Practical example

Importing the relevant libraries

In [2]:
```
# For this practical example we will need the following libraries and module
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 [3]:
```
# Load the data from a .csv in the same folder
raw_data = pd.read_csv('1.04. Real-life example.csv')

# Let's explore the top 5 rows of the df
raw_data.head()
```

Out[3]:

	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 [4]:

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

# Note that categorical variables don't have some types of numerical descrip
# and numerical variables don't have some types of categorical descriptives
```

Out[4]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	
count	4345	4173.000000	4345	4345.000000	4195.000000	4345	4345	4345.0
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	2006.5
std	NaN	25584.242620	NaN	105.705797	5.066437	NaN	NaN	6.71
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.0
25%	NaN	6999.000000	NaN	86.000000	1.800000	NaN	NaN	2003.0
50%	NaN	11500.000000	NaN	155.000000	2.200000	NaN	NaN	2008.0
75%	NaN	21700.000000	NaN	230.000000	3.000000	NaN	NaN	2012.0
max	NaN	300000.000000	NaN	980.000000	99.990000	NaN	NaN	2016.0

Determining the variables of interest

In [5]:

```python
# For these several lessons, we will create the regression without 'Model'
# Certainly, when you work on the problem on your own, you could create a re
data = raw_data.drop(['Model'],axis=1)

# Let's check the descriptives without 'Model'
data.describe(include='all')
```

Out[5]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	
count	4345	4173.000000	4345	4345.000000	4195.000000	4345	4345	4345.0
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	2006.5
std	NaN	25584.242620	NaN	105.705797	5.066437	NaN	NaN	6.71
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.0
25%	NaN	6999.000000	NaN	86.000000	1.800000	NaN	NaN	2003.0
50%	NaN	11500.000000	NaN	155.000000	2.200000	NaN	NaN	2008.0
75%	NaN	21700.000000	NaN	230.000000	3.000000	NaN	NaN	2012.0
max	NaN	300000.000000	NaN	980.000000	99.990000	NaN	NaN	2016.0

Dealing with missing values

In [6]:

```python
# data.isnull() # shows a df with the information whether a data point is nu
# Since True = the data point is missing, while False = the data point is no
# This will give us the total number of missing values feature-wise
data.isnull().sum()
```

Out[6]:

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

In [7]:

```python
# Let's simply drop all missing values
# This is not always recommended, however, when we remove less than 5% of th
data_no_mv = data.dropna(axis=0)
```

In [8]:
```
# Let's check the descriptives without the missing values
data_no_mv.describe(include='all')
```

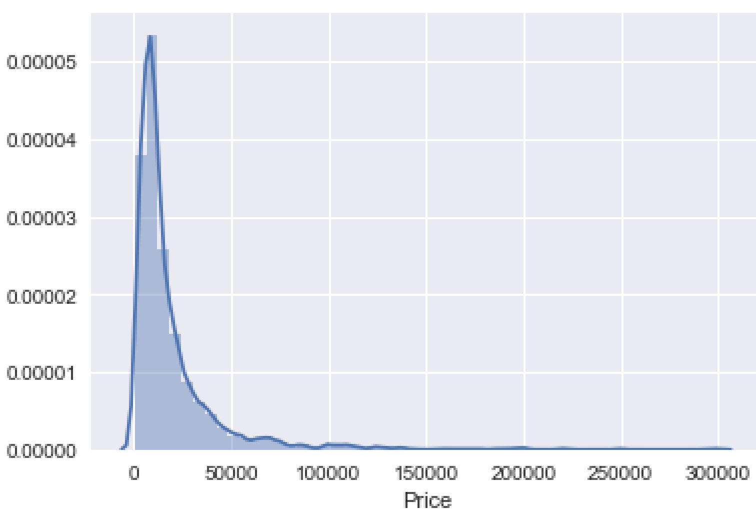
Out[8]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	
count	4025	4025.000000	4025	4025.000000	4025.000000	4025	4025	4025.00
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	2006.37
std	NaN	25815.734988	NaN	103.394703	4.935941	NaN	NaN	6.69
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.00
25%	NaN	6999.000000	NaN	90.000000	1.800000	NaN	NaN	2003.00
50%	NaN	11500.000000	NaN	158.000000	2.200000	NaN	NaN	2007.00
75%	NaN	21900.000000	NaN	230.000000	3.000000	NaN	NaN	2012.00
max	NaN	300000.000000	NaN	980.000000	99.990000	NaN	NaN	2016.00

Exploring the PDFs

In [9]:
```
# A great step in the data exploration is to display the probability distrib
# 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
sns.distplot(data_no_mv['Price'])
```

