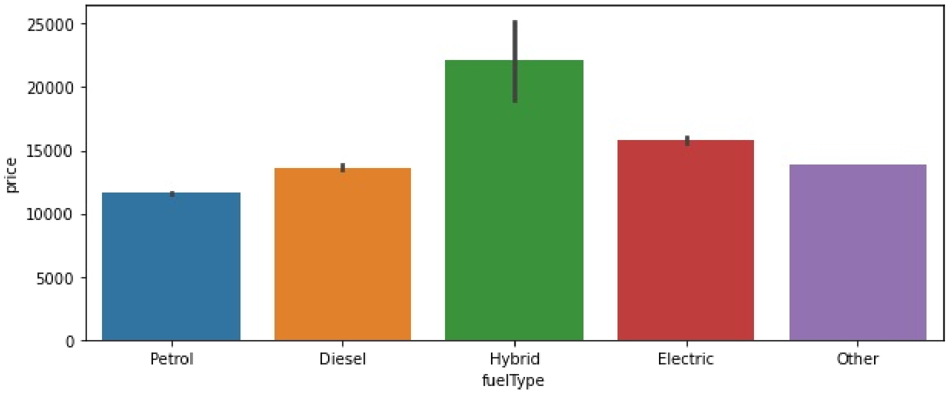
In [5]:

```
#how categorical variables attribute to price

fig=plt.figure(figsize=(10,4))
sns.barplot(x = Ford.transmission, y = Ford.price)
```

```
fig=plt.figure(figsize=(10,4))
sns.barplot(x = Ford.fuelType, y = Ford.price)
```

Out[5]: <AxesSubplot:xlabel='fuelType', ylabel='price'>

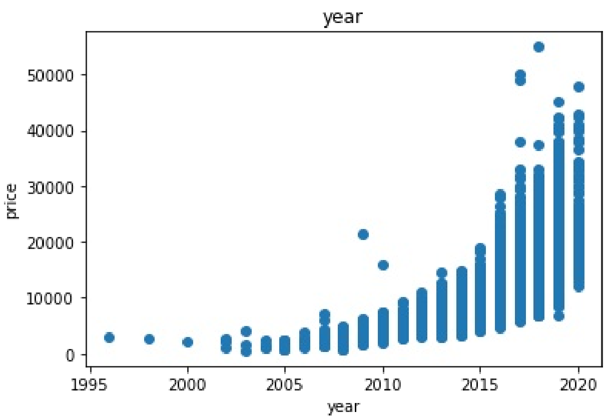




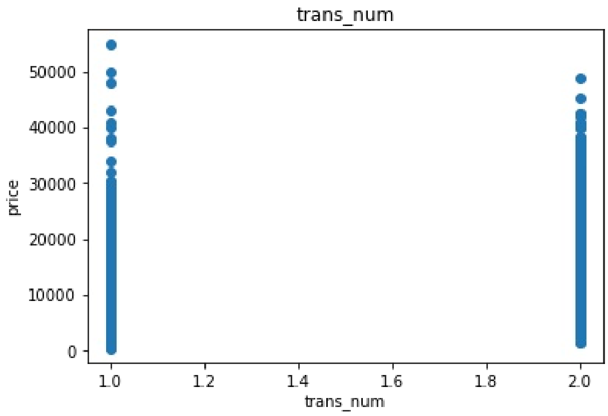
In [6]:
```
#mean price of Semi-Auto and Automatic is similar, divide the transmission of cars into 'manual' and assign as 
#similarly assign fuelType as 'not Hybrid' and 'Hybrid'

##mapping, and assign new columns

trans_mapping = {'Manual':1,'Semi-Auto':2,'Automatic':2}
Ford['trans_num'] = Ford['transmission'].replace(trans_mapping)

fuel_mapping = {'Petrol':1,'Diesel':1,'Other':1,'Electric':1,'Hybrid':2}
Ford['fuel_num'] = Ford['fuelType'].replace(fuel_mapping)

#Merge, to reduce the interference of categorical data to the regression
```

In [7]:
```
#how numeric variables attribute to price

for i in ['year','trans_num','fuel_num','engineSize','mileage','tax','mpg']:
    x = Ford[i]
    y = Ford['price']
    print(np.corrcoef(Ford[i],Ford['price']))
    plt.scatter(x, y, marker='o')
    plt.title(i)
    plt.xlabel(i)
    plt.ylabel('price')
    plt.show()
```

