Practical Example

Importing the relevant libraries

In [4]:

```
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()
```

Loading the raw data

In [6]:

```
raw_data = pd.read_csv('/Users/saurabhsant/Downloads/1.04. Real-life exa
mple.csv')
raw_data.head()
```

Out[6]:

	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

Preprocessing

Exploring the descriptive statistics of the variables

In [8]:
```
raw_data.describe(include='all')
```

Out[8]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	
count	4345	4173.000000	4345	4345.000000	4195.000000	4345	4345	4345.
unique	7	NaN	6	NaN	NaN	4	2	
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	
freq	936	NaN	1649	NaN	NaN	2019	3947	
mean	NaN	19418.746935	NaN	161.237284	2.790734	NaN	NaN	2006.
std	NaN	25584.242620	NaN	105.705797	5.066437	NaN	NaN	6.
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.
25%	NaN	6999.000000	NaN	86.000000	1.800000	NaN	NaN	2003.
50%	NaN	11500.000000	NaN	155.000000	2.200000	NaN	NaN	2008.
75%	NaN	21700.000000	NaN	230.000000	3.000000	NaN	NaN	2012.
max	NaN	300000.000000	NaN	980.000000	99.990000	NaN	NaN	2016.

In [9]:
```
data = raw_data.drop(['Model'], axis =1)
data.describe(include='all')
```

Out[9]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	
count	4345	4173.000000	4345	4345.000000	4195.000000	4345	4345	4345.
unique	7	NaN	6	NaN	NaN	4	2	
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	
freq	936	NaN	1649	NaN	NaN	2019	3947	
mean	NaN	19418.746935	NaN	161.237284	2.790734	NaN	NaN	2006.
std	NaN	25584.242620	NaN	105.705797	5.066437	NaN	NaN	6.
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.
25%	NaN	6999.000000	NaN	86.000000	1.800000	NaN	NaN	2003.
50%	NaN	11500.000000	NaN	155.000000	2.200000	NaN	NaN	2008.
75%	NaN	21700.000000	NaN	230.000000	3.000000	NaN	NaN	2012.
max	NaN	300000.000000	NaN	980.000000	99.990000	NaN	NaN	2016.

