this model predicts the price of used cars according to diffrent factors!

importing libraries

In [1]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import seaborn as sns
sns.set()
```

1.loading the data

In [2]:
```
data = pd.read_csv("cars_data.csv")
#the top 5 rows of the df
data.head()
```

Out[2]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year	Model
0	BMW	4200.0	sedan	277	2.0	Petrol	yes	1991	320
1	Mercedes-Benz	7900.0	van	427	2.9	Diesel	yes	1999	Sprinter 212
2	Mercedes-Benz	13300.0	sedan	358	5.0	Gas	yes	2003	S 500
3	Audi	23000.0	crossover	240	4.2	Petrol	yes	2007	Q7
4	Toyota	18300.0	crossover	120	2.0	Petrol	yes	2011	Rav 4

data preprocessing

Exploring the descriptive statistics of the variables

In [3]:
```python
# Descriptive statistics are very useful for initial exploration of the variables
# By default, only descriptives for the numerical variables are shown
# To include the categorical ones, you should specify this with an argument
data.describe(include = 'all')
```

Out[3]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year
count	4345	4173.000000	4345	4345.000000	4195.000000	4345	4345	4345.000000
unique	7	NaN	6	NaN	NaN	4	2	NaN
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	NaN
freq	936	NaN	1649	NaN	NaN	2019	3947	NaN
mean	NaN	19418.746935	NaN	161.237284	2.790734	NaN	NaN	2006.550058
std	NaN	25584.242620	NaN	105.705797	5.066437	NaN	NaN	6.719097
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.000000
25%	NaN	6999.000000	NaN	86.000000	1.800000	NaN	NaN	2003.000000
50%	NaN	11500.000000	NaN	155.000000	2.200000	NaN	NaN	2008.000000
75%	NaN	21700.000000	NaN	230.000000	3.000000	NaN	NaN	2012.000000
max	NaN	300000.000000	NaN	980.000000	99.990000	NaN	NaN	2016.000000

In [4]:
```python
data = data.drop(['Model'], axis = 1)
data.describe(include = 'all')
```

Out[4]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year
count	4345	4173.000000	4345	4345.000000	4195.000000	4345	4345	4345.000000
unique	7	NaN	6	NaN	NaN	4	2	NaN
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	NaN
freq	936	NaN	1649	NaN	NaN	2019	3947	NaN
mean	NaN	19418.746935	NaN	161.237284	2.790734	NaN	NaN	2006.550058
std	NaN	25584.242620	NaN	105.705797	5.066437	NaN	NaN	6.719097
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.000000
25%	NaN	6999.000000	NaN	86.000000	1.800000	NaN	NaN	2003.000000
50%	NaN	11500.000000	NaN	155.000000	2.200000	NaN	NaN	2008.000000
75%	NaN	21700.000000	NaN	230.000000	3.000000	NaN	NaN	2012.000000
max	NaN	300000.000000	NaN	980.000000	99.990000	NaN	NaN	2016.000000

In [5]:
```
# data.isnull() # shows a df with the information whether a data point is null
# Since True = the data point is missing, while False = the data point is not missing
# This will give us the total number of missing values feature-wise
data.isnull().sum()
```

Out[5]:
```
Brand             0
Price           172
Body              0
Mileage           0
EngineV         150
Engine Type       0
Registration      0
Year              0
dtype: int64
```

removing null values as it is less that 5% of the total data, due to rule of thumb

In [6]:
```
data = data.dropna(axis = 0)
```

In [7]:
```
data.describe(include = 'all')
```

Out[7]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year
count	4025	4025.000000	4025	4025.000000	4025.000000	4025	4025	4025.000000
unique	7	NaN	6	NaN	NaN	4	2	NaN
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	NaN
freq	880	NaN	1534	NaN	NaN	1861	3654	NaN
mean	NaN	19552.308065	NaN	163.572174	2.764586	NaN	NaN	2006.379627
std	NaN	25815.734988	NaN	103.394703	4.935941	NaN	NaN	6.695595
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.000000
25%	NaN	6999.000000	NaN	90.000000	1.800000	NaN	NaN	2003.000000
50%	NaN	11500.000000	NaN	158.000000	2.200000	NaN	NaN	2007.000000
75%	NaN	21900.000000	NaN	230.000000	3.000000	NaN	NaN	2012.000000
max	NaN	300000.000000	NaN	980.000000	99.990000	NaN	NaN	2016.000000

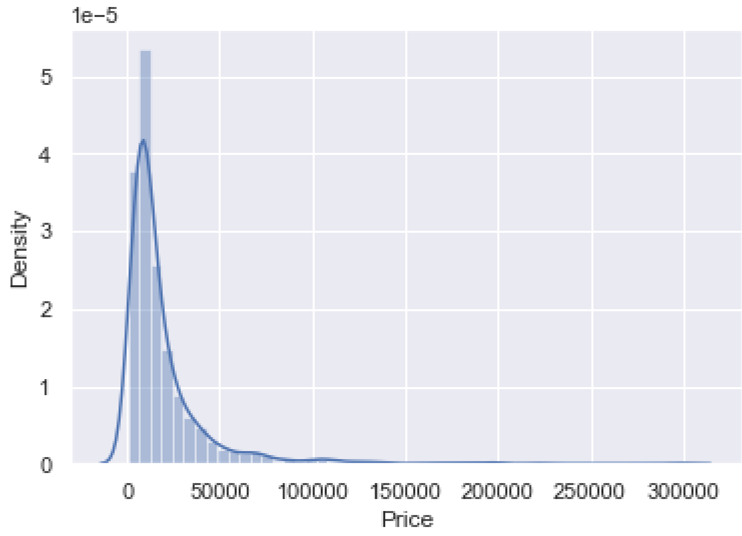
as shown in count raw, all the counts of the data are the same,

exploring PDSs

In [8]:
```
# A great step in the data exploration is to display the probability distribution fun
# The PDF will show us how that variable is distributed
# This makes it very easy to spot anomalies, such as outliers
# The PDF is often the basis on which we decide whether we want to transform a featur
sns.distplot(data['Price'])
```

