Results Screenshots:

car-price-prediction-regression.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

```
### Looking at the datatypes of the dataset

dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19237 entries, 0 to 19236
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                19237 non-null  int64  
 1   Price             19237 non-null  int64  
 2   Levy              19237 non-null  object 
 3   Manufacturer      19237 non-null  object 
 4   Model             19237 non-null  object 
 5   Prod. year        19237 non-null  int64  
 6   Category          19237 non-null  object 
 7   Leather interior  19237 non-null  object 
 8   Fuel type         19237 non-null  object 
 9   Engine volume     19237 non-null  object 
 10  Mileage           19237 non-null  object 
 11  Cylinders         19237 non-null  float64
 12  Gear box type     19237 non-null  object 
 13  Drive wheels      19237 non-null  object 
 14  Doors             19237 non-null  object 
 15  Wheel             19237 non-null  object 
 16  Color             19237 non-null  object 
 17  Airbags           19237 non-null  int64  
dtypes: float64(1), int64(4), object(13)
memory usage: 2.6+ MB
```

Here, the columns - Manufacturer, Model, Category, Leather interior, Fuel type, Gear box type, Drive wheels, Doors, Wheel, Color are categorical.

[15]

```
### Modifying the datatypes of the columns to category

dataset.Manufacturer = dataset.Manufacturer.astype('category')
dataset.Model = dataset.Model.astype('category')
dataset.Category = dataset.Category.astype('category')
dataset["Leather interior"] = dataset["Leather interior"].astype('category')
dataset["Fuel type"] = dataset["Fuel type"].astype('category')
dataset["Gear box type"] = dataset["Gear box type"].astype('category')
dataset["Drive wheels"] = dataset["Drive wheels"].astype('category')
dataset.Doors = dataset.Doors.astype('category')
dataset.Wheel = dataset.Wheel.astype('category')
dataset.Color = dataset.Color.astype('category')
```

Looking at the modified datatypes of the columns in the dataset.

```
### Looking at the modified datatypes of the dataset

dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19237 entries, 0 to 19236
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                19237 non-null  int64
 1   Price             19237 non-null  int64
 2   Levy              19237 non-null  object
 3   Manufacturer      19237 non-null  category
 4   Model             19237 non-null  category
 5   Prod. year        19237 non-null  int64
 6   Category          19237 non-null  category
 7   Leather interior  19237 non-null  category
 8   Fuel type         19237 non-null  category
 9   Engine volume     19237 non-null  object
 10  Mileage           19237 non-null  object
 11  Cylinders         19237 non-null  float64
 12  Gear box type     19237 non-null  category
 13  Drive wheels      19237 non-null  category
 14  Doors             19237 non-null  category
 15  Wheel             19237 non-null  category
 16  Color             19237 non-null  category
 17  Airbags           19237 non-null  int64
dtypes: category(10), float64(1), int64(4), object(3)
memory usage: 1.5+ MB
```

```
### Missing values (-) in the dataset

print('Missing values in the dataset:\n')
for each_column in dataset.columns:
    print('Column: {} - {}'.format(each_column, list(dataset[each_column]).count('-')))
```

```
Missing values in the dataset:

Column: ID - 0
Column: Price - 0
Column: Levy - 5819
Column: Manufacturer - 0
Column: Model - 0
Column: Prod. year - 0
Column: Category - 0
Column: Leather interior - 0
Column: Fuel type - 0
Column: Engine volume - 0
Column: Mileage - 0
Column: Cylinders - 0
Column: Gear box type - 0
Column: Drive wheels - 0
Column: Doors - 0
Column: Wheel - 0
Column: Color - 0
Column: Airbags - 0
```

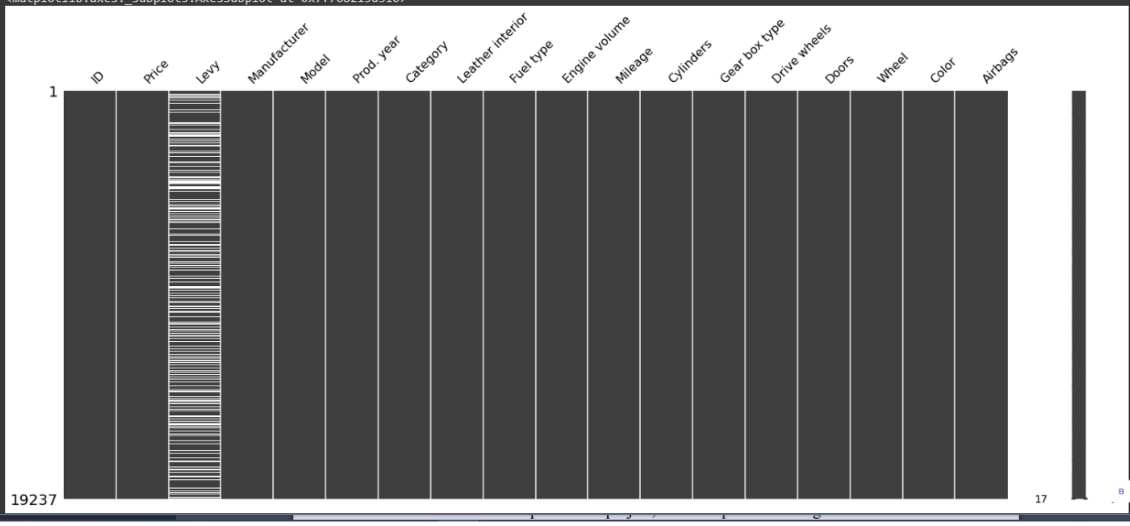
From the above dataset, we can see that there are missing values in the column - Levy.