Dealing with missing values

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
Brand             0
Price           172
Body              0
Mileage           0
EngineV         150
Engine Type       0
Registration      0
Year              0
dtype: int64
```

In [11]:
```
data_no_mv = data.dropna(axis=0)
```

In [12]:
```
data_no_mv.describe(include='all')
```

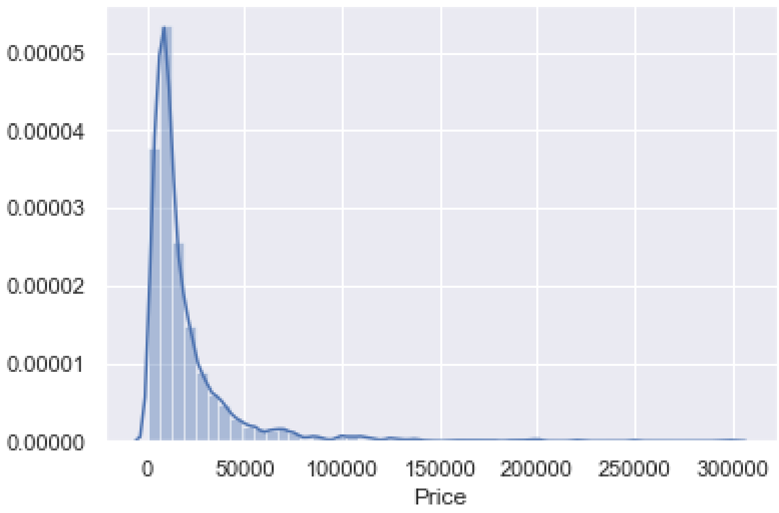
Out[12]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	
count	4025	4025.000000	4025	4025.000000	4025.000000	4025	4025	4025.
unique	7	NaN	6	NaN	NaN	4	2	
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	
freq	880	NaN	1534	NaN	NaN	1861	3654	
mean	NaN	19552.308065	NaN	163.572174	2.764586	NaN	NaN	2006.
std	NaN	25815.734988	NaN	103.394703	4.935941	NaN	NaN	6.
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.
25%	NaN	6999.000000	NaN	90.000000	1.800000	NaN	NaN	2003.
50%	NaN	11500.000000	NaN	158.000000	2.200000	NaN	NaN	2007.
75%	NaN	21900.000000	NaN	230.000000	3.000000	NaN	NaN	2012.
max	NaN	300000.000000	NaN	980.000000	99.990000	NaN	NaN	2016.

Exploring the PDFs

In [14]:
```
sns.distplot(data_no_mv['Price'])
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c1f5444e0>
```



Dealing with outliers

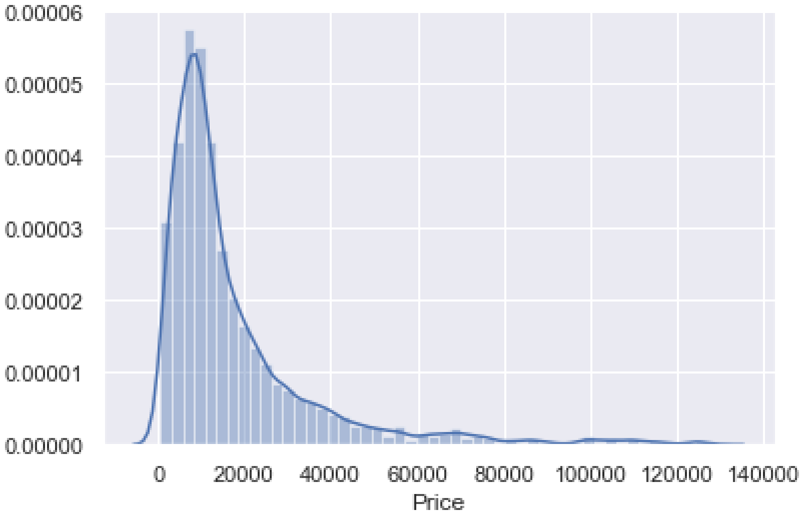
In [16]:
```
q = data_no_mv['Price'].quantile(0.99)
data_1 = data_no_mv[data_no_mv['Price']<q]
data_1.describe(include="all")
```

Out[16]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	
count	3984	3984.000000	3984	3984.000000	3984.000000	3984	3984	3984.0
unique	7	NaN	6	NaN	NaN	4	2	
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	
freq	880	NaN	1528	NaN	NaN	1853	3613	
mean	NaN	17837.117460	NaN	165.116466	2.743770	NaN	NaN	2006.2
std	NaN	18976.268315	NaN	102.766126	4.956057	NaN	NaN	6.6
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.0
25%	NaN	6980.000000	NaN	93.000000	1.800000	NaN	NaN	2002.7
50%	NaN	11400.000000	NaN	160.000000	2.200000	NaN	NaN	2007.0
75%	NaN	21000.000000	NaN	230.000000	3.000000	NaN	NaN	2011.0
max	NaN	129222.000000	NaN	980.000000	99.990000	NaN	NaN	2016.0

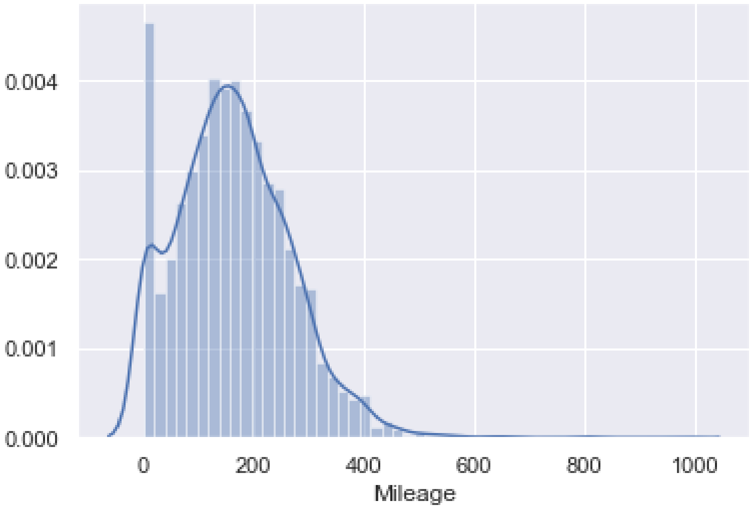
```
In [17]: sns.distplot(data_1['Price'])
```