```
C:\Users\pc\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[8]: <AxesSubplot:xlabel='Price', ylabel='Density'>



dealing with outliers

In [9]:
```
q = data['Price'].quantile(0.99)
```

In [10]:
```
data_1 = data[data['Price'] <q]
```

In [11]:
```
data_1.describe(include = 'all')
```

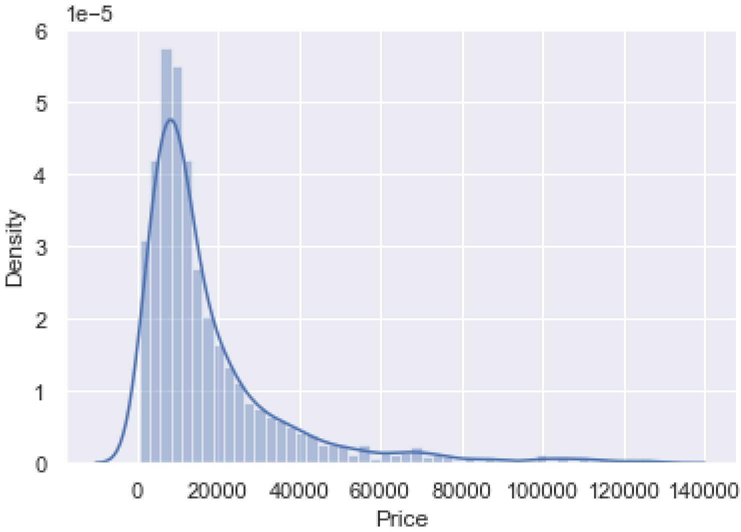
Out[11]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year
count	3984	3984.000000	3984	3984.000000	3984.000000	3984	3984	3984.000000
unique	7	NaN	6	NaN	NaN	4	2	NaN
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	NaN
freq	880	NaN	1528	NaN	NaN	1853	3613	NaN
mean	NaN	17837.117460	NaN	165.116466	2.743770	NaN	NaN	2006.292922
std	NaN	18976.268315	NaN	102.766126	4.956057	NaN	NaN	6.672745
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.000000
25%	NaN	6980.000000	NaN	93.000000	1.800000	NaN	NaN	2002.750000
50%	NaN	11400.000000	NaN	160.000000	2.200000	NaN	NaN	2007.000000
75%	NaN	21000.000000	NaN	230.000000	3.000000	NaN	NaN	2011.000000
max	NaN	129222.000000	NaN	980.000000	99.990000	NaN	NaN	2016.000000

In [12]:
```python
sns.distplot(data_1['Price'])
```

```
C:\Users\pc\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

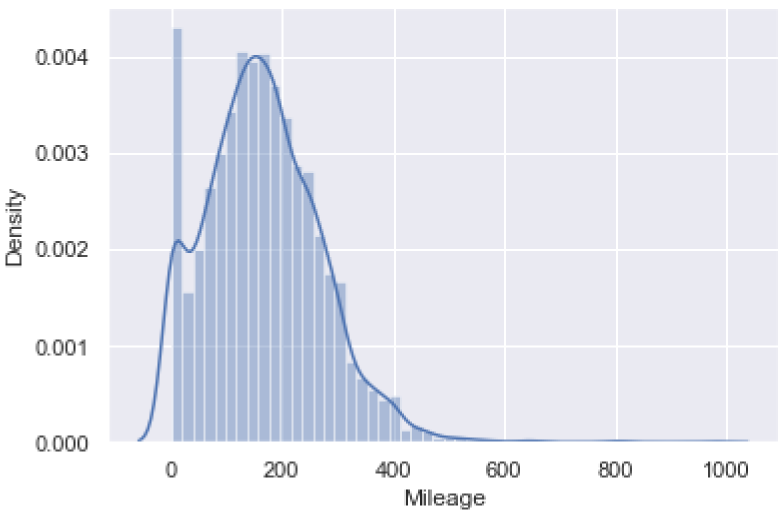
Out[12]: <AxesSubplot:xlabel='Price', ylabel='Density'>



In [13]:
```python
sns.distplot(data_1['Mileage'])
```

```
C:\Users\pc\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[13]: <AxesSubplot:xlabel='Mileage', ylabel='Density'>