Out[9]: `<matplotlib.axes._subplots.AxesSubplot at 0x2b897ab4940>`



Dealing with outliers

In [10]:

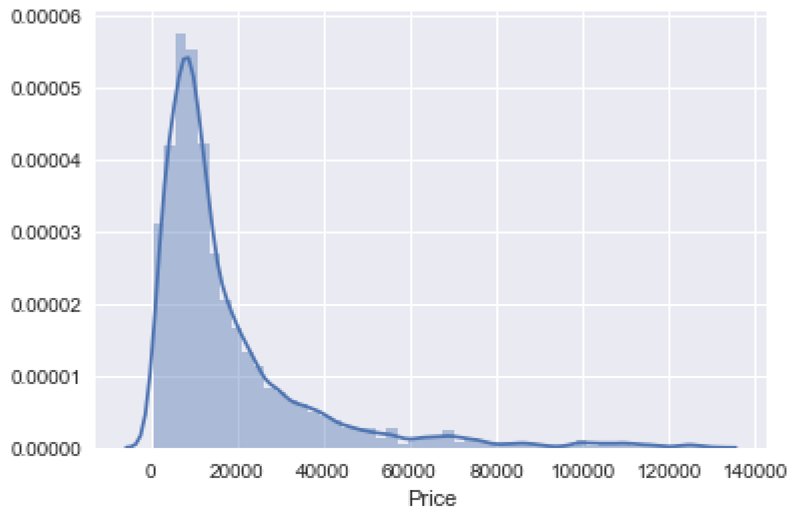
```
# Obviously there are some outliers present

# Without diving too deep into the topic, we can deal with the problem easil
# Here, the outliers are situated around the higher prices (right side of th
# Logic should also be applied
# This is a dataset about used cars, therefore one can imagine how $300,000

# Outliers are a great issue for OLS, thus we must deal with them in some wa
# It may be a useful exercise to try training a model without removing the o

# Let's declare a variable that will be equal to the 99th percentile of the
q = data_no_mv['Price'].quantile(0.99)
# Then we can create a new df, with the condition that all prices must be be
data_1 = data_no_mv[data_no_mv['Price']<q]
# In this way we have essentially removed the top 1% of the data about 'Pric
data_1.describe(include='all')
```

Out[10]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	
count	3984	3984.000000	3984	3984.000000	3984.000000	3984	3984	3984.00
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	2006.29
std	NaN	18976.268315	NaN	102.766126	4.956057	NaN	NaN	6.67
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.00
25%	NaN	6980.000000	NaN	93.000000	1.800000	NaN	NaN	2002.75
50%	NaN	11400.000000	NaN	160.000000	2.200000	NaN	NaN	2007.00
75%	NaN	21000.000000	NaN	230.000000	3.000000	NaN	NaN	2011.00
max	NaN	129222.000000	NaN	980.000000	99.990000	NaN	NaN	2016.00