```
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1fd2b630>
```



```
In [18]: sns.distplot(data_no_mv['Mileage'])
```

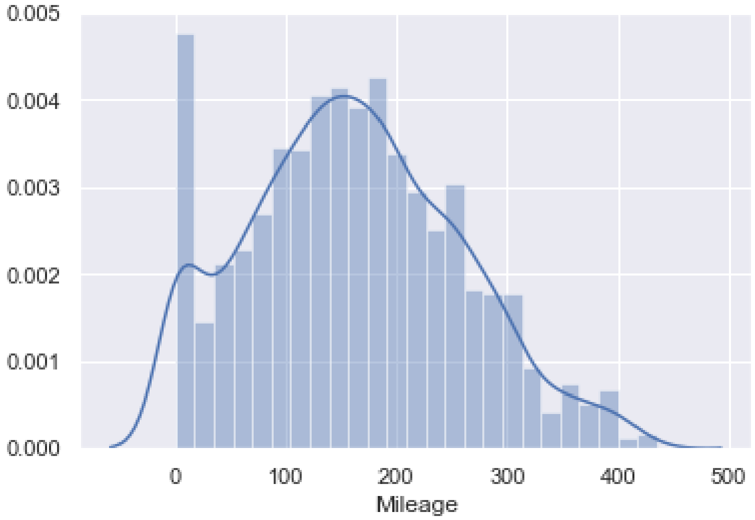
```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1e95c828>
```



```
In [19]: q = data_1['Mileage'].quantile(0.99)
         data_2 = data_1[data_1['Mileage']<q]
```

```
In [20]: sns.distplot(data_2['Mileage'])
```

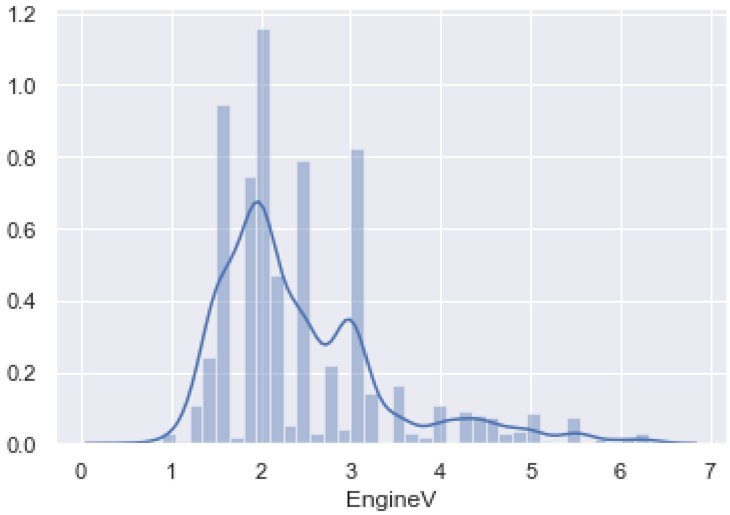
```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1ead6668>
```