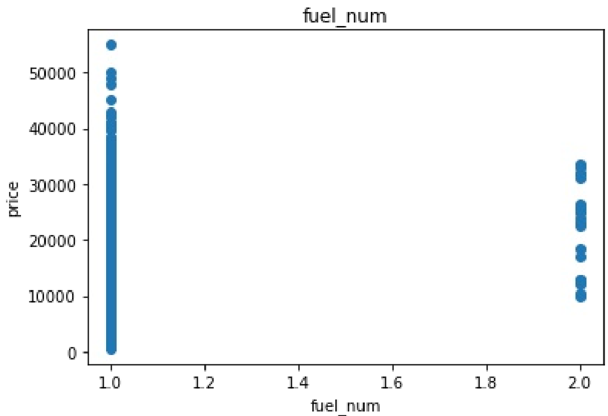
```
[[1.         0.64546504]
 [0.64546504 1.        ]]
```



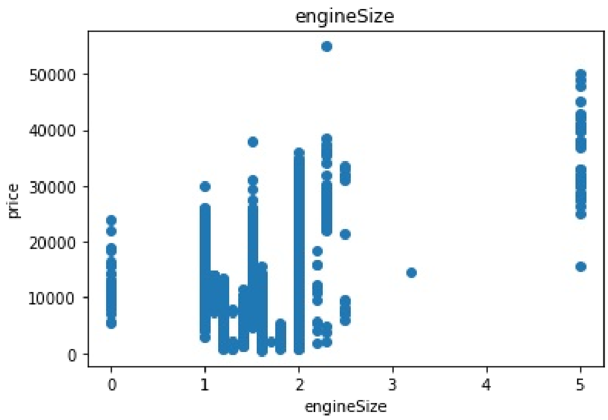

```
[[1.        0.2589846]
 [0.2589846 1.       ]]
```



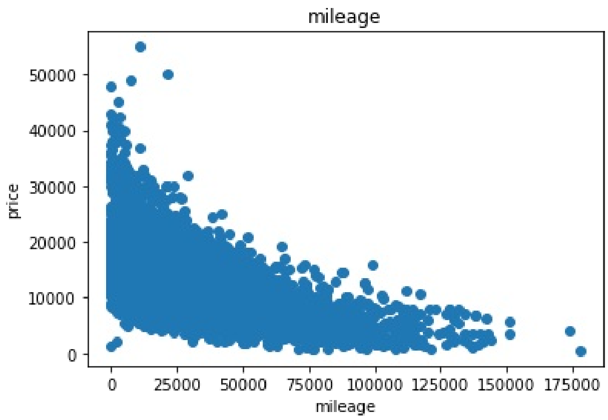

```
[[1.         0.07288931]
 [0.07288931 1.        ]]
```



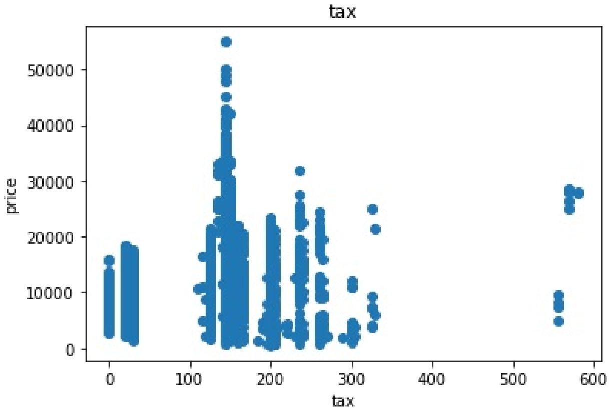

```
[[1.       0.411203]
 [0.411203 1.      ]]
```



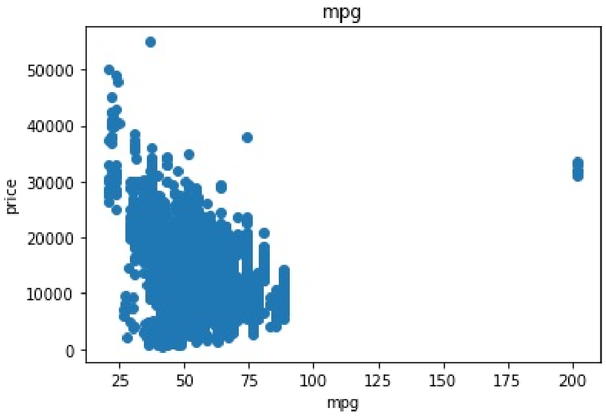

```
[[ 1.         -0.53060994]
 [-0.53060994  1.        ]]
```




```
[[1.         0.40699933]
 [0.40699933 1.        ]]
```