+ Code + Text

```
### Visual representation of the missing data in the dataset

missingno.matrix(dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f708213d310>
```




```
def detect_outliers(df, n, features_list):
    outlier_indices = []
    for feature in features_list:
        Q1 = np.percentile(df[feature], 25)
        Q3 = np.percentile(df[feature], 75)
        IQR = Q3 - Q1
        outlier_step = 1.5 * IQR
        outlier_list_col = df[(df[feature] < Q1 - outlier_step) | (df[feature] > Q3 + outlier_step)].index
        outlier_indices.extend(outlier_list_col)
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list(key for key, value in outlier_indices.items() if value > n)
    return multiple_outliers

outliers_to_drop = detect_outliers(dataset, 2, ['Price', 'Levy', 'Prod. year', 'Mileage', 'Cylinders', 'Airbags',
                                                'Engine volume'])
print("We will drop these {} indices: ".format(len(outliers_to_drop)), outliers_to_drop)
```

```
We will drop these 270 indices:  [90, 211, 420, 483, 573, 579, 723, 724, 747, 1019, 1083, 1128, 1225, 1364, 1459, 1490, 1509, 1510, 1562, 1662, 1704, 1823, 201
```

Now let's look at the data present in the rows.

```
[69] dataset.iloc[outliers_to_drop, :]
```

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Drive wheels	Doors	Wheel	Color	Air
90	45807330	77775	1604.0	MERCEDES-BENZ	GL 63 AMG	2014	Jeep	Yes	Petrol	5.5	433811.0	8.0	Automatic	4x4	04-May	Left wheel	Black	
211	45156280	72130	1885.0	PORSCHE	Panamera	2010	Hatchback	Yes	Petrol	4.8	196800.0	8.0	Tiptronic	4x4	04-May	Left wheel	White	
420	45763904	81539	1935.0	LEXUS	GX 460	2016	Jeep	Yes	Petrol	4.6	155976.0	8.0	Automatic	4x4	04-May	Left wheel	White	
483	45761340	69935	1646.0	LEXUS	GX 470	2015	Jeep	Yes	Petrol	4.6	273493.0	8.0	Automatic	4x4	04-May	Left wheel	Silver	
573	45731517	119172	1301.0	BMW	M6	2014	Coupe	Yes	Petrol	4.4	33500.0	8.0	Tiptronic	Rear	04-May	Left wheel	White	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7272	45416515	35438	NaN	BMW	X6	2009	Jeep	Yes	Petrol	4.4	960000.0	8.0	Tiptronic	4x4	04-May	Left wheel	Silver	
9114	45813297	31988	3015.0	FERRARI	F50	2017	Coupe	Yes	Petrol	6.3	419200.0	12.0	Automatic	Rear	02-Mar	Left wheel	Silver	
10973	45416515	35438	NaN	BMW	X6	2009	Jeep	Yes	Petrol	4.4	960000.0	8.0	Tiptronic	4x4	04-May	Left wheel	Silver	
13850	45796827	1000	NaN	MERCEDES-BENZ	GLS 63 AMG	2014	Sedan	Yes	Petrol	6.3	748742.0	1.0	Manual	Front	02-Mar	Left wheel	Black	
15665	45806588	706	1086.0	DODGE	Avenger	2012	Sedan	Yes	Petrol	3.6	667058.0	6.0	Automatic	Front	04-May	Left wheel	Blue	

270 rows × 18 columns

[70]
```
### Drop outliers and reset index

print("Before: {} rows".format(len(dataset)))
dataset = dataset.drop(outliers_to_drop, axis = 0).reset_index(drop = True)
print("After: {} rows".format(len(dataset)))
```

```
Before: 19237 rows
After: 18967 rows
```

[71]
```
### Lets look at the new dataset

dataset
```

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Drive wheels	Doors	Wheel	Color	Airba
0	45654403	13328	1399.0	LEXUS	RX 450	2010	Jeep	Yes	Hybrid	3.5	186005.0	6.0	Automatic	4x4	04-May	Left wheel	Silver	
1	44731507	16621	1018.0	CHEVROLET	Equinox	2011	Jeep	No	Petrol	3.0	192000.0	6.0	Tiptronic	4x4	04-May	Left wheel	Black	
2	45774419	8467	NaN	HONDA	FIT	2006	Hatchback	No	Petrol	1.3	200000.0	4.0	Variator	Front	04-May	Right-hand drive	Black	
3	45769185	3607	862.0	FORD	Escape	2011	Jeep	Yes	Hybrid	2.5	168966.0	4.0	Automatic	4x4	04-May	Left wheel	White	
4	45809263	11726	446.0	HONDA	FIT	2014	Hatchback	Yes	Petrol	1.3	91901.0	4.0	Automatic	Front	04-May	Left wheel	Silver	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
18962	45798355	8467	NaN	MERCEDES-BENZ	CLK 200	1999	Coupe	Yes	CNG	2.0	300000.0	4.0	Manual	Rear	02-Mar	Left wheel	Silver	
18963	45778856	15681	831.0	HYUNDAI	Sonata	2011	Sedan	Yes	Petrol	2.4	161600.0	4.0	Tiptronic	Front	04-May	Left wheel	Red	
18964	45804997	26108	836.0	HYUNDAI	Tucson	2010	Jeep	Yes	Diesel	2.0	116365.0	4.0	Automatic	Front	04-May	Left wheel	Grey	
18965	45793526	5331	1288.0	CHEVROLET	Captiva	2007	Jeep	Yes	Diesel	2.0	51258.0	4.0	Automatic	Front	04-May	Left wheel	Black	
18966	45813273	470	753.0	HYUNDAI	Sonata	2012	Sedan	Yes	Hybrid	2.4	186923.0	4.0	Automatic	Front	04-	Left	White	