```
In [21]: data_3 = data_2[data_2['EngineV']<6.5]
```

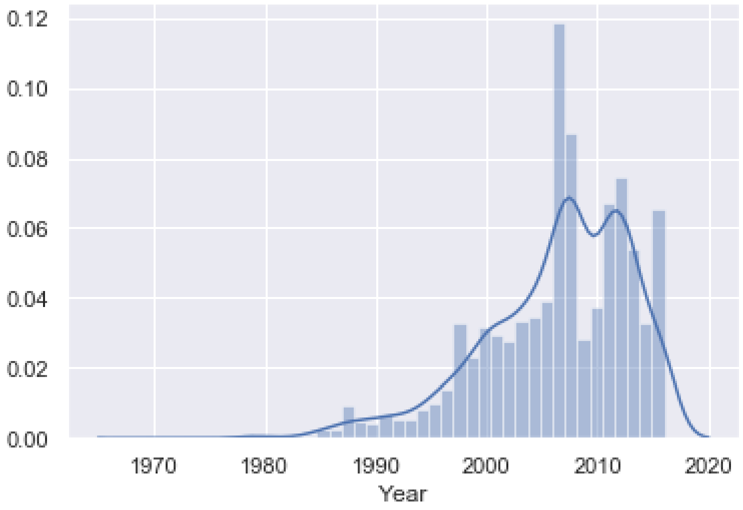
```
In [22]: sns.distplot(data_3['EngineV'])
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1ec0a4e0>
```



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

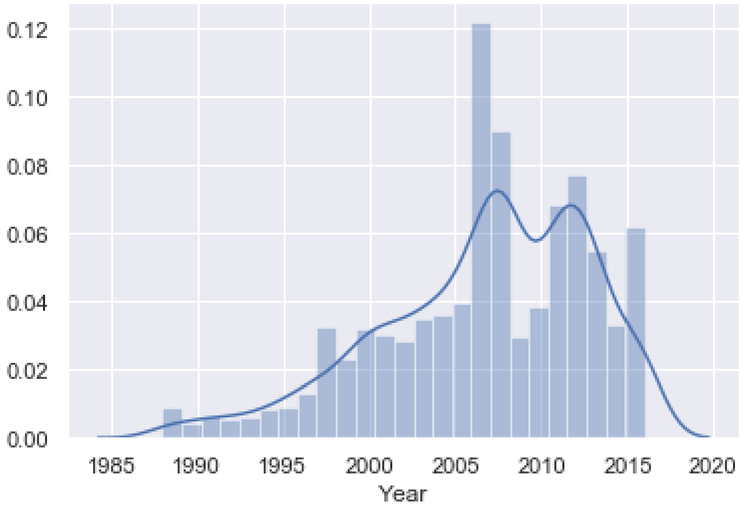
```
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1edb22e8>
```



```
In [24]: q = data_3['Year'].quantile(0.01)
         data_4 = data_3[data_3['Year']>q]
```

```
In [25]: sns.distplot(data_4['Year'])
```

```
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1eecd0f0>
```



```
In [26]: data_cleaned = data_4.reset_index(drop=True)
```

In [27]:
```
data_cleaned.describe(include='all')
```

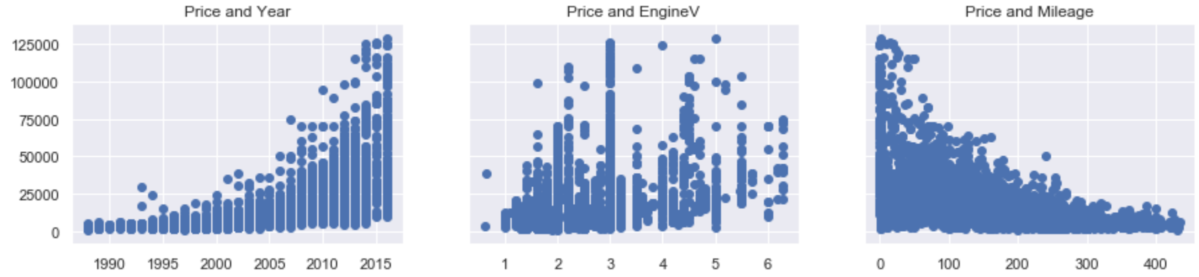
Out[27]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	
count	3867	3867.000000	3867	3867.000000	3867.000000	3867	3867	3867.
unique	7	NaN	6	NaN	NaN	4	2	
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	
freq	848	NaN	1467	NaN	NaN	1807	3505	
mean	NaN	18194.455679	NaN	160.542539	2.450440	NaN	NaN	2006.
std	NaN	19085.855165	NaN	95.633291	0.949366	NaN	NaN	6.
min	NaN	800.000000	NaN	0.000000	0.600000	NaN	NaN	1988.
25%	NaN	7200.000000	NaN	91.000000	1.800000	NaN	NaN	2003.
50%	NaN	11700.000000	NaN	157.000000	2.200000	NaN	NaN	2008.
75%	NaN	21700.000000	NaN	225.000000	3.000000	NaN	NaN	2012.
max	NaN	129222.000000	NaN	435.000000	6.300000	NaN	NaN	2016.

Checking the OLS assumptions

In [28]:
```
f, (ax1, ax2, ax3) = plt.subplots(1, 3, sharey=True, figsize = (15,3))
ax1.scatter(data_cleaned['Year'], data_cleaned['Price'])
ax1.set_title('Price and Year')
ax2.scatter(data_cleaned['EngineV'], data_cleaned['Price'])
ax2.set_title('Price and EngineV')
ax3.scatter(data_cleaned['Mileage'], data_cleaned['Price'])
ax3.set_title('Price and Mileage')