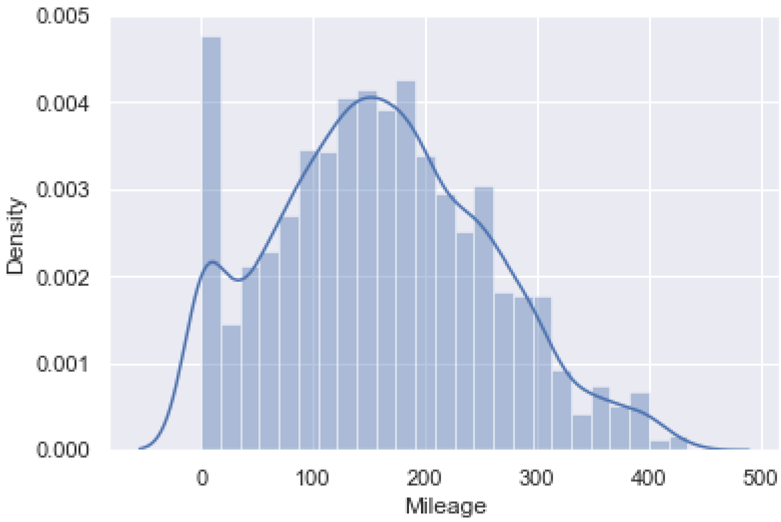
In [14]:
```python
q = data_1['Mileage'].quantile(0.99)
```

In [15]:
```python
data_2 = data_1[data_1['Mileage']<q]
```

In [16]:
```python
sns.distplot(data_2['Mileage'])
```

```
C:\Users\pc\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

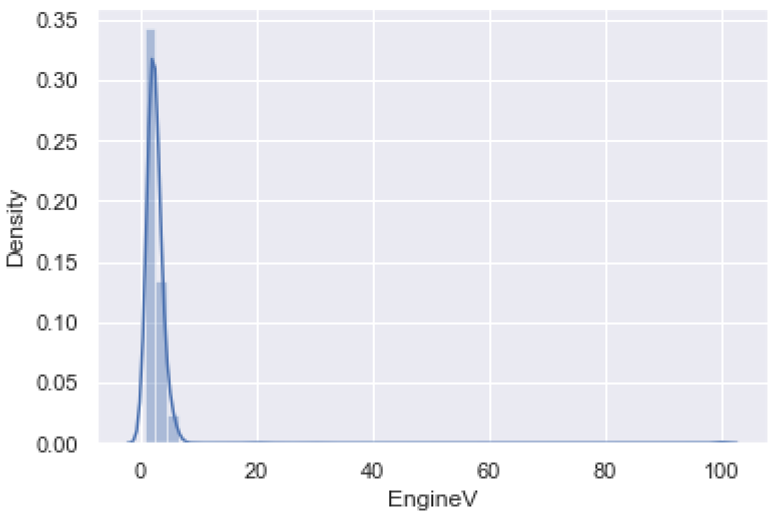
Out[16]:
```
<AxesSubplot:xlabel='Mileage', ylabel='Density'>
```



In [17]:
```python
sns.distplot(data_2['EngineV'])
```

```
C:\Users\pc\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[17]:
```
<AxesSubplot:xlabel='EngineV', ylabel='Density'>
```

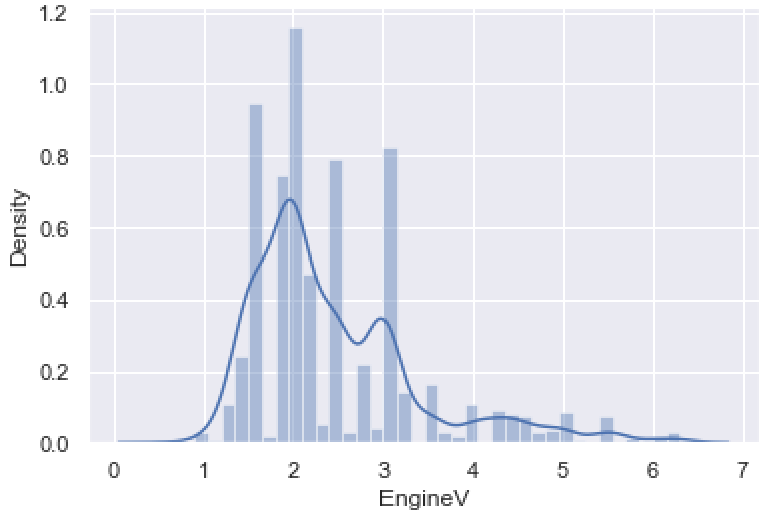


In [18]:
```python
data_3 = data_2[data_2['EngineV']<6.5]
```

In [19]:
```python
sns.distplot(data_3['EngineV'])
```

```
C:\Users\pc\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

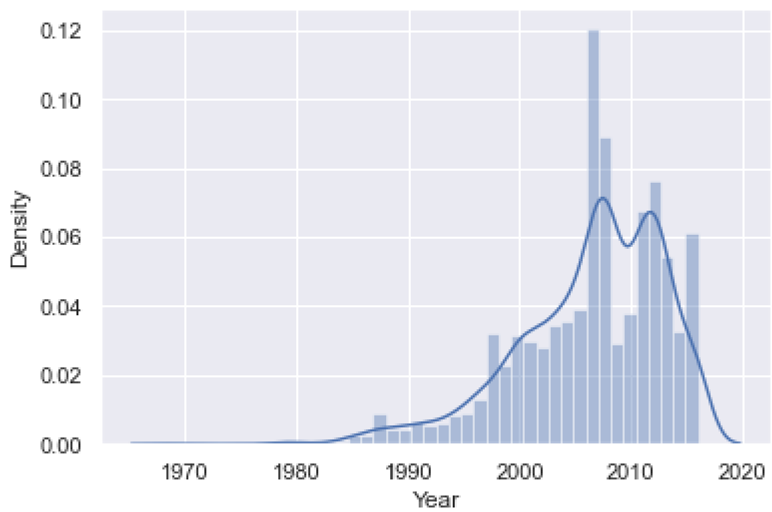
Out[19]: <AxesSubplot:xlabel='EngineV', ylabel='Density'>



In [20]:
```python
sns.distplot(data_3['Year'])
```