[84]
```
### Seperating the categories into class 1 and 2

class_1 = []
class_2 = []

for index in range(len(mean_price_category)):
    if mean_price_category.iloc[index, 1] <= 20000:
        class_1.append(mean_price_category.iloc[index, 0])
    else:
        class_2.append(mean_price_category.iloc[index, 0])

print('Categories with less than 20000 mean price: ', class_1)
print('Categories with more than 20000 mean price: ', class_2)
```

```
Categories with less than 20000 mean price:  ['Hatchback', 'Limousine', 'Microbus', 'Sedan']
Categories with more than 20000 mean price:  ['Cabriolet', 'Coupe', 'Goods wagon', 'Jeep', 'Minivan', 'Pickup', 'Universal']
```

[85]
```
### Modifying the Category column in the dataset

category_data = dataset['Category']
new_category_data = []

for value in category_data:
    if value in class_1:
        new_category_data.append(1)
    else:
        new_category_data.append(2)

dataset['Category'] = new_category_data
```

```
### Looking at the modified dataset

dataset
```

	Price	Levy	Prod. year	Category	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Doors	Wheel	Airbags
0	13328	1399.0	2010	2	Hybrid	3.5	186005.0	6.0	Automatic	04-May	Left wheel	12
1	16621	1018.0	2011	2	Petrol	3.0	192000.0	6.0	Tiptronic	04-May	Left wheel	8
2	8467	781.0	2006	1	Petrol	1.3	200000.0	4.0	Variator	04-May	Right-hand drive	2
3	3607	862.0	2011	2	Hybrid	2.5	168966.0	4.0	Automatic	04-May	Left wheel	0
4	11726	446.0	2014	1	Petrol	1.3	91901.0	4.0	Automatic	04-May	Left wheel	4
...	...	...	...	...	...	...	...	...	...	...	...	...
18962	8467	781.0	1999	2	CNG	2.0	300000.0	4.0	Manual	02-Mar	Left wheel	5
18963	15681	831.0	2011	1	Petrol	2.4	161600.0	4.0	Tiptronic	04-May	Left wheel	8
18964	26108	836.0	2010	2	Diesel	2.0	116365.0	4.0	Automatic	04-May	Left wheel	4
18965	5331	1288.0	2007	2	Diesel	2.0	51258.0	4.0	Automatic	04-May	Left wheel	4
18966	470	753.0	2012	1	Hybrid	2.4	186923.0	4.0	Automatic	04-May	Left wheel	12

18967 rows × 12 columns

```
[87] ### Creating the new fuel type data

fuel_type_data = dataset['Fuel type']
new_fuel_type_data = []

for value in fuel_type_data:
    if value in {'Hybrid', 'Hydrogen', 'Plug-in Hybrid'}:
        new_fuel_type_data.append('Other')
    else:
        new_fuel_type_data.append(value)

set(new_fuel_type_data)
```

```
{'CNG', 'Diesel', 'LPG', 'Other', 'Petrol'}
```

```
[88] ### Modifying the Fuel Type column

dataset['Fuel type'] = new_fuel_type_data
```

```
### Looking at the modified dataset

dataset
```

	Price	Levy	Prod. year	Category	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Doors	Wheel	Airbags
0	13328	1399.0	2010	2	Other	3.5	186005.0	6.0	Automatic	04-May	Left wheel	12
1	16621	1018.0	2011	2	Petrol	3.0	192000.0	6.0	Tiptronic	04-May	Left wheel	8
2	8467	781.0	2006	1	Petrol	1.3	200000.0	4.0	Variator	04-May	Right-hand drive	2
3	3607	862.0	2011	2	Other	2.5	168966.0	4.0	Automatic	04-May	Left wheel	0
4	11726	446.0	2014	1	Petrol	1.3	91901.0	4.0	Automatic	04-May	Left wheel	4
...	...	...	...	...	...	...	...	...	...	...	...	...
18962	8467	781.0	1999	2	CNG	2.0	300000.0	4.0	Manual	02-Mar	Left wheel	5
18963	15681	831.0	2011	1	Petrol	2.4	161600.0	4.0	Tiptronic	04-May	Left wheel	8
18964	26108	836.0	2010	2	Diesel	2.0	116365.0	4.0	Automatic	04-May	Left wheel	4
18965	5331	1288.0	2007	2	Diesel	2.0	51258.0	4.0	Automatic	04-May	Left wheel	4
18966	470	753.0	2012	1	Other	2.4	186923.0	4.0	Automatic	04-May	Left wheel	12