plt.show()
```



Relaxing the assumptions

In [29]:

```
log_price = np.log(data_cleaned['Price'])
data_cleaned['log_price'] = log_price
data_cleaned
```

Out[29]:

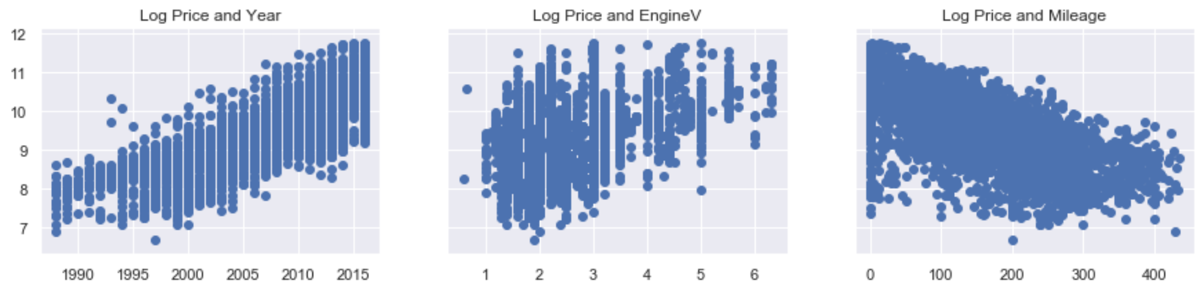
	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year	log_price
0	BMW	4200.0	sedan	277	2.0	Petrol	yes	1991	8.342840
1	Mercedes-Benz	7900.0	van	427	2.9	Diesel	yes	1999	8.974618
2	Mercedes-Benz	13300.0	sedan	358	5.0	Gas	yes	2003	9.495519
3	Audi	23000.0	crossover	240	4.2	Petrol	yes	2007	10.043249
4	Toyota	18300.0	crossover	120	2.0	Petrol	yes	2011	9.814656
...	...	...	...	...	...	...	...	...	...
3862	Volkswagen	11500.0	van	163	2.5	Diesel	yes	2008	9.350102
3863	Toyota	17900.0	sedan	35	1.6	Petrol	yes	2014	9.792556
3864	Mercedes-Benz	125000.0	sedan	9	3.0	Diesel	yes	2014	11.736069
3865	BMW	6500.0	sedan	1	3.5	Petrol	yes	1999	8.779557
3866	Volkswagen	13500.0	van	124	2.0	Diesel	yes	2013	9.510445

3867 rows × 9 columns

In [31]:

```
f, (ax1, ax2, ax3) = plt.subplots(1, 3, sharey=True, figsize = (15,3))
ax1.scatter(data_cleaned['Year'], data_cleaned['log_price'])
ax1.set_title('Log Price and Year')
ax2.scatter(data_cleaned['EngineV'], data_cleaned['log_price'])
ax2.set_title('Log Price and EngineV')
ax3.scatter(data_cleaned['Mileage'], data_cleaned['log_price'])
ax3.set_title('Log Price and Mileage')

plt.show()
```



In [32]:

```
data_cleaned = data_cleaned.drop(['Price'],axis=1)
```

Multicollinearity

In [34]:
```
data_cleaned.columns.values
```

Out[34]:
```
array(['Brand', 'Body', 'Mileage', 'EngineV', 'Engine Type',
       'Registration', 'Year', 'log_price'], dtype=object)
```

In [36]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
variables = data_cleaned[['Mileage', 'Year', 'EngineV']]
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(variables.values, i) for i in range(variables.shape[1])]
vif["features"] = variables.columns
```

In [37]:
```
vif
```

Out[37]:

	VIF	features
0	3.791584	Mileage
1	10.354854	Year
2	7.662068	EngineV

In [38]:
```
data_no_multicollinearity = data_cleaned.drop(['Year'], axis=1)
```

Create dummy variables

In [39]:
```
data_with_dummies = pd.get_dummies(data_no_multicollinearity, drop_first=True)
```

In [40]:
```
data_with_dummies.head()
```

Out[40]:

	Mileage	EngineV	log_price	Brand_BMW	Brand_Mercedes-Benz	Brand_Mitsubishi	Brand_Renault
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

In [ ]:
```

```

In [42]:
```
data_with_dummies.columns.values
```

Out[42]:
```
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 [45]:
```
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']
data_preprocessed = data_with_dummies[cols]
data_preprocessed.head()
```

Out[45]:

	log_price	Mileage	EngineV	Brand_BMW	Brand_Mercedes-Benz	Brand_Mitsubishi	Brand_Renault
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

Linear regression model

Declare the inputs and the targets

In [46]:
```
targets = data_preprocessed['log_price']
inputs = data_preprocessed.drop(['log_price'],axis=1)
```

Scale the data

In [50]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(inputs)
```

Out[50]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [51]:
```
inputs_scaled = scaler.transform(inputs)
```

Train Test Split

In [52]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(inputs_scaled, targets, test_size=0.2,random_state=365)
```

Create the regression

In [53]:

```
reg = LinearRegression()
reg.fit(x_train, y_train)
```

Out[53]:

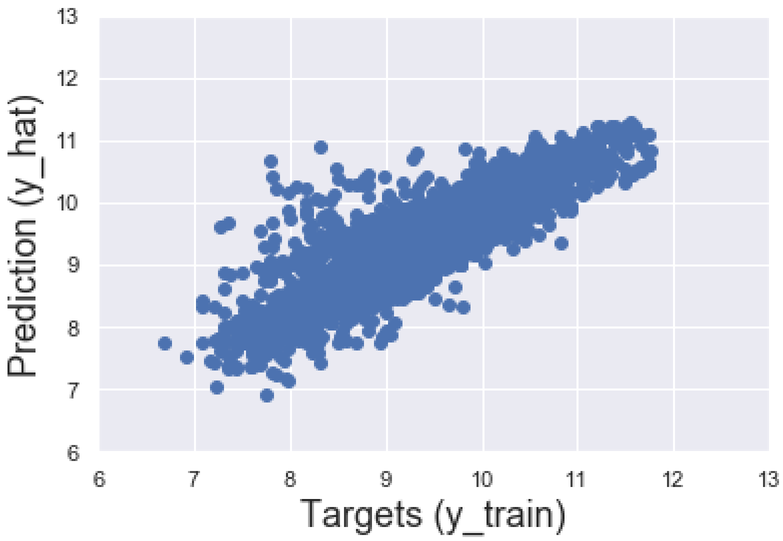
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [54]:

```
y_hat = reg.predict(x_train)
```

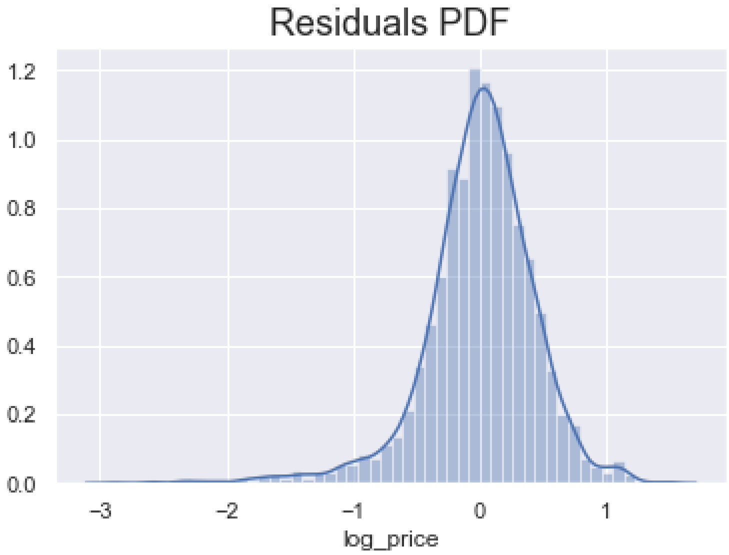
In [56]:

```
plt.scatter(y_train, y_hat)
plt.xlabel('Targets (y_train)', size = 18)
plt.ylabel('Prediction (y_hat)', size = 18)
plt.xlim(6,13)
plt.ylim(6,13)
plt.show()
```



```
In [59]: sns.distplot(y_train - y_hat)
         plt.title("Residuals PDF", size=18)

Out[59]: Text(0.5, 1.0, 'Residuals PDF')
```



```
In [60]: reg.score(x_train, y_train)

Out[60]: 0.744996578792662
```

Finding the weights and bias

```
In [61]: reg.intercept_

Out[61]: 9.415239458021299
```

```
In [62]: reg.coef_

Out[62]: array([-0.44871341,  0.20903483,  0.0142496 ,  0.01288174, -0.14055166,
                -0.17990912, -0.06054988, -0.08992433, -0.1454692 , -0.10144383,
                -0.20062984, -0.12988747, -0.16859669, -0.12149035, -0.03336798,
                -0.14690868,  0.32047333])
```

In [63]:

```
reg_summary = pd.DataFrame(inputs.columns.values, columns=['Features'])
reg_summary['Weights'] = reg.coef_
reg_summary
```

Out[63]:

	Features	Weights
0	Mileage	-0.448713
1	EngineV	0.209035
2	Brand_BMW	0.014250
3	Brand_Mercedes-Benz	0.012882
4	Brand_Mitsubishi	-0.140552
5	Brand_Renault	-0.179909
6	Brand_Toyota	-0.060550
7	Brand_Volkswagen	-0.089924
8	Body_hatch	-0.145469
9	Body_other	-0.101444
10	Body_sedan	-0.200630
11	Body_vagon	-0.129887
12	Body_van	-0.168597
13	Engine Type_Gas	-0.121490
14	Engine Type_Other	-0.033368
15	Engine Type_Petrol	-0.146909
16	Registration_yes	0.320473

In [64]:

```
data_cleaned['Brand'].unique()
```

Out[64]:

```
array(['BMW', 'Mercedes-Benz', 'Audi', 'Toyota', 'Renault', 'Volkswage
n',
       'Mitsubishi'], dtype=object)
```

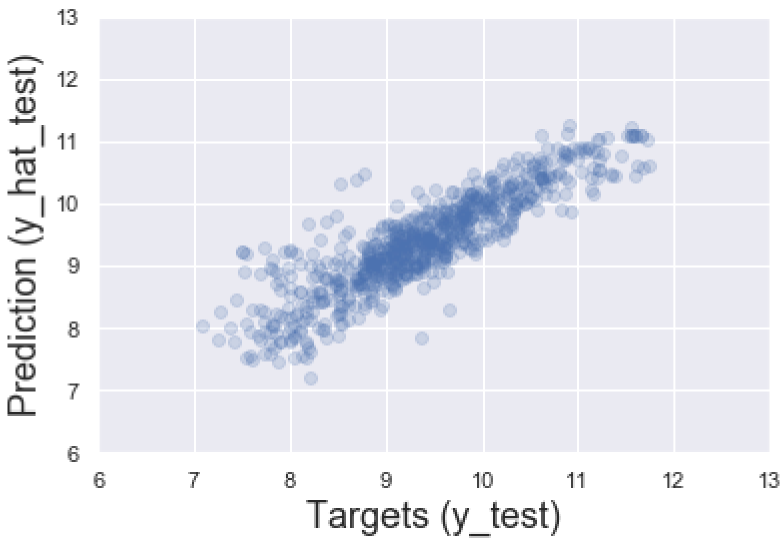
Testing

In [65]:

```
y_hat_test = reg.predict(x_test)
```

In [67]:

```
plt.scatter(y_test, y_hat_test, alpha=0.2)
plt.xlabel('Targets (y_test)', size = 18)
plt.ylabel('Prediction (y_hat_test)', size = 18)
plt.xlim(6,13)
plt.ylim(6,13)
plt.show()
```



In [77]:

```
df_pf = pd.DataFrame(np.exp(y_hat_test), columns=["Prediction"])
df_pf.head()
```

Out[77]:

	Prediction
0	10685.501696
1	3499.255242
2	7553.285218
3	7463.963017
4	11353.490075

In [79]:

```
df_pf['Target'] = np.exp(y_test)
df_pf.head()
```

Out[79]:

	Prediction	Target
0	10685.501696	2300.0
1	3499.255242	2800.0
2	7553.285218	2500.0
3	7463.963017	6400.0
4	11353.490075	9150.0

In [80]:

```
y_test = y_test.reset_index(drop=True)
```

In [81]:
```
df_pf['Residual'] = df_pf['Target'] - df_pf['Prediction']
```

In [82]:
```
df_pf['Difference%'] = np.absolute(df_pf['Residual']/df_pf['Target']*100
)
df_pf
```

Out[82]:

	Prediction	Target	Residual	Difference%
0	10685.501696	2300.0	-8385.501696	364.587030
1	3499.255242	2800.0	-699.255242	24.973402
2	7553.285218	2500.0	-5053.285218	202.131409
3	7463.963017	6400.0	-1063.963017	16.624422
4	11353.490075	9150.0	-2203.490075	24.081859
...	...	...	...	...
769	29651.726363	29500.0	-151.726363	0.514327
770	10732.071179	9600.0	-1132.071179	11.792408
771	13922.446953	18300.0	4377.553047	23.921055
772	27487.751303	68500.0	41012.248697	59.871896
773	13491.163043	10800.0	-2691.163043	24.918176

774 rows × 4 columns

In [83]:
```
df_pf.describe()
```

Out[83]:

	Prediction	Target	Residual	Difference%
count	774.000000	774.000000	774.000000	774.000000
mean	15946.760167	18165.817106	2219.056939	36.256693
std	13133.197604	19967.858908	10871.218143	55.066507
min	1320.562768	1200.000000	-29456.498331	0.062794
25%	7413.644234	6900.000000	-2044.191251	12.108022
50%	11568.168859	11600.000000	142.518577	23.467728
75%	20162.408805	20500.000000	3147.343497	39.563570
max	77403.055224	126000.000000	85106.162329	512.688080

In [85]:

```
#pd.options.display.max_rows = 999
df_pf.sort_values(by=['Difference%'])
```

Out[85]:

	Prediction	Target	Residual	Difference%
698	30480.847838	30500.0	19.152162	0.062794
742	16960.310476	16999.0	38.689524	0.227599
60	12469.207487	12500.0	30.792513	0.246340
110	25614.137960	25500.0	-114.137960	0.447600
367	42703.676996	42500.0	-203.676996	0.479240
...	...	...	...	...
657	32481.045510	6000.0	-26481.045510	441.350758
162	9954.416247	1800.0	-8154.416247	453.023125
451	35956.498331	6500.0	-29456.498331	453.176897
532	10019.903027	1800.0	-8219.903027	456.661279
639	30628.277108	4999.0	-25629.277108	512.688080

774 rows × 4 columns

In [ ]:

```

```