```
C:\Users\pc\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[20]: <AxesSubplot:xlabel='Year', ylabel='Density'>



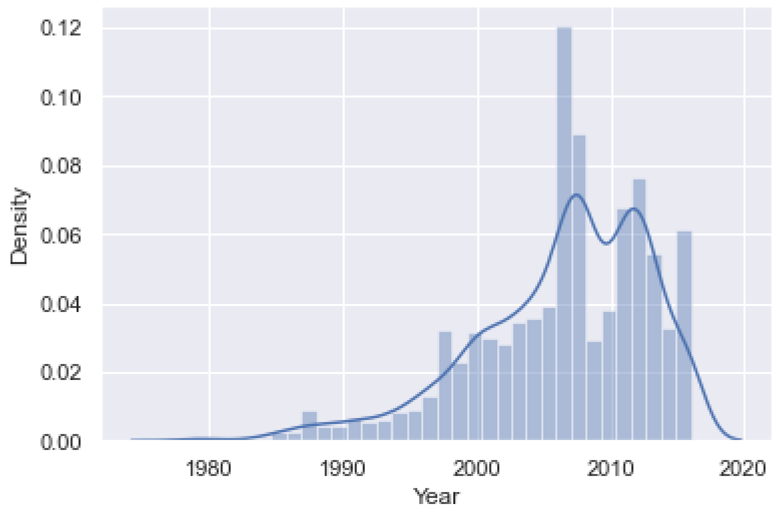
In [21]:
```python
q = data_3['Year'].quantile(0.1)
```

In [22]:
```python
data_4 = data_3[data_3['Year']>1970]
# data_4 = data_3[data_3['Year']>1975]
```