18967 rows × 12 columns

```python
[90] ### Seperating the categories into class 1 and 2

gear_box_data = dataset['Gear box type']
new_gear_box_data = []

for value in gear_box_data:
    if value in {'Automatic', 'Variator'}:
        new_gear_box_data.append(1)
    else:
        new_gear_box_data.append(2)

set(new_gear_box_data)
```

```
{1, 2}
```

```python
[91] ### Modifying the Gear box type column

dataset['Gear box type'] = new_gear_box_data
```

```python
[92] ### Looking at the modified dataset

dataset
```

	Price	Levy	Prod. year	Category	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Doors	Wheel	Airbags
0	13328	1399.0	2010	2	Other	3.5	186005.0	6.0	1	04-May	Left wheel	12
1	16621	1018.0	2011	2	Petrol	3.0	192000.0	6.0	2	04-May	Left wheel	8
2	8467	781.0	2006	1	Petrol	1.3	200000.0	4.0	1	04-May	Right-hand drive	2
3	3607	862.0	2011	2	Other	2.5	168966.0	4.0	1	04-May	Left wheel	0
4	11726	446.0	2014	1	Petrol	1.3	91901.0	4.0	1	04-May	Left wheel	4
...	...	...	...	...	...	...	...	...	...	...	...	...
18962	8467	781.0	1999	2	CNG	2.0	300000.0	4.0	2	02-Mar	Left wheel	5
18963	15681	831.0	2011	1	Petrol	2.4	161600.0	4.0	2	04-May	Left wheel	8
18964	26108	836.0	2010	2	Diesel	2.0	116365.0	4.0	1	04-May	Left wheel	4
18965	5331	1288.0	2007	2	Diesel	2.0	51258.0	4.0	1	04-May	Left wheel	4
18966	470	753.0	2012	1	Other	2.4	186923.0	4.0	1	04-May	Left wheel	12

18967 rows × 12 columns

[94]
```
### Creating the new Doors data

doors_data = dataset['Doors']
new_doors_data = []

for value in doors_data:
    if value == '04-May':
        new_doors_data.append('4-5')
    elif value == '02-Mar':
        new_doors_data.append('2-3')
    else:
        new_doors_data.append(value)

set(new_doors_data)
```

```
{'2-3', '4-5', '>5'}
```

[95]
```
### Modifying the Doors column

dataset['Doors'] = new_doors_data
```

```
### Looking at the modified dataset

dataset
```

	Price	Levy	Prod. year	Category	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Doors	Wheel	Airbags
0	13328	1399.0	2010	2	Other	3.5	186005.0	6.0	1	4-5	Left wheel	12
1	16621	1018.0	2011	2	Petrol	3.0	192000.0	6.0	2	4-5	Left wheel	8
2	8467	781.0	2006	1	Petrol	1.3	200000.0	4.0	1	4-5	Right-hand drive	2
3	3607	862.0	2011	2	Other	2.5	168966.0	4.0	1	4-5	Left wheel	0
4	11726	446.0	2014	1	Petrol	1.3	91901.0	4.0	1	4-5	Left wheel	4
...	...	...	...	...	...	...	...	...	...	...	...	...
18962	8467	781.0	1999	2	CNG	2.0	300000.0	4.0	2	2-3	Left wheel	5
18963	15681	831.0	2011	1	Petrol	2.4	161600.0	4.0	2	4-5	Left wheel	8
18964	26108	836.0	2010	2	Diesel	2.0	116365.0	4.0	1	4-5	Left wheel	4
18965	5331	1288.0	2007	2	Diesel	2.0	51258.0	4.0	1	4-5	Left wheel	4
18966	470	753.0	2012	1	Other	2.4	186923.0	4.0	1	4-5	Left wheel	12

18967 rows × 12 columns

+ Code +

```python
[97] ### Creating the Age data

     year_data = dataset['Prod. year']
     age_data = []

     for value in year_data:
         age_data.append(2022 - value)

     len(set(age_data))
```

```
51
```

```python
[98] ### Creating the Age column

     dataset['Age'] = age_data
```

```python
[99] ### Removing the Prod. year column

     dataset.drop(['Prod. year'], axis = 1, inplace = True)
```

```python
### Looking at the modified dataset

dataset
```

	Price	Levy	Category	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Doors	Wheel	Airbags	Age
0	13328	1399.0	2	Other	3.5	186005.0	6.0	1	4-5	Left wheel	12	12
1	16621	1018.0	2	Petrol	3.0	192000.0	6.0	2	4-5	Left wheel	8	11
2	8467	781.0	1	Petrol	1.3	200000.0	4.0	1	4-5	Right-hand drive	2	16
3	3607	862.0	2	Other	2.5	168966.0	4.0	1	4-5	Left wheel	0	11
4	11726	446.0	1	Petrol	1.3	91901.0	4.0	1	4-5	Left wheel	4	8
...	...	...	...	...	...	...	...	...	...	...	...	...
18962	8467	781.0	2	CNG	2.0	300000.0	4.0	2	2-3	Left wheel	5	23
18963	15681	831.0	1	Petrol	2.4	161600.0	4.0	2	4-5	Left wheel	8	11
18964	26108	836.0	2	Diesel	2.0	116365.0	4.0	1	4-5	Left wheel	4	12
18965	5331	1288.0	2	Diesel	2.0	51258.0	4.0	1	4-5	Left wheel	4	15
18966	470	753.0	1	Other	2.4	186923.0	4.0	1	4-5	Left wheel	12	10