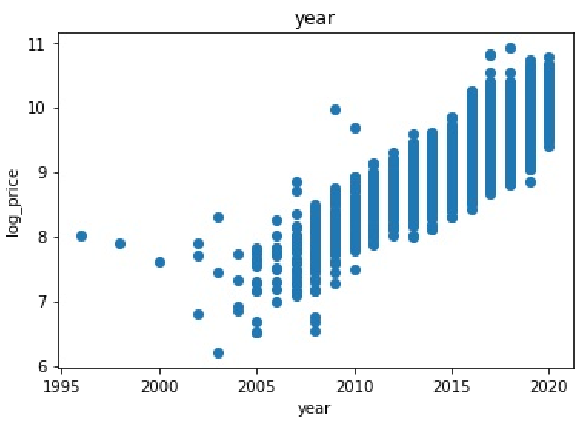



```
[[ 1.         -0.34655662]
 [-0.34655662  1.        ]]
```

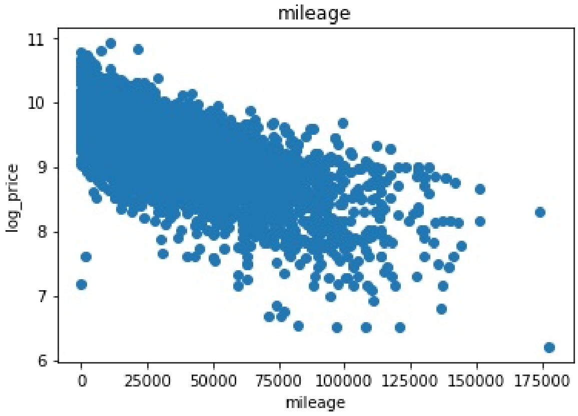


In [8]:
```python
#most engineSize fall near 0 and between 1-3
#most tax fall near 0 and between 100-350
#most mpg fall between 10-90, with a few high outliers

#year & mileage seemed to be correlated with log(price)
Ford['log_price'] = np.log(Ford['price'])

for i in ['year','mileage']:
    x = Ford[i]
    y = Ford['log_price']
    print(np.corrcoef(Ford[i],Ford['log_price']))
    plt.scatter(x, y, marker='o')
    plt.title(i)
    plt.xlabel(i)
    plt.ylabel('log_price')
    plt.show()
```

```
[[1.         0.77352907]
 [0.77352907 1.        ]]
```

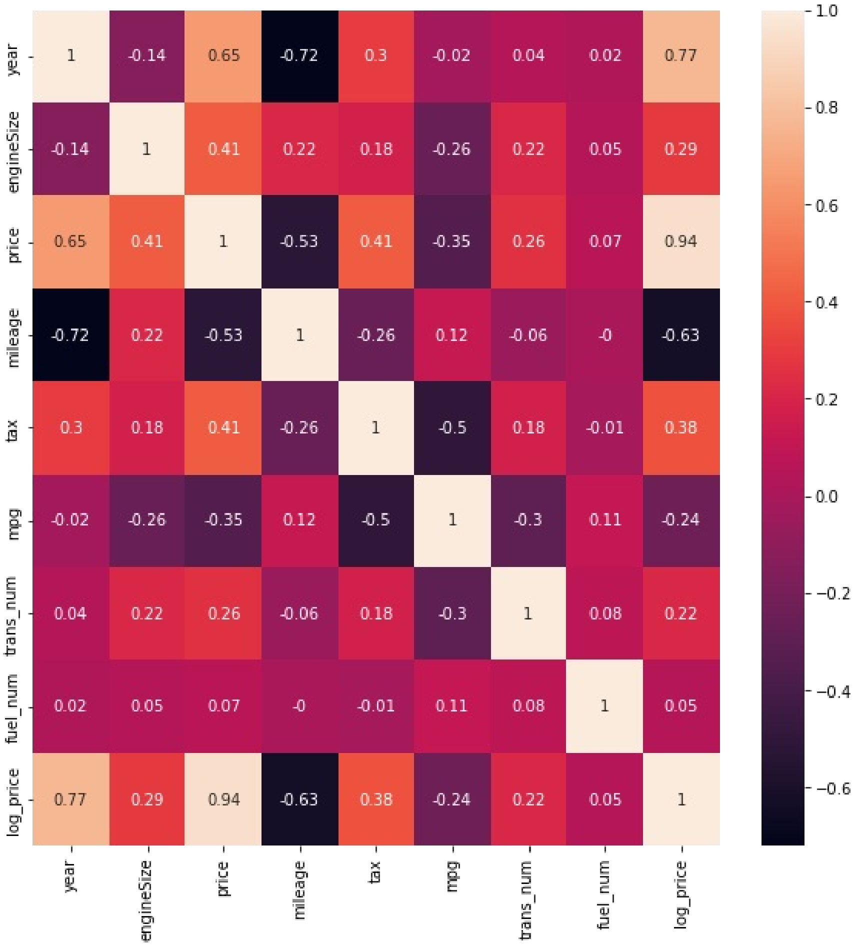



```
[[ 1.         -0.62649271]
 [-0.62649271  1.        ]]
```