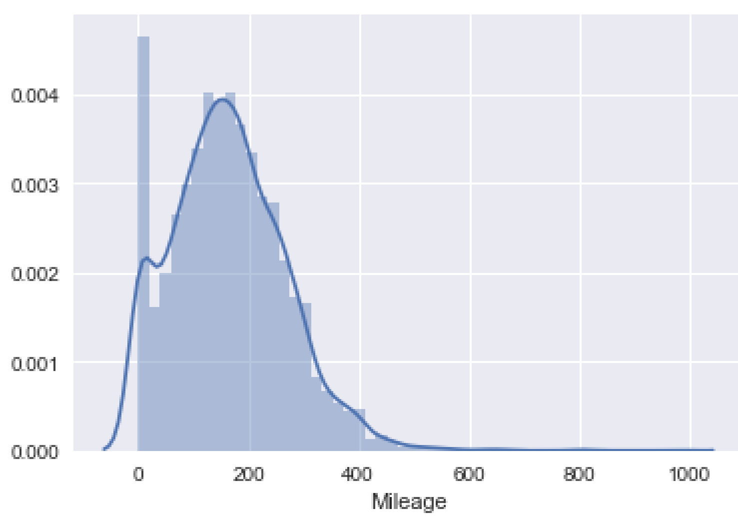
In [11]:
```
# We can check the PDF once again to ensure that the result is still distrib
# however, there are much fewer outliers
sns.distplot(data_1['Price'])
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x2b897eacfd0>



In [12]:
```
# We can treat the other numerical variables in a similar way
sns.distplot(data_no_mv['Mileage'])
```

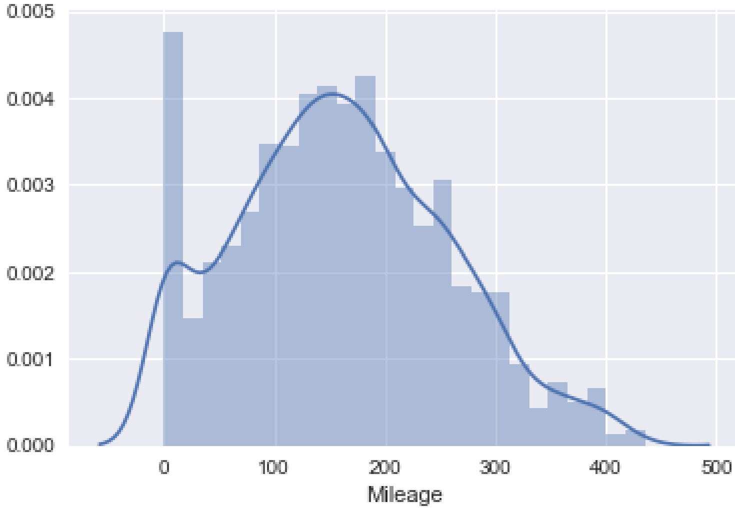
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x2b897e36e80>



In [13]:
```
q = data_1['Mileage'].quantile(0.99)
data_2 = data_1[data_1['Mileage']<q]
```

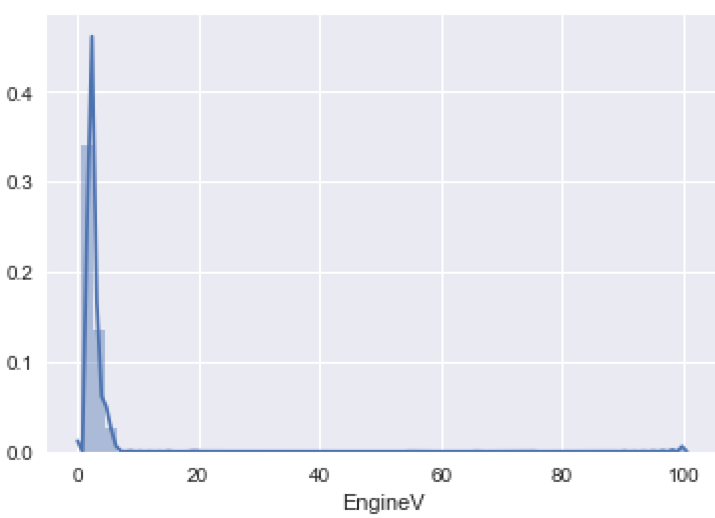
In [14]:
```
# This plot looks kind of normal, doesn't it?
sns.distplot(data_2['Mileage'])
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x2b8980359b0>



In [15]:
```
# The situation with engine volume is very strange
# In such cases it makes sense to manually check what may be causing the pro
# In our case the issue comes from the fact that most missing values are ind
# There are also some incorrect entries like 75
sns.distplot(data_no_mv['EngineV'])
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x2b897f681d0>

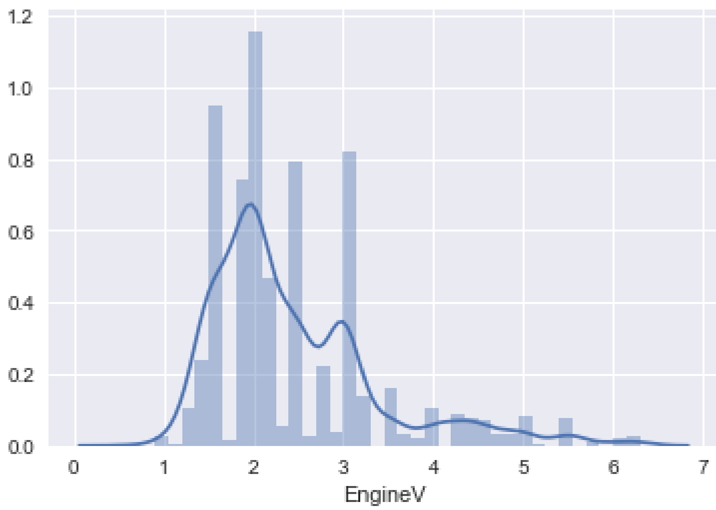


In [16]:
```
# A simple Google search can indicate the natural domain of this variable
# Car engine volumes are usually (always?) below 6.5l
# This is a prime example of the fact that a domain expert (a person working
# may find it much easier to determine problems with the data than an outsid
data_3 = data_2[data_2['EngineV']<6.5]
```

In [17]:

```
# Following this graph, we realize we can actually treat EngineV as a catego
# Even so, in this course we won't, but that's yet something else you may tr
sns.distplot(data_3['EngineV'])
```

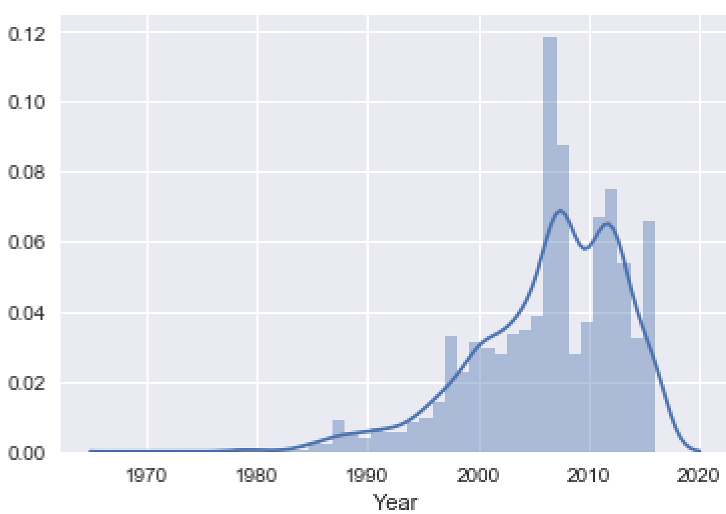
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x2b8981a0b00>



In [18]:

```
# Finally, the situation with 'Year' is similar to 'Price' and 'Mileage'
# However, the outliers are on the low end
sns.distplot(data_no_mv['Year'])
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x2b89825e6a0>

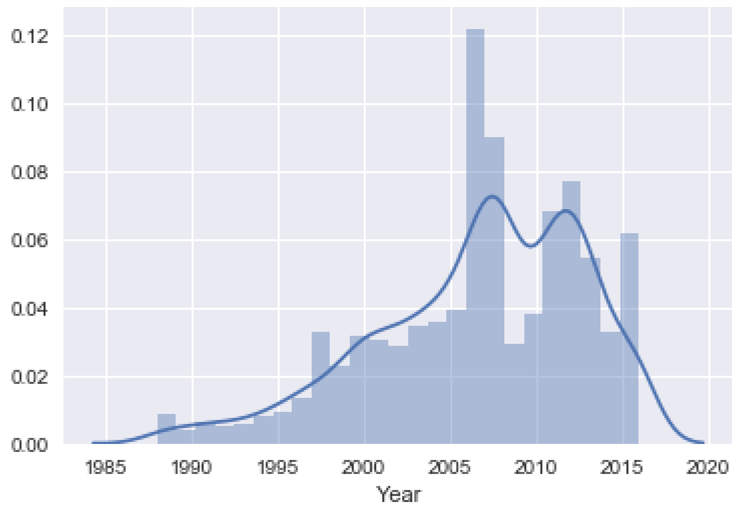


In [19]:

```
# I'll simply remove them
q = data_3['Year'].quantile(0.01)
data_4 = data_3[data_3['Year']>q]
```

In [20]:
```python
# Here's the new result
sns.distplot(data_4['Year'])
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x2b898296d30>



In [21]:
```python
# When we remove observations, the original indexes are preserved
# If we remove observations with indexes 2 and 3, the indexes will go as: 0,
# That's very problematic as we tend to forget about it (later you will see

# Finally, once we reset the index, a new column will be created containing
# We won't be needing it, thus 'drop=True' to completely forget about it
data_cleaned = data_4.reset_index(drop=True)
```

In [22]:
```python
# Let's see what's left
data_cleaned.describe(include='all')
```

Out[22]:

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	
count	3867	3867.000000	3867	3867.000000	3867.000000	3867	3867	3867.0
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	2006.7
std	NaN	19085.855165	NaN	95.633291	0.949366	NaN	NaN	6.1
min	NaN	800.000000	NaN	0.000000	0.600000	NaN	NaN	1988.0
25%	NaN	7200.000000	NaN	91.000000	1.800000	NaN	NaN	2003.0
50%	NaN	11700.000000	NaN	157.000000	2.200000	NaN	NaN	2008.0
75%	NaN	21700.000000	NaN	225.000000	3.000000	NaN	NaN	2012.0
max	NaN	129222.000000	NaN	435.000000	6.300000	NaN	NaN	2016.0