18967 rows × 12 columns

```
### One Hot Encoding the columns - Category, Fuel type, Gear box type, Doors, Wheel of the dataset

encoded_dataset = pd.get_dummies(data = dataset, columns = ['Category', 'Fuel type', 'Gear box type', 'Doors', 'Wheel'])
encoded_dataset
```

	Price	Levy	Engine volume	Mileage	Cylinders	Airbags	Age	Category_1	Category_2	Fuel type_CNG	...	Fuel type_LPG	Fuel type_Other	Fuel type_Petrol	Gear box type_1	Ge b type
0	49.474439	0.629677	0.658421	0.179513	0.333333	0.7500	0.479728	0	1	0	...	0	1	0	1	
1	52.930471	0.566695	0.628821	0.180635	0.333333	0.5000	0.456381	0	1	0	...	0	0	1	0	
2	43.023184	0.512535	0.469937	0.182087	0.200000	0.1250	0.556958	1	0	0	...	0	0	1	1	
3	32.941926	0.532881	0.593937	0.176152	0.200000	0.0000	0.456381	0	1	0	...	0	1	0	1	
4	47.567865	0.392874	0.469937	0.156094	0.200000	0.2500	0.370976	1	0	0	...	0	0	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
18962	43.023184	0.512535	0.551429	0.197086	0.200000	0.3125	0.654463	0	1	1	...	0	0	0	0	
18963	51.997927	0.525355	0.586145	0.174611	0.200000	0.5000	0.456381	1	0	0	...	0	0	1	0	
18964	60.716837	0.526589	0.551429	0.163619	0.200000	0.2500	0.479728	0	1	0	...	0	0	0	1	
18965	37.253177	0.613505	0.551429	0.138741	0.200000	0.2500	0.539627	0	1	0	...	0	0	0	1	
18966	16.837872	0.504954	0.586145	0.179686	0.200000	0.7500	0.430812	1	0	0	...	0	1	0	1	

18967 rows × 21 columns

```
[114] ### Create the column - Target using Price

target_data = encoded_dataset['Price']
encoded_dataset['Target'] = target_data

### Dropping the column - Price

encoded_dataset.drop(['Price'], axis = 1, inplace = True)
encoded_dataset
```

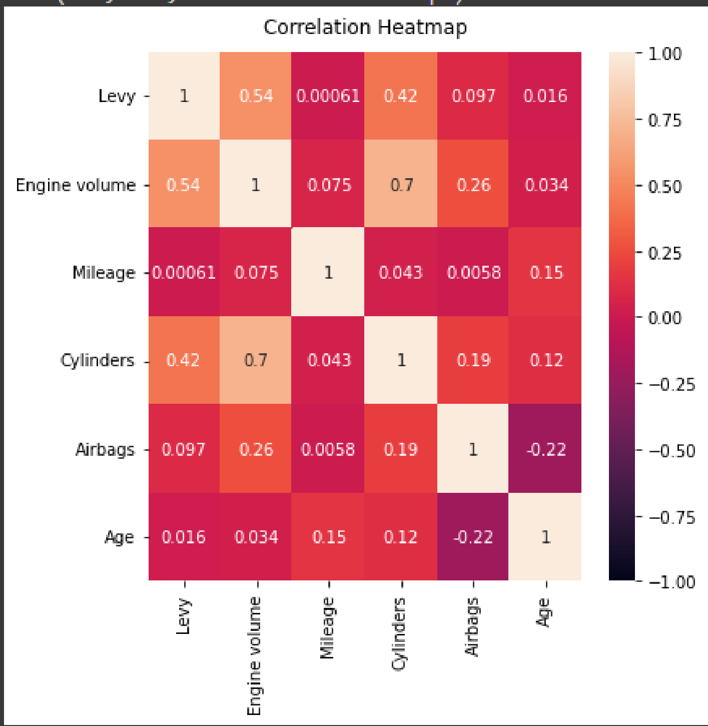
	Levy	Engine volume	Mileage	Cylinders	Airbags	Age	Category_1	Category_2	Fuel type_CNG	Fuel type_Diesel	...	Fuel type_Other	Fuel type_Petrol	Gear box type_1	Gear box type_2	Doo
0	0.629677	0.658421	0.179513	0.333333	0.7500	0.479728	0	1	0	0	...	1	0	1	0	
1	0.566695	0.628821	0.180635	0.333333	0.5000	0.456381	0	1	0	0	...	0	1	0	1	
2	0.512535	0.469937	0.182087	0.200000	0.1250	0.556958	1	0	0	0	...	0	1	1	0	
3	0.532881	0.593937	0.176152	0.200000	0.0000	0.456381	0	1	0	0	...	1	0	1	0	
4	0.392874	0.469937	0.156094	0.200000	0.2500	0.370976	1	0	0	0	...	0	1	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
18962	0.512535	0.551429	0.197086	0.200000	0.3125	0.654463	0	1	1	0	...	0	0	0	1	
18963	0.525355	0.586145	0.174611	0.200000	0.5000	0.456381	1	0	0	0	...	0	1	0	1	
18964	0.526589	0.551429	0.163619	0.200000	0.2500	0.479728	0	1	0	1	...	0	0	1	0	
18965	0.613505	0.551429	0.138741	0.200000	0.2500	0.539627	0	1	0	1	...	0	0	1	0	
18966	0.504954	0.586145	0.179686	0.200000	0.7500	0.430812	1	0	0	0	...	1	0	1	0	