In [23]:
```
sns.distplot(data_4['Year'])
```

```
C:\Users\pc\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[23]: <AxesSubplot:xlabel='Year', ylabel='Density'>



In [24]:
```
data = data_4.reset_index(drop=True)
```

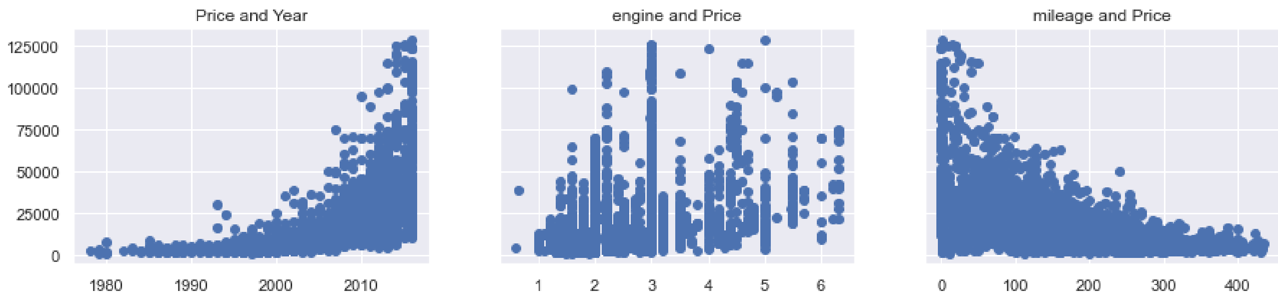
In [25]:
```
data.describe(include = 'all')
```

Out[25]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year
count	3920	3920.000000	3920	3920.000000	3920.000000	3920	3920	3920.000000
unique	7	NaN	6	NaN	NaN	4	2	NaN
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	NaN
freq	862	NaN	1498	NaN	NaN	1818	3558	NaN
mean	NaN	17984.081878	NaN	161.282653	2.443406	NaN	NaN	2006.415561
std	NaN	19042.148809	NaN	96.080356	0.946302	NaN	NaN	6.569588
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1978.000000
25%	NaN	7000.000000	NaN	92.000000	1.800000	NaN	NaN	2003.000000
50%	NaN	11500.000000	NaN	158.000000	2.200000	NaN	NaN	2008.000000
75%	NaN	21500.000000	NaN	229.000000	3.000000	NaN	NaN	2012.000000
max	NaN	129222.000000	NaN	435.000000	6.300000	NaN	NaN	2016.000000

checking ols assumptions

In [26]:
```
# Here we decided to use some matplotlib code, without explaining it
# You can simply use plt.scatter() for each of them (with your current knowledge)
# But since Price is the 'y' axis of all the plots, it made sense to plot them side-b
f, (ax1,ax2,ax3) = plt.subplots(1,3, sharey = True , figsize = (15,3))#sharey -> shar
ax1.scatter(data['Year'] , data['Price'])
ax1.set_title('Price and Year')
ax2.scatter(data['EngineV'], data['Price'])
ax2.set_title('engine and Price')
ax3.scatter(data['Mileage'], data['Price'])
ax3.set_title('mileage and Price')
plt.show()
```



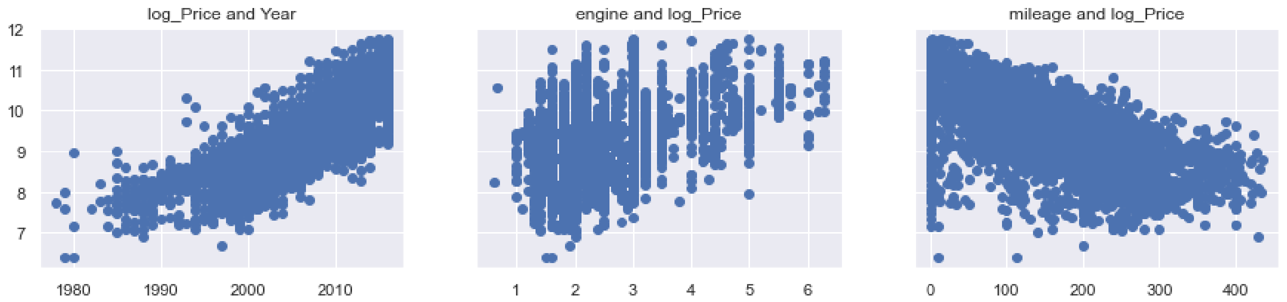
In [27]:
```
# Let's transform 'Price' with a log transformation
log_price = np.log(data['Price'])
# Then we add it to our data frame
data['log_Price']= log_price
data
```

Out[27]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year	log_Price
0	BMW	4200.0	sedan	277	2.0	Petrol	yes	1991	8.342840
1	Mercedes-Benz	7900.0	van	427	2.9	Diesel	yes	1999	8.974618
2	Mercedes-Benz	13300.0	sedan	358	5.0	Gas	yes	2003	9.495519
3	Audi	23000.0	crossover	240	4.2	Petrol	yes	2007	10.043249
4	Toyota	18300.0	crossover	120	2.0	Petrol	yes	2011	9.814656
...	...	...	...	...	...	...	...	...	...
3915	Toyota	17900.0	sedan	35	1.6	Petrol	yes	2014	9.792556
3916	Mercedes-Benz	125000.0	sedan	9	3.0	Diesel	yes	2014	11.736069
3917	BMW	6500.0	sedan	1	3.5	Petrol	yes	1999	8.779557
3918	BMW	8000.0	sedan	194	2.0	Petrol	yes	1985	8.987197
3919	Volkswagen	13500.0	van	124	2.0	Diesel	yes	2013	9.510445

3920 rows × 9 columns

In [28]:
```
#PLOT WITH LOG OF PRICE
f, (ax1,ax2,ax3) = plt.subplots(1,3, sharey = True , figsize = (15,3))#sharey -> shar
ax1.scatter(data['Year'] , data['log_Price'])
ax1.set_title('log_Price and Year')
ax2.scatter(data['EngineV'], data['log_Price'])
ax2.set_title('engine and log_Price')
ax3.scatter(data['Mileage'], data['log_Price'])
ax3.set_title('mileage and log_Price')
plt.show()
```



now we can se a liner relationship in all 3 plots

In [29]:
```
data = data.drop(['Price'], axis = 1)
```

In [30]:
```
data
```

Out[30]:

	Brand	Body	Mileage	EngineV	Engine Type	Registration	Year	log_Price
0	BMW	sedan	277	2.0	Petrol	yes	1991	8.342840
1	Mercedes-Benz	van	427	2.9	Diesel	yes	1999	8.974618
2	Mercedes-Benz	sedan	358	5.0	Gas	yes	2003	9.495519
3	Audi	crossover	240	4.2	Petrol	yes	2007	10.043249
4	Toyota	crossover	120	2.0	Petrol	yes	2011	9.814656
...	...	...	...	...	...	...	...	...
3915	Toyota	sedan	35	1.6	Petrol	yes	2014	9.792556
3916	Mercedes-Benz	sedan	9	3.0	Diesel	yes	2014	11.736069
3917	BMW	sedan	1	3.5	Petrol	yes	1999	8.779557
3918	BMW	sedan	194	2.0	Petrol	yes	1985	8.987197
3919	Volkswagen	van	124	2.0	Diesel	yes	2013	9.510445

3920 rows × 8 columns

multicolliniarity check:

In [32]:
```
data.columns.values
```

Out[32]:
```
array(['Brand', 'Body', 'Mileage', 'EngineV', 'Engine Type',
       'Registration', 'Year', 'log_Price'], dtype=object)
```

In [33]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [35]:
```python
# To make this as easy as possible to use, we declare a variable where we put
# all features where we want to check for multicollinearity
# since our categorical data is not yet preprocessed, we will only take the numerical
variables = data[['Mileage','Year','EngineV']]
```

In [36]:
```python
vif = pd.DataFrame()
```

In [37]:
```python
vif['VIF'] = [variance_inflation_factor(variables.values, i ) for i in range(variable
```

In [38]:
```python
vif['Features'] = variables.columns
```

In [39]:
```python
vif
```

Out[39]:

	VIF	Features
0	3.790463	Mileage
1	10.394632	Year
2	7.669290	EngineV

In [40]:
```python
# Since Year has the highest VIF, I will remove it from the model
# This will drive the VIF of other variables down!!!
# So even if EngineV seems with a high VIF, too, once 'Year' is gone that will no lon
data_no_multicollinearity = data.drop(['Year'],axis=1)
```

create dummy variables to preprocess all categorical features

In [41]:
```python
data_dummy = pd.get_dummies(data_no_multicollinearity,drop_first = True)
```

In [42]:
```python
data_dummy
```

Out[42]:

	Mileage	EngineV	log_Price	Brand_BMW	Brand_Mercedes-Benz	Brand_Mitsubishi	Brand_Renault	Bra
0	277	2.0	8.342840	1	0	0	0	
1	427	2.9	8.974618	0	1	0	0	
2	358	5.0	9.495519	0	1	0	0	
3	240	4.2	10.043249	0	0	0	0	
4	120	2.0	9.814656	0	0	0	0	
...	...	...	...	...	...	...	...	
3915	35	1.6	9.792556	0	0	0	0	
3916	9	3.0	11.736069	0	1	0	0	
3917	1	3.5	8.779557	1	0	0	0	
3918	194	2.0	8.987197	1	0	0	0	
3919	124	2.0	9.510445	0	0	0	0	

3920 rows × 18 columns

In [44]:
```
variables_dummy = data_dummy[['Brand_BMW' , 'Brand_Mercedes-Benz' , 'Brand_Mitsubishi
vif_dummy = pd.DataFrame()
```

In [45]:
```
vif_dummy['VIF'] = [variance_inflation_factor(variables_dummy.values, i ) for i in ra
```