In [9]:
```python
# Correlation Matrix - states the relation between each column
correlation_matrix = Ford.corr().round(2)
# annot = True to print the values inside the square
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(data=correlation_matrix, annot=True, ax = ax)
```

Out[9]: <AxesSubplot:>



In [10]:
```python
#Model 1

#year has 0.65 correlation with price, mileage has -0.53 correlation with price,
#meanwhile year and mileage have -0.72 correlation with each other, are highly dependent

#engineSize and tax both have 0.41 correlation with price
#tax have -0.5 correlation with mpg

#Model1: X = year, mileage, engineSize, tax, Y = price
```

In [11]:
```python
#Model 2

#year has 0.77 correlation with log_price, mileage has -0.63 correlation with log_price,
#meanwhile year and mileage have -0.72 correlation with each other, are highly dependent

#tax 0.38,and tax has -0.5 correlation with mpg
# engineSize 0.29
```

```
#Model2: X = year, mileage, engineSize, tax, Y = log_price
```

In [12]:
```
# converting the daya set into array
X1 = np.array(pd.concat([Ford['year'],Ford['mileage'],Ford['engineSize'],Ford['tax']],axis=1))
Y1 = Ford['price'].values
X2 = np.array(pd.concat([Ford['year'],Ford['mileage'],Ford['engineSize'],Ford['tax']],axis=1))
Y2 = Ford['log_price'].values

# splitting the dataset into train and test. 30% of the whole data is for testing
x1_train, x1_test, y1_train, y1_test = train_test_split(X1,Y1, test_size=0.3, random_state = 42)
x2_train, x2_test, y2_train, y2_test = train_test_split(X2,Y2, test_size=0.3, random_state = 42)

print(x1_train == x2_train)
```

```
[[ True  True  True  True]
 [ True  True  True  True]
 [ True  True  True  True]
 ...
 [ True  True  True  True]
 [ True  True  True  True]
 [ True  True  True  True]]
```

In [13]:
```
#Apllying Multiple linear regression model on the dataset
Ford_model1 = LinearRegression()
Ford_model1.fit(x1_train, y1_train)
Ford_model2 = LinearRegression()
Ford_model2.fit(x2_train, y2_train)

#Predicting the train data using the linear model
y1_pred_train = Ford_model1.predict(x1_train)
y2_pred_train = Ford_model2.predict(x2_train)

#Predicting the test data using the Linear model
y1_pred_test = Ford_model1.predict(x1_test)
y2_pred_test = Ford_model2.predict(x2_test)

print("Coefficients of the Linear Regression Model 1:", Ford_model1.coef_)
print("Intercept of the Linear Regression Model 1:", Ford_model1.intercept_)
print("Coefficients of the Linear Regression Model 2:", Ford_model2.coef_)
print("Intercept of the Linear Regression Model 2:", Ford_model2.intercept_)
```

```
Coefficients of the Linear Regression Model 1: [ 1.21292845e+03 -6.04409697e-02  5.67107198e+03  6.50246287e+00
]
Intercept of the Linear Regression Model 1: -2441011.0945755113
Coefficients of the Linear Regression Model 2: [ 1.30621388e-01 -4.95493934e-06  3.92918890e-01  2.47154619e-04
]
Intercept of the Linear Regression Model 2: -254.5480523731459
```

In [14]:
```
#Formula of the model1:
#price = 1.21292845e+03*year - 6.04409697e-02*mileage + 5.67107198e+03*engineSize + 6.50246287e+00*tax - 244101

#Formula of the model2:
#log_price = 1.30621388e-01*year - 4.95493934e-06*mileage + 3.92918890e-01*engineSize + 2.47154619e-04*tax - 25
```