18967 rows × 21 columns

```
### Plotting the correlation between various columns of the filter_dataset

plt.figure(figsize = (6, 6))
heatmap = sns.heatmap(filter_dataset.corr(), vmin = -1, vmax = 1, annot = True)
heatmap.set_title('Correlation Heatmap', fontdict = {'fontsize' : 12}, pad = 12)
```

```
Text(0.5, 1.0, 'Correlation Heatmap')
```



```
[120] ### Splitting the dataset to the matrices X and Y

      X = encoded_dataset.iloc[:, : -1].values
      Y = encoded_dataset.iloc[:, -1].values
```

```
[121] ### Looking at the new training data - X

      X

      array([[0.62967695, 0.65842071, 0.17951255, ..., 0.        , 1.        ,
              0.        ],
             [0.56669518, 0.62882066, 0.18063451, ..., 0.        , 1.        ,
              0.        ],
             [0.51253544, 0.46993749, 0.18208737, ..., 0.        , 0.        ,
              1.        ],
             ...,
             [0.52658936, 0.55142852, 0.16361879, ..., 0.        , 1.        ,
              0.        ],
             [0.61350496, 0.55142852, 0.1387409 , ..., 0.        , 1.        ,
              0.        ],
             [0.50495351, 0.58614549, 0.17968628, ..., 0.        , 1.        ,
              0.        ]])
```

```
[122] ### Looking at the new test data - Y

      Y

      array([49.47443883, 52.93047067, 43.0231844 , ..., 60.71683673,
             37.25317749, 16.83787204])
```

```
[123] ### Dividing the dataset into train and test in the ratio of 80 : 20

      X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 27, shuffle = True)
```

+ Code + Text

```
X_train

array([[0.51253544, 0.66911524, 0.16090855, ..., 0.        , 1.        ,
        0.        ],
       [0.51253544, 0.55142852, 0.16212153, ..., 0.        , 0.        ,
        1.        ],
       [0.67276052, 0.6471106 , 0.17875086, ..., 0.        , 1.        ,
        0.        ],
       ...,
       [0.54507973, 0.62882066, 0.18728915, ..., 0.        , 1.        ,
        0.        ],
       [0.57838427, 0.55142852, 0.10923963, ..., 0.        , 1.        ,
        0.        ],
       [0.51253544, 0.53142795, 0.15057149, ..., 0.        , 1.        ,
        0.        ]])
```

```
[125] X_test

      array([[0.54170822, 0.58614549, 0.15335443, ..., 0.        , 1.        ,
              0.        ],
             [0.59987646, 0.58614549, 0.        , ..., 0.        , 1.        ,
              0.        ],
             [0.44311923, 0.48390293, 0.15367126, ..., 0.        , 1.        ,
              0.        ],
             ...,
             [0.50824049, 0.55142852, 0.16160244, ..., 0.        , 1.        ,
              0.        ],
             [0.51253544, 0.64120146, 0.18124524, ..., 0.        , 1.        ,
              0.        ],
             [0.61212663, 0.4969244 , 0.05441558, ..., 0.        , 1.        ,
              0.        ]])
```

```
[126] Y_train

      array([55.24763947, 36.04751909, 16.83787204, ..., 50.00365691,
             29.6349931 , 55.79245286])
```

```
[127] Y_test

      array([57.8565661 , 53.97325547, 62.08268892, ..., 20.04504146,
             45.774181  , 11.16801217])
```

5.2.1 Applying Multi Linear Regression

```
[130] ### Training the Multi Linear Regression model on the Training set

      linear_regressor = LinearRegression()
      linear_regressor.fit(X_train, Y_train)
```

```
LinearRegression()
```

```
[131] ### Predicting the Test set results

      Y_pred = linear_regressor.predict(X_test)
```

```
[132] ### Calculating RMSE and Adjusted R-squared for the model

      mse = round(mean_squared_error(Y_test, Y_pred), 3)
      rmse = round(sqrt(mse), 3)

      r2_value = round(r2_score(Y_test, Y_pred), 3)

      model_rmse['Multi Linear Regression'] = rmse
      model_r2['Multi Linear Regression'] = r2_value

      print('Root Mean Squared Error of the model is : {}'.format(rmse))
      print('R-squared value of the model is : {}'.format(r2_value))
```

```
Root Mean Squared Error of the model is : 15.269
R-squared value of the model is : 0.216
```