```
C:\Users\pc\anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:1738:
RuntimeWarning: divide by zero encountered in double_scalars
  return 1 - self.ssr/self.uncentered_tss
```

In [47]:
```
vif_dummy['Features'] = variables_dummy.columns
```

In [48]:
```
vif_dummy
```

Out[48]:

	VIF	Features
0	0.187302	Brand_BMW
1	0.000000	Brand_Mercedes-Benz
2	0.160656	Brand_Mitsubishi
3	0.412844	Brand_Renault
4	0.491054	Brand_Toyota
5	0.109049	Brand_Volkswagen

In [50]:
```
data_dummy.columns.values
```

Out[50]:
```
array(['Mileage', 'EngineV', 'log_Price', 'Brand_BMW',
       'Brand_Mercedes-Benz', 'Brand_Mitsubishi', 'Brand_Renault',
       'Brand_Toyota', 'Brand_Volkswagen', 'Body_hatch', 'Body_other',
       'Body_sedan', 'Body_vagon', 'Body_van', 'Engine Type_Gas',
       'Engine Type_Other', 'Engine Type_Petrol', 'Registration_yes'],
      dtype=object)
```

In [52]:
```
cols = ['log_Price','Mileage', 'EngineV', 'Brand_BMW',
       'Brand_Mercedes-Benz', 'Brand_Mitsubishi', 'Brand_Renault',
       'Brand_Toyota', 'Brand_Volkswagen', 'Body_hatch', 'Body_other',
       'Body_sedan', 'Body_vagon', 'Body_van', 'Engine Type_Gas',
       'Engine Type_Other', 'Engine Type_Petrol', 'Registration_yes']
```

In [53]:
```
final_data = data_dummy[cols]
```

In [55]:
```
final_data
```

Out[55]:

	log_Price	Mileage	EngineV	Brand_BMW	Brand_Mercedes-Benz	Brand_Mitsubishi	Brand_Renault	Bra
0	8.342840	277	2.0	1	0	0	0	
1	8.974618	427	2.9	0	1	0	0	
2	9.495519	358	5.0	0	1	0	0	
3	10.043249	240	4.2	0	0	0	0	
4	9.814656	120	2.0	0	0	0	0	
...	...	...	...	...	...	...	...	
3915	9.792556	35	1.6	0	0	0	0	
3916	11.736069	9	3.0	0	1	0	0	
3917	8.779557	1	3.5	1	0	0	0	
3918	8.987197	194	2.0	1	0	0	0	
3919	9.510445	124	2.0	0	0	0	0	

3920 rows × 18 columns

linear regression model

In [57]:
```
#target is log price
target = final_data['log_Price']
inputs = final_data.drop(['log_Price'], axis = 1)
```

scaling the data

In [58]:
```
# Import the scaling module
from sklearn.preprocessing import StandardScaler