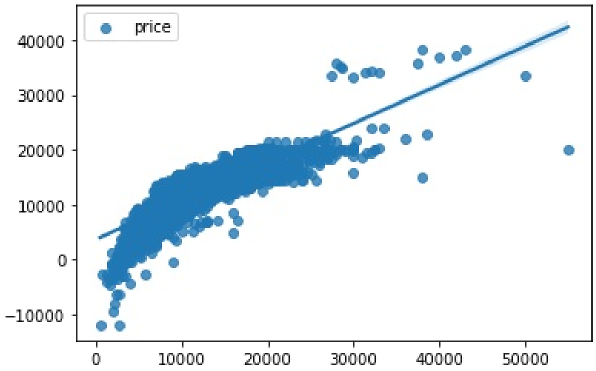
In [15]:
```
# Evaluating them odel using R^2 and Mean Square Error
print("Evaluation of Model 1:")
print("Train Data:")
print("R2:", metrics.r2_score(y1_train, y1_pred_train))
print('MSE:', metrics.mean_squared_error(y1_train, y1_pred_train))
print("Test Data:")
print("R2:", metrics.r2_score(y1_test, y1_pred_test))
print('MSE:', metrics.mean_squared_error(y1_test, y1_pred_test))
print()
print("Evaluation of Model 2:")
print("Train Data:")
print("R2:", metrics.r2_score(y2_train, y2_pred_train))
print('MSE:', metrics.mean_squared_error(y2_train, y2_pred_train))
print("Test Data:")
print("R2:", metrics.r2_score(y2_test, y2_pred_test))
print('MSE:', metrics.mean_squared_error(y2_test, y2_pred_test))
```

```
Evaluation of Model 1:
Train Data:
R2: 0.7171205257472901
MSE: 6328658.5325517915
Test Data:
R2: 0.7045192726008103
MSE: 6715130.612902848

Evaluation of Model 2:
Train Data:
R2: 0.7912761725664388
MSE: 0.03405826853750039
Test Data:
R2: 0.7856161317308468
MSE: 0.03535325769257484
```

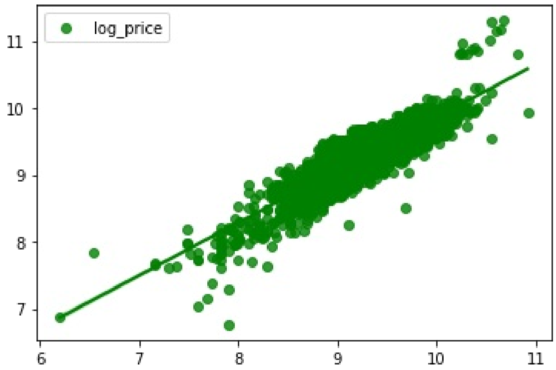
In [16]:
```
model1 = sns.regplot(y1_test,y1_pred_test,label='price')
```

```
model1.legend(loc="best")
plt.show()
model2 = sns.regplot(y2_test,y2_pred_test,label='log_price',color='g')
model2.legend(loc="best")
plt.show()
```

```
C:\Users\yujia\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variabl
es as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing oth
er arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```




```
C:\Users\yujia\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variabl
es as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing oth
er arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [17]:

```
# Viewing the Actual and Predicted test data
y1_pred_test_prop = 100 * (y1_pred_test - y1_test) / y1_test
y2_pred_test_prop = 100 * ( np.exp(y2_pred_test) - np.exp(y2_test)) / np.exp(y2_test)

pd.DataFrame({"Actual price1 $": y1_test,"Model1 Predict $": y1_pred_test, 'Percentage difference1 %':y1_pred_t
              "Actual price2 $": np.exp(y2_test),"Model2 Predict $": np.exp(y2_pred_test), 'Percentage differen
```

Out[17]:

	Actual price1 $	Model1 Predict $	Percentage difference1 %	Actual price2 $	Model2 Predict $	Percentage difference2 %
0	12495	15337.935630	22.752586	12495.0	14238.186007	13.951068
1	8999	10358.034754	15.102064	8999.0	9911.860278	10.144019
2	7998	9453.844850	18.202611	7998.0	8595.231905	7.467266
3	5491	3878.884776	-29.359228	5491.0	5323.545407	-3.049619
4	3790	2495.748208	-34.149124	3790.0	3772.756562	-0.454972
...	...	...	...	...	...	...
5385	4298	6974.165945	62.265378	4298.0	6977.403922	62.340715
5386	10800	12134.821939	12.359462	10800.0	11846.210762	9.687137
5387	6597	7950.857907	20.522327	6597.0	7589.529923	15.045171
5388	10250	8964.412441	-12.542318	10250.0	8631.733142	-15.787969
5389	8999	10933.408548	21.495817	8999.0	10141.191431	12.692426

5390 rows × 6 columns

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js