5.2.2 Applying Lasso Regression

```
[133] ### Training the Lasso Regression model on the Training set

      lasso = Lasso()
      parameters = {'alpha': [1e-15, 1e-10, 1e-8, 1e-3, 1e-2, 1, 5, 10, 20, 30, 35, 40, 45, 50, 55, 100]}
      lasso_regressor = GridSearchCV(lasso, parameters, scoring = 'neg_mean_squared_error', cv = 5)
      lasso_regressor.fit(X_train, Y_train)
```

```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.001, 0.01, 1, 5, 10,
                                   20, 30, 35, 40, 45, 50, 55, 100]},
             scoring='neg_mean_squared_error')
```

```
[134] # Finding out negative mean squared error in Lasso Regression

      print(lasso_regressor.best_params_)
      print(lasso_regressor.best_score_)
```

```
{'alpha': 0.001}
-250.45196068544993
```

```
[135] ### Predicting the Test set results

      Y_pred = lasso_regressor.predict(X_test)
```

[136]
```
### Calculating RMSE and Adjusted R-squared for the model

mse = round(mean_squared_error(Y_test, Y_pred), 3)
rmse = round(sqrt(mse), 3)

r2_value = round(r2_score(Y_test, Y_pred), 3)

model_rmse['Lasso Regression'] = rmse
model_r2['Lasso Regression'] = r2_value

print('Root Mean Squared Error of the model is : {}'.format(rmse))
print('R-squared value of the model is : {}'.format(r2_value))
```

```
Root Mean Squared Error of the model is : 15.269
R-squared value of the model is : 0.216
```

5.2.4 Applying Support Vector Regression

[137]
```
### Training the Support Vector Regression model on the Training set

support_vector_regressor = SVR(kernel = 'rbf')
support_vector_regressor.fit(X_train, Y_train)
```

```
SVR()
```

[138]
```
### Predicting the Test set results

Y_pred = support_vector_regressor.predict(X_test)
```

[139]
```
### Calculating RMSE and Adjusted R-squared for the model

mse = round(mean_squared_error(Y_test, Y_pred), 3)
rmse = round(sqrt(mse), 3)

r2_value = round(r2_score(Y_test, Y_pred), 3)

model_rmse['Support Vector Regression'] = rmse
model_r2['Support Vector Regression'] = r2_value

print('Root Mean Squared Error of the model is : {}'.format(rmse))
print('R-squared value of the model is : {}'.format(r2_value))
```

```
Root Mean Squared Error of the model is : 14.105
R-squared value of the model is : 0.331
```

5.2.6 Applying Random Forest Regression (10 trees)

[140]
```
### Training the Random Forest Regression model on the Training set

random_forest_regressor = RandomForestRegressor(n_estimators = 10, random_state = 27)
random_forest_regressor.fit(X_train, Y_train)
```

```
RandomForestRegressor(n_estimators=10, random_state=27)
```

[141]
```
### Predicting the Test set results

Y_pred = random_forest_regressor.predict(X_test)
```

[142]
```
### Calculating RMSE and Adjusted R-squared for the model

mse = round(mean_squared_error(Y_test, Y_pred), 3)
rmse = round(sqrt(mse), 3)

r2_value = round(r2_score(Y_test, Y_pred), 3)

model_rmse['Random Forest Regression (10 trees)'] = rmse
model_r2['Random Forest Regression (10 trees)'] = r2_value

print('Root Mean Squared Error of the model is : {}'.format(rmse))
print('R-squared value of the model is : {}'.format(r2_value))
```

```
Root Mean Squared Error of the model is : 10.62
R-squared value of the model is : 0.621
```

5.2.7 Applying Random Forest Regression (25 trees)

[143]
```
### Training the Random Forest Regression model on the Training set

random_forest_regressor = RandomForestRegressor(n_estimators = 25, random_state = 27)
random_forest_regressor.fit(X_train, Y_train)
```

```
RandomForestRegressor(n_estimators=25, random_state=27)
```

[144]
```
### Predicting the Test set results

Y_pred = random_forest_regressor.predict(X_test)
```

[145]
```
### Calculating RMSE and Adjusted R-squared for the model

mse = round(mean_squared_error(Y_test, Y_pred), 3)
rmse = round(sqrt(mse), 3)

r2_value = round(r2_score(Y_test, Y_pred), 3)

model_rmse['Random Forest Regression (25 trees)'] = rmse
model_r2['Random Forest Regression (25 trees)'] = r2_value

print('Root Mean Squared Error of the model is : {}'.format(rmse))
print('R-squared value of the model is : {}'.format(r2_value))
```

```
Root Mean Squared Error of the model is : 10.324
R-squared value of the model is : 0.642
```

5.2.8 Applying Random Forest Regression (50 trees)

[146]
```
### Training the Random Forest Regression model on the Training set

random_forest_regressor = RandomForestRegressor(n_estimators = 50, random_state = 27)
random_forest_regressor.fit(X_train, Y_train)
```

```
RandomForestRegressor(n_estimators=50, random_state=27)
```

+ Code + Text

[147]
```
### Predicting the Test set results

Y_pred = random_forest_regressor.predict(X_test)
```

[148]
```
### Calculating RMSE and Adjusted R-squared for the model

mse = round(mean_squared_error(Y_test, Y_pred), 3)
rmse = round(sqrt(mse), 3)

r2_value = round(r2_score(Y_test, Y_pred), 3)

model_rmse['Random Forest Regression (50 trees)'] = rmse
model_r2['Random Forest Regression (50 trees)'] = r2_value

print('Root Mean Squared Error of the model is : {}'.format(rmse))
print('R-squared value of the model is : {}'.format(r2_value))
```

```
Root Mean Squared Error of the model is : 10.149
R-squared value of the model is : 0.654
```