# Create a scaler object
scaler = StandardScaler()
# Fit the inputs (calculate the mean and standard deviation feature-wise)
scaler.fit(inputs)
```

Out[58]:
```
StandardScaler()
```

In [59]:
```
# Scale the features and store them in a new variable (the actual scaling procedure)
inputs_scaled = scaler.transform(inputs)
```

train, test and split

In [60]:
```
# Import the module for the split
from sklearn.model_selection import train_test_split
```

In [61]:
```
x_train,x_test,y_train,y_test = train_test_split(inputs_scaled,target,test_size = 0.2
```

create the regression

In [62]:
```python
# Create a linear regression object
reg = LinearRegression()
# Fit the regression with the scaled TRAIN inputs and targets
reg.fit(x_train,y_train)
```

Out[62]: LinearRegression()

In [63]:
```python
# Let's check the outputs of the regression
# I'll store them in y_hat as this is the 'theoretical' name of the predictions
y_hat = reg.predict(x_train)
```

In [66]:
```python
# The simplest way to compare the targets (y_train) and the predictions (y_hat) is to
# The closer the points to the 45-degree line, the better the prediction
plt.scatter(y_train,y_hat)
# Let's also name the axes
plt.xlabel('Targets (y_train)',size=18)
plt.ylabel('Predictions (y_hat)',size=18)
# Sometimes the plot will have different scales of the x-axis and the y-axis
# This is an issue as we won't be able to interpret the '45-degree line'
# We want the x-axis and the y-axis to be the same
plt.xlim(6,13)
plt.ylim(6,13)
plt.show()
```

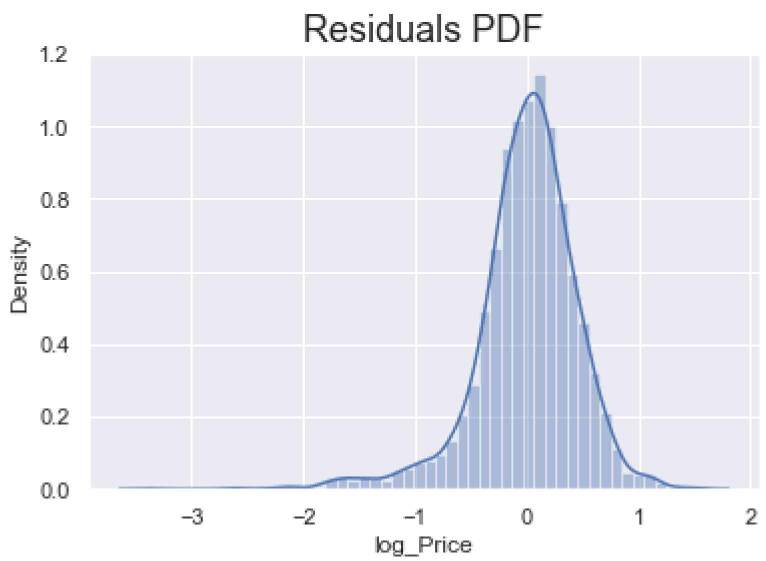
In [67]:
```python
# Another useful check of our model is a residual plot
# We can plot the PDF of the residuals and check for anomalies
sns.distplot(y_train - y_hat)

# Include a title
plt.title("Residuals PDF", size=18)

# In the best case scenario this plot should be normally distributed
# In our case we notice that there are many negative residuals (far away from the mea
# Given the definition of the residuals (y_train - y_hat), negative values imply
# that y_hat (predictions) are much higher than y_train (the targets)
# This is food for thought to improve our model
```

```
C:\Users\pc\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[67]:
```
Text(0.5, 1.0, 'Residuals PDF')
```



In [68]:
```python
# Find the R-squared of the model
reg.score(x_train,y_train)

# Note that this is NOT the adjusted R-squared
# in other words... find the Adjusted R-squared to have the appropriate measure :)
```

Out[68]:
```
0.7400068462570588
```

In [106]:
```python
inputs.columns
```

Out[106]:
```
Index(['Mileage', 'EngineV', 'Brand_BMW', 'Brand_Mercedes-Benz',
       'Brand_Mitsubishi', 'Brand_Renault', 'Brand_Toyota', 'Brand_Volkswagen',
       'Body_hatch', 'Body_other', 'Body_sedan', 'Body_vagon', 'Body_van',
       'Engine Type_Gas', 'Engine Type_Other', 'Engine Type_Petrol',
       'Registration_yes'],
      dtype='object')
```

In [107]:
```
inputs
```

Out[107]:

	Mileage	EngineV	Brand_BMW	Brand_Mercedes-Benz	Brand_Mitsubishi	Brand_Renault	Brand_Toyota
0	277	2.0	1	0	0	0	0
1	427	2.9	0	1	0	0	0
2	358	5.0	0	1	0	0	0
3	240	4.2	0	0	0	0	0
4	120	2.0	0	0	0	0	1
...	...	...	...	...	...	...	...
3915	35	1.6	0	0	0	0	1
3916	9	3.0	0	1	0	0	0
3917	1	3.5	1	0	0	0	0
3918	194	2.0	1	0	0	0	0
3919	124	2.0	0	0	0	0	0

3920 rows × 17 columns

In [111]:
```
x_new = pd.DataFrame({'Mileage':[99] ,'EngineV':[2.5] , 'Brand_BMW' : [0],'Brand_Merc
'Brand_Renault':[0],'Brand_Toyota': [0]  , 'Brand_Volkswagen':[0]  , 'Body_hatch' :[0
'Body_vagon':[0] , 'Body_van':[0] , 'Engine Type_Gas' :[1],
'Engine Type_Other':[0], 'Engine Type_Petrol':[0], 'Registration_yes':[1] })
```

In [112]:
```
x_new
```

Out[112]:

	Mileage	EngineV	Brand_BMW	Brand_Mercedes-Benz	Brand_Mitsubishi	Brand_Renault	Brand_Toyota	Br
0	99	2.5	0	1	0	0	0	

In [121]:
```
scaler.fit(x_new)
```

Out[121]: StandardScaler()

In [122]:
```
x_scaled = scaler.transform(x_new)
```

In [123]:
```
x_scaled
```

Out[123]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0.]])
```

In [124]:
```
y_pred = reg.predict(x_scaled)
print(np.exp(y_pred))
```