5.2.9 Applying Random Forest Regression (100 trees)

```
[149] ### Training the Random Forest Regression model on the Training set

      random_forest_regressor = RandomForestRegressor(n_estimators = 100, random_state = 27)
      random_forest_regressor.fit(X_train, Y_train)
```

```
RandomForestRegressor(random_state=27)
```

```
[150] ### Predicting the Test set results

      Y_pred = random_forest_regressor.predict(X_test)
```

```
[151] ### Calculating RMSE and Adjusted R-squared for the model

      mse = round(mean_squared_error(Y_test, Y_pred), 3)
      rmse = round(sqrt(mse), 3)

      r2_value = round(r2_score(Y_test, Y_pred), 3)

      model_rmse['Random Forest Regression (100 trees)'] = rmse
      model_r2['Random Forest Regression (100 trees)'] = r2_value

      print('Root Mean Squared Error of the model is : {}'.format(rmse))
      print('R-squared value of the model is : {}'.format(r2_value))
```

```
Root Mean Squared Error of the model is : 10.091
R-squared value of the model is : 0.658
```

5.2.10 Applying Random Forest Regression (1000 trees)

[152]
```
### Training the Random Forest Regression model on the Training set

random_forest_regressor = RandomForestRegressor(n_estimators = 1000, random_state = 27)
random_forest_regressor.fit(X_train, Y_train)
```

```
RandomForestRegressor(n_estimators=1000, random_state=27)
```

[153]
```
### Predicting the Test set results

Y_pred = random_forest_regressor.predict(X_test)
```

[154]
```
### Calculating RMSE and Adjusted R-squared for the model

mse = round(mean_squared_error(Y_test, Y_pred), 3)
rmse = round(sqrt(mse), 3)

r2_value = round(r2_score(Y_test, Y_pred), 3)

model_rmse['Random Forest Regression (1000 trees)'] = rmse
model_r2['Random Forest Regression (1000 trees)'] = r2_value

print('Root Mean Squared Error of the model is : {}'.format(rmse))
print('R-squared value of the model is : {}'.format(r2_value))
```

```
Root Mean Squared Error of the model is : 10.04
R-squared value of the model is : 0.661
```

5.3.1 RMSE, R-squared of the models

Now we will tabulate all the models along with their rmse, r-squared. This data is stored in the model_performance dictionary. We will use the tabulate package for tabulating the results.

[155]
```
### Looking at the model rmse dictionary

model_rmse
```

```
OrderedDict([('Multi Linear Regression', 15.269),
             ('Lasso Regression', 15.269),
             ('Support Vector Regression', 14.105),
             ('Random Forest Regression (10 trees)', 10.62),
             ('Random Forest Regression (25 trees)', 10.324),
             ('Random Forest Regression (50 trees)', 10.149),
             ('Random Forest Regression (100 trees)', 10.091),
             ('Random Forest Regression (1000 trees)', 10.04)])
```

[156]
```
### Looking at the model r-squared dictionary

model_r2
```

```
OrderedDict([('Multi Linear Regression', 0.216),
             ('Lasso Regression', 0.216),
             ('Support Vector Regression', 0.331),
             ('Random Forest Regression (10 trees)', 0.621),
             ('Random Forest Regression (25 trees)', 0.642),
             ('Random Forest Regression (50 trees)', 0.654),
             ('Random Forest Regression (100 trees)', 0.658),
             ('Random Forest Regression (1000 trees)', 0.661)])
```

```python
### Tabulating the results

table = []
table.append(['S.No.', 'Classification Model', 'Root Mean Squared Error', 'R-squared'])
count = 1

for model in model_rmse:
    row = [count, model, model_rmse[model], model_r2[model]]
    table.append(row)
    count += 1

print(tabulate(table, headers = 'firstrow', tablefmt = 'fancy_grid'))
```

S.No.	Classification Model	Root Mean Squared Error	R-squared
1	Multi Linear Regression	15.269	0.216
2	Lasso Regression	15.269	0.216
3	Support Vector Regression	14.105	0.331
4	Random Forest Regression (10 trees)	10.62	0.621
5	Random Forest Regression (25 trees)	10.324	0.642
6	Random Forest Regression (50 trees)	10.149	0.654
7	Random Forest Regression (100 trees)	10.091	0.658
8	Random Forest Regression (1000 trees)	10.04	0.661

From the above table, we can see that the model Random Forest Regression (1000 trees) has the least Root Mean Squared Error of 10.043 and the highest R-squared value of 0.661.

Random forest model prediction

```python
[158] X_test[0]
```

```
array([0.54170822, 0.58614549, 0.15335443, 0.2       , 0.75      ,
       0.33518485, 1.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 1.        , 1.        , 0.        ,
       0.        , 1.        , 0.        , 1.        , 0.        ])
```

```python
[159] random_forest_regressor.predict(X_test[0].reshape(1,-1))
```

```
array([32.88202831])
```

6. Conclusion

Hence, for this problem, we will use Random Forest regressor to predict the Sales Price.