```
[12022.86745248]
```

the prediction for such a car is 12,022 USD

finding weights and bias

In [69]:
```
# Obtain the bias (intercept) of the regression
reg.intercept_
```

Out[69]:
```
9.394565736417933
```

In [70]:
```
# Obtain the weights (coefficients) of the regression
reg.coef_

# Note that they are barely interpretable if at all
```

Out[70]:
```
array([-0.46616709,  0.22648636,  0.01762355,  0.00740655, -0.12760288,
       -0.17458568, -0.05664382, -0.08939368, -0.15751762, -0.10492303,
       -0.20419713, -0.12995513, -0.15374739, -0.12901571, -0.0276273 ,
       -0.15068186,  0.30564701])
```

In [71]:
```
# Create a regression summary where we can compare them with one-another
reg_summary = pd.DataFrame(inputs.columns.values, columns=['Features'])
reg_summary['Weights'] = reg.coef_
reg_summary
```

Out[71]:

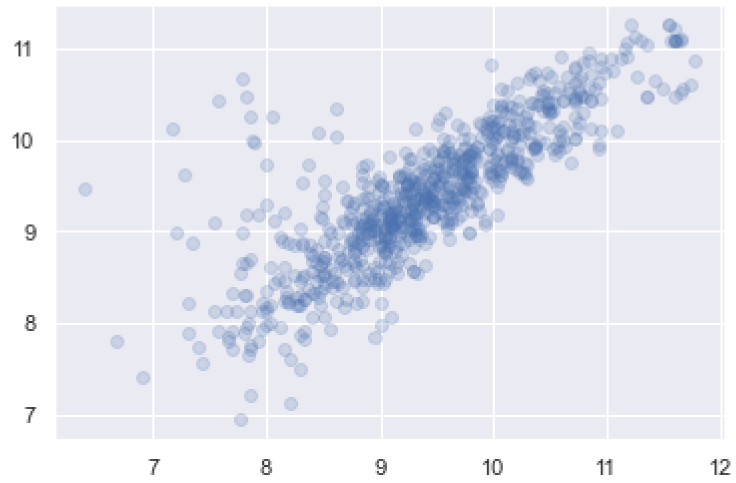
	Features	Weights
0	Mileage	-0.466167
1	EngineV	0.226486
2	Brand_BMW	0.017624
3	Brand_Mercedes-Benz	0.007407
4	Brand_Mitsubishi	-0.127603
5	Brand_Renault	-0.174586
6	Brand_Toyota	-0.056644
7	Brand_Volkswagen	-0.089394
8	Body_hatch	-0.157518
9	Body_other	-0.104923
10	Body_sedan	-0.204197
11	Body_vagon	-0.129955
12	Body_van	-0.153747
13	Engine Type_Gas	-0.129016
14	Engine Type_Other	-0.027627
15	Engine Type_Petrol	-0.150682
16	Registration_yes	0.305647

testing

In [73]:
```
y_hat_test = reg.predict(x_test)
```

In [78]:
```
plt.scatter(y_test,y_hat_test, alpha = 0.2)
```

Out[78]: <matplotlib.collections.PathCollection at 0x15ac6292b50>



In [98]:
```
# Finally, let's manually check these predictions
# To obtain the actual prices, we take the exponential of the log_price
df_pf = pd.DataFrame(np.exp(y_hat_test) , columns = ['Prediction'])
```

In [99]:
```
df_pf
```

Out[99]:

	Prediction
0	52487.914835
1	9278.593572
2	18862.594083
3	10870.554377
4	18037.162787
...	...
779	23244.225004
780	17365.406542
781	2983.876850
782	32005.415924
783	13251.348839

784 rows × 1 columns

In [100]:
```python
# We can also include the test targets in that data frame (so we can manually compare
df_pf['Target'] = np.exp(y_test)
df_pf

# Note that we have a lot of missing values
# There is no reason to have ANY missing values, though
# This suggests that something is wrong with the data frame / indexing
```

Out[100]:

	Prediction	Target
0	52487.914835	51000.0
1	9278.593572	8900.0
2	18862.594083	17800.0
3	10870.554377	13900.0
4	18037.162787	10600.0
...	...	...
779	23244.225004	21800.0
780	17365.406542	24000.0
781	2983.876850	3100.0
782	32005.415924	23500.0
783	13251.348839	11500.0

784 rows × 2 columns

In [101]:
```python
# After displaying y_test, we find what the issue is
# The old indexes are preserved (recall earlier in that code we made a note on that)
# The code was: data_cleaned = data_4.reset_index(drop=True)

# Therefore, to get a proper result, we must reset the index and drop the old indexin
y_test = y_test.reset_index(drop = True)

# Check the result
y_test.head()
```

Out[101]:
```
0    10.839581
1     9.093807
2     9.786954
3     9.539644
4     9.268609
Name: log_Price, dtype: float64
```

In [102]:
```
df_pf['Target'] = np.exp(y_test)
df_pf
```

Out[102]:

	Prediction	Target
0	52487.914835	51000.0
1	9278.593572	8900.0
2	18862.594083	17800.0
3	10870.554377	13900.0
4	18037.162787	10600.0
...	...	...
779	23244.225004	21800.0
780	17365.406542	24000.0
781	2983.876850	3100.0
782	32005.415924	23500.0
783	13251.348839	11500.0

784 rows × 2 columns

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: