РТ5-61Б, Забурунов Л. В.

Технологии Машинного Обучения

Лабораторная Работа №2

"Обработка пропусков в данных. Кодирование категориальных признаков. Масштабирование данных"

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns


lab2_data = pd.read_csv("ML_Datasets/Lab2/train-data.csv")
```

1. Исследование структуры данных

In [2]:

```
lab2_data.shape
```

Out[2]:

```
(6019, 14)
```

In [3]:

```
lab2_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6019 entries, 0 to 6018
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         6019 non-null   int64  
 1   Name               6019 non-null   object 
 2   Location           6019 non-null   object 
 3   Year               6019 non-null   int64  
 4   Kilometers_Driven  6019 non-null   int64  
 5   Fuel_Type          6019 non-null   object 
 6   Transmission       6019 non-null   object 
 7   Owner_Type         6019 non-null   object 
 8   Mileage            6017 non-null   object 
 9   Engine             5983 non-null   object 
 10  Power              5983 non-null   object 
 11  Seats              5977 non-null   float64
 12  New_Price          824 non-null    object 
 13  Price              6019 non-null   float64
dtypes: float64(2), int64(3), object(9)
memory usage: 658.5+ KB
```

In [4]:

```
## Исследуем структуру набора данных
def PrintDatasetInfo(dataframe):
    index = 0
    for column in dataframe.columns:
        column_name = column
        column_type = str(dataframe[column].dtypes)
        column_values = "(continuous)" if column_type == "float64" else dataframe[column].unique()
        column_nulls = dataframe[dataframe[column].isnull()].shape[0]
        print("\nСтолбец {0} (тип {1}) имеет {2} пропусков (индекс {3})".format(column_name, column_type, colu
        index = index + 1

PrintDatasetInfo(lab2_data)
```

```
Столбец Unnamed: 0 (тип int64) имеет 0 пропусков (индекс 0)

Столбец Name (тип object) имеет 0 пропусков (индекс 1)

Столбец Location (тип object) имеет 0 пропусков (индекс 2)

Столбец Year (тип int64) имеет 0 пропусков (индекс 3)

Столбец Kilometers_Driven (тип int64) имеет 0 пропусков (индекс 4)

Столбец Fuel_Type (тип object) имеет 0 пропусков (индекс 5)

Столбец Transmission (тип object) имеет 0 пропусков (индекс 6)

Столбец Owner_Type (тип object) имеет 0 пропусков (индекс 7)

Столбец Mileage (тип object) имеет 2 пропусков (индекс 8)

Столбец Engine (тип object) имеет 36 пропусков (индекс 9)

Столбец Power (тип object) имеет 36 пропусков (индекс 10)

Столбец Seats (тип float64) имеет 42 пропусков (индекс 11)

Столбец New_Price (тип object) имеет 5195 пропусков (индекс 12)

Столбец Price (тип float64) имеет 0 пропусков (индекс 13)
```

Автор набора данных описывает имеющиеся столбцы следующим образом:

1. Unnamed - ID;
2. Name - название авто;
3. Location - место продажи или выставления на продажу;
4. Year - год выпуска авто;
5. Kilometeres_Driven - общий пробег в километрах;
6. Fuel_Type - тип используемого топлива;
7. Transmission - коробка передач (механика/автомат);
8. Owner_Type - какой по счёту владелец;
9. Mileage - удельный расход топлива;
10. Engine - объём двигателя;
11. Power - мощность двигателя;
12. Seats - число сидений;
13. New_Price - цена новой машины идентичной модели;
14. Price - цена на текущий образец

In [5]:

```
lab2_data.head(20)
```

Out[5]:

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Pric
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	Na
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	Na
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lak
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	Na
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	Na
5	5	Hyundai EON LPG Era Plus Option	Hyderabad	2012	75000	LPG	Manual	First	21.1 km/kg	814 CC	55.2 bhp	5.0	Na
6	6	Nissan Micra Diesel XV	Jaipur	2013	86999	Diesel	Manual	First	23.08 kmpl	1461 CC	63.1 bhp	5.0	Na
7	7	Toyota Innova Crysta 2.8 GX AT 8S	Mumbai	2016	36000	Diesel	Automatic	First	11.36 kmpl	2755 CC	171.5 bhp	8.0	21 Lak
8	8	Volkswagen Vento Diesel Comfortline	Pune	2013	64430	Diesel	Manual	First	20.54 kmpl	1598 CC	103.6 bhp	5.0	Na
9	9	Tata Indica Vista Quadrajet LS	Chennai	2012	65932	Diesel	Manual	Second	22.3 kmpl	1248 CC	74 bhp	5.0	Na
10	10	Maruti Ciaz Zeta	Kochi	2018	25692	Petrol	Manual	First	21.56 kmpl	1462 CC	103.25 bhp	5.0	10.65 Lak
11	11	Honda City 1.5 V AT Sunroof	Kolkata	2012	60000	Petrol	Automatic	First	16.8 kmpl	1497 CC	116.3 bhp	5.0	Na
12	12	Maruti Swift VDI BSIV	Jaipur	2015	64424	Diesel	Manual	First	25.2 kmpl	1248 CC	74 bhp	5.0	Na
13	13	Land Rover Range Rover 2.2L Pure	Delhi	2014	72000	Diesel	Automatic	First	12.7 kmpl	2179 CC	187.7 bhp	5.0	Na
14	14	Land Rover Freelander 2 TD4 SE	Pune	2012	85000	Diesel	Automatic	Second	0.0 kmpl	2179 CC	115 bhp	5.0	Na
15	15	Mitsubishi Pajero Sport 4X4	Delhi	2014	110000	Diesel	Manual	First	13.5 kmpl	2477 CC	175.56 bhp	7.0	32.01 Lak
16	16	Honda Amaze S i-Dtech	Kochi	2016	58950	Diesel	Manual	First	25.8 kmpl	1498 CC	98.6 bhp	5.0	Na
17	17	Maruti Swift DDiS VDI	Jaipur	2017	25000	Diesel	Manual	First	28.4 kmpl	1248 CC	74 bhp	5.0	Na
18	18	Renault Duster 85PS Diesel RxL Plus	Kochi	2014	77469	Diesel	Manual	First	20.45 kmpl	1461 CC	83.8 bhp	5.0	Na
19	19	Mercedes-Benz New C-Class C 220 CDI BE Avantgare	Bangalore	2014	78500	Diesel	Automatic	First	14.84 kmpl	2143 CC	167.62 bhp	5.0	Na

Отметим следующие проблемы в структуре датасета:

1. Столбец Owner_Type можно перевести в разряд числовых, поскольку по смыслу это кол-во предыдущих владельцев;
2. В столбце Mileage видим числовой признак с указанием единицы измерения, причём единицы разные;
3. В столбцах Engine и Power добавлены единицы измерения, однако это лишняя информация, поскольку единица используется одна;
4. Колонка Seats имеет значения int, поэтому можно безболезненно превратить её в целочисленную;
5. В колонке New_Price катастрофически много пропусков, поэтому данную колонку можно вовсе исключить.

2. Корректировка структуры набора данных

In [6]:

```
lab2_array = lab2_data.to_numpy()
lab2_array_1 = lab2_array.copy()
```

In [7]:

```
for i in range(lab2_array.shape[1]):
    print(lab2_array[:, i])
```

```
[0 1 2 ... 6016 6017 6018]
['Maruti Wagon R LXI CNG' 'Hyundai Creta 1.6 CRDi SX Option'
 'Honda Jazz V' ... 'Mahindra Xylo D4 BSIV' 'Maruti Wagon R VXI'
 'Chevrolet Beat Diesel']
['Mumbai' 'Pune' 'Chennai' ... 'Jaipur' 'Kolkata' 'Hyderabad']
[2010 2015 2011 ... 2012 2013 2011]
[72000 41000 46000 ... 55000 46000 47000]
['CNG' 'Diesel' 'Petrol' ... 'Diesel' 'Petrol' 'Diesel']
['Manual' 'Manual' 'Manual' ... 'Manual' 'Manual' 'Manual']
['First' 'First' 'First' ... 'Second' 'First' 'First']
['26.6 km/kg' '19.67 kmpl' '18.2 kmpl' ... '14.0 kmpl' '18.9 kmpl'
 '25.44 kmpl']
['998 CC' '1582 CC' '1199 CC' ... '2498 CC' '998 CC' '936 CC']
['58.16 bhp' '126.2 bhp' '88.7 bhp' ... '112 bhp' '67.1 bhp' '57.6 bhp']
[5.0 5.0 5.0 ... 8.0 5.0 5.0]
[nan nan '8.61 Lakh' ... nan nan nan]
[1.75 12.5 4.5 ... 2.9 2.65 2.5]
```

Исключение ненужных столбцов

С точки зрения алгоритмов МО не будут являться полезными колонки Name, ID и New_Price

In [8]:

```
lab2_array = np.delete(lab2_array, [12, 1, 0], 1)
```

In [9]:

```
for i in range(lab2_array.shape[1]):
    print(str(i) + ":", lab2_array[:, i])
```

```
0: ['Mumbai' 'Pune' 'Chennai' ... 'Jaipur' 'Kolkata' 'Hyderabad']
1: [2010 2015 2011 ... 2012 2013 2011]
2: [72000 41000 46000 ... 55000 46000 47000]
3: ['CNG' 'Diesel' 'Petrol' ... 'Diesel' 'Petrol' 'Diesel']
4: ['Manual' 'Manual' 'Manual' ... 'Manual' 'Manual' 'Manual']
5: ['First' 'First' 'First' ... 'Second' 'First' 'First']
6: ['26.6 km/kg' '19.67 kmpl' '18.2 kmpl' ... '14.0 kmpl' '18.9 kmpl'
 '25.44 kmpl']
7: ['998 CC' '1582 CC' '1199 CC' ... '2498 CC' '998 CC' '936 CC']
8: ['58.16 bhp' '126.2 bhp' '88.7 bhp' ... '112 bhp' '67.1 bhp' '57.6 bhp']
9: [5.0 5.0 5.0 ... 8.0 5.0 5.0]
10: [1.75 12.5 4.5 ... 2.9 2.65 2.5]
```

Owner_Type

Здесь перекодируем текстовые значения в числовые (UPD изобрёл велосипед с "label encoding")

In [10]:

```
lab2_data["Owner_Type"].unique()
```

Out[10]:

```
array(['First', 'Second', 'Fourth & Above', 'Third'], dtype=object)
```

In [11]:

```
for rowNum in range(lab2_array.shape[0]):
    if (lab2_array[rowNum][5] == "First"):
        lab2_array[rowNum][5] = 1
    elif (lab2_array[rowNum][5] == "Second"):
        lab2_array[rowNum][5] = 2
    elif (lab2_array[rowNum][5] == "Third"):
```

```
        lab2_array[rowNum][5] = 3
    else:
        lab2_array[rowNum][5] = 4
```

Mileage.

Здесь видим расхождения в единицах измерения.

In [12]:

```
lab2_data["Mileage"].unique()
```

Out[12]:

```
array(['26.6 km/kg', '19.67 kmpl', '18.2 kmpl', '20.77 kmpl', '15.2 kmpl',
       '21.1 km/kg', '23.08 kmpl', '11.36 kmpl', '20.54 kmpl',
       '22.3 kmpl', '21.56 kmpl', '16.8 kmpl', '25.2 kmpl', '12.7 kmpl',
       '0.0 kmpl', '13.5 kmpl', '25.8 kmpl', '28.4 kmpl', '20.45 kmpl',
       '14.84 kmpl', '22.69 kmpl', '23.65 kmpl', '13.53 kmpl',
       '18.5 kmpl', '14.4 kmpl', '20.92 kmpl', '17.5 kmpl', '12.8 kmpl',
       '19.01 kmpl', '14.53 kmpl', '11.18 kmpl', '12.4 kmpl',
       '16.09 kmpl', '14.0 kmpl', '24.3 kmpl', '18.15 kmpl', '11.74 kmpl',
       '22.07 kmpl', '19.7 kmpl', '25.4 kmpl', '25.32 kmpl', '14.62 kmpl',
       '14.28 kmpl', '14.9 kmpl', '11.25 kmpl', '24.4 kmpl', '16.55 kmpl',
       '17.11 kmpl', '22.9 kmpl', '17.8 kmpl', '18.9 kmpl', '15.04 kmpl',
       '25.17 kmpl', '20.36 kmpl', '13.29 kmpl', '13.68 kmpl',
       '20.0 kmpl', '15.8 kmpl', '25.0 kmpl', '16.4 kmpl', '24.52 kmpl',
       '22.1 kmpl', '8.5 kmpl', '15.1 kmpl', '16.95 kmpl', '19.64 kmpl',
       '16.5 kmpl', '18.53 kmpl', '17.57 kmpl', '18.0 kmpl', '23.2 kmpl',
       '16.73 kmpl', '17.0 kmpl', '13.0 kmpl', '17.68 kmpl', '22.7 kmpl',
       '16.2 kmpl', '15.26 kmpl', '23.0 kmpl', '19.83 kmpl', '14.94 kmpl',
       '17.71 kmpl', '14.74 kmpl', '16.0 kmpl', '22.32 kmpl',
       '12.99 kmpl', '23.3 kmpl', '19.15 kmpl', '10.8 kmpl', '15.0 kmpl',
       '22.0 kmpl', '21.9 kmpl', '12.05 kmpl', '11.7 kmpl', '21.21 kmpl',
       '20.73 kmpl', '21.1 kmpl', '24.07 kmpl', '19.0 kmpl', '20.58 kmpl',
       '19.27 kmpl', '11.5 kmpl', '18.6 kmpl', '21.14 kmpl', '11.05 kmpl',
       '21.76 kmpl', '7.81 kmpl', '21.66 kmpl', '17.2 kmpl', '20.63 kmpl',
       '19.4 kmpl', '14.8 kmpl', '26.0 kmpl', '20.4 kmpl', '21.5 kmpl',
       '15.3 kmpl', '17.9 kmpl', '16.6 kmpl', '22.54 kmpl', '25.44 kmpl',
       '13.7 kmpl', '22.48 kmpl', '12.9 kmpl', '19.98 kmpl', '21.4 kmpl',
       '19.81 kmpl', '15.4 kmpl', '25.47 kmpl', '19.87 kmpl',
       '17.45 kmpl', '14.7 kmpl', '15.64 kmpl', '15.73 kmpl',
       '23.59 kmpl', '16.1 kmpl', '27.4 kmpl', '20.46 kmpl', '15.29 kmpl',
       '20.51 kmpl', '11.8 kmpl', '14.3 kmpl', '14.67 kmpl', '17.19 kmpl',
       '21.03 kmpl', '22.5 kmpl', '16.82 kmpl', '11.72 kmpl', '17.4 kmpl',
       '17.05 kmpl', '24.0 kmpl', '28.09 kmpl', '20.5 kmpl', '13.1 kmpl',
       '19.91 kmpl', '18.7 kmpl', '16.38 kmpl', '11.57 kmpl', '17.3 kmpl',
       '22.95 kmpl', '18.88 kmpl', '23.4 kmpl', '22.74 kmpl',
       '12.07 kmpl', '17.1 kmpl', '18.48 kmpl', '16.47 kmpl', '23.1 kmpl',
       '14.07 kmpl', '16.02 kmpl', '19.3 kmpl', '17.7 kmpl', '9.52 kmpl',
       '14.75 kmpl', '26.3 km/kg', '11.3 kmpl', '21.12 kmpl',
       '21.02 kmpl', '14.45 kmpl', '19.33 kmpl', '13.8 kmpl', '24.7 kmpl',
       '11.0 kmpl', '11.07 kmpl', '21.43 kmpl', '14.21 kmpl',
       '18.86 kmpl', '16.07 kmpl', '13.49 kmpl', '20.38 kmpl',
       '12.0 kmpl', '17.01 kmpl', '13.2 kmpl', '20.37 kmpl', '15.1 km/kg',
       '15.96 kmpl', '14.16 kmpl', '13.17 kmpl', '27.62 kmpl',
       '25.1 kmpl', '15.17 kmpl', '11.33 kmpl', '17.92 kmpl',
       '12.55 kmpl', '12.6 kmpl', '17.72 kmpl', '18.16 kmpl',
       '15.68 kmpl', '15.5 kmpl', '12.1 kmpl', '14.83 kmpl', '17.6 kmpl',
       '14.6 kmpl', '14.66 kmpl', '10.93 kmpl', '20.68 kmpl', '9.9 kmpl',
       '21.13 kmpl', '20.14 kmpl', '19.2 kmpl', '27.3 kmpl', '16.36 kmpl',
       '26.59 kmpl', '12.5 kmpl', '13.6 kmpl', '15.06 kmpl', '10.13 kmpl',
       '17.21 kmpl', '15.97 kmpl', '10.5 kmpl', '14.69 kmpl', '23.9 kmpl',
       '19.1 kmpl', '21.27 kmpl', '15.9 kmpl', '20.7 kmpl', '14.1 kmpl',
       '20.89 kmpl', '18.12 kmpl', '12.3 kmpl', '19.71 kmpl', '9.43 kmpl',
       '13.4 kmpl', '13.14 kmpl', '18.1 kmpl', '22.77 kmpl', '14.49 kmpl',
       '12.39 kmpl', '10.91 kmpl', '20.85 kmpl', '15.63 kmpl',
       '27.39 kmpl', '18.3 kmpl', '16.78 kmpl', '25.5 kmpl', '10.0 kmpl',
       '13.73 kmpl', '24.2 kmpl', '14.02 kmpl', '26.83 km/kg',
       '16.77 kmpl', '24.5 kmpl', '20.34 kmpl', '21.7 kmpl', '9.7 kmpl',
       '14.33 kmpl', '21.64 kmpl', '13.2 km/kg', '19.16 kmpl',
       '16.93 kmpl', '9.0 kmpl', '26.2 km/kg', '16.3 kmpl', '12.62 kmpl',
       '17.3 km/kg', '20.64 kmpl', '14.24 kmpl', '18.06 kmpl',
       '10.2 kmpl', '10.1 kmpl', '18.25 kmpl', '13.93 kmpl', '25.83 kmpl',
       '8.6 kmpl', '13.24 kmpl', '17.09 kmpl', '23.84 kmpl', '8.45 kmpl',
       '19.6 kmpl', '19.5 kmpl', '20.3 kmpl', '16.05 kmpl', '11.2 kmpl',
       '27.03 kmpl', '18.78 kmpl', '12.35 kmpl', '14.59 kmpl',
       '17.32 kmpl', '14.95 kmpl', '13.22 kmpl', '23.03 kmpl',
       '33.44 km/kg', '15.6 kmpl', '19.12 kmpl', '10.98 kmpl',
```

```
       '33.54 km/kg', '16.46 kmpl', '18.4 kmpl', '11.1 kmpl',
       '13.01 kmpl', '18.8 kmpl', '16.52 kmpl', '18.44 kmpl',
       '19.49 kmpl', '23.5 kmpl', '23.8 kmpl', '12.65 kmpl', '20.65 kmpl',
       '21.72 kmpl', '12.19 kmpl', '26.1 kmpl', '18.33 kmpl',
       '12.81 kmpl', '17.5 km/kg', '17.06 kmpl', '17.67 kmpl',
       '19.34 kmpl', '8.3 kmpl', '16.96 kmpl', '11.79 kmpl', '20.86 kmpl',
       '16.98 kmpl', '11.68 kmpl', '15.74 kmpl', '15.7 kmpl',
       '18.49 kmpl', '10.9 kmpl', '19.59 kmpl', '11.4 kmpl', '13.06 kmpl',
       '21.0 kmpl', '15.15 kmpl', '16.9 kmpl', '18.23 kmpl', '25.0 km/kg',
       '17.16 kmpl', '17.43 kmpl', '19.08 kmpl', '18.56 kmpl',
       '11.9 kmpl', '24.6 km/kg', '21.79 kmpl', '12.95 kmpl', '25.6 kmpl',
       '13.45 km/kg', '26.21 kmpl', '13.58 kmpl', '16.25 kmpl',
       '10.4 kmpl', '17.44 kmpl', '19.2 km/kg', '22.71 kmpl',
       '17.54 kmpl', '22.1 km/kg', '17.0 km/kg', '15.87 kmpl', '9.5 kmpl',
       '11.56 kmpl', '14.39 kmpl', '19.09 kmpl', '17.85 kmpl',
       '31.79 km/kg', '18.18 kmpl', '21.19 kmpl', '21.8 kmpl',
       '15.42 kmpl', '14.47 kmpl', '19.69 kmpl', '12.83 kmpl', '8.0 kmpl',
       '22.8 km/kg', '12.63 kmpl', '14.57 kmpl', '27.28 kmpl',
       '15.41 kmpl', '32.26 km/kg', '18.19 kmpl', '13.33 kmpl',
       '16.7 kmpl', '17.84 kmpl', '20.0 km/kg', '23.19 kmpl',
       '11.49 kmpl', '18.51 kmpl', '13.44 kmpl', '8.7 kmpl', '8.77 kmpl',
       '17.97 kmpl', '23.57 kmpl', '12.37 kmpl', '9.1 kmpl', '12.51 kmpl',
       '19.44 kmpl', '21.38 kmpl', '16.51 kmpl', '24.8 kmpl',
       '14.42 kmpl', '14.53 km/kg', '26.8 kmpl', '24.04 kmpl', '9.8 kmpl',
       '19.68 kmpl', '21.4 km/kg', '21.2 kmpl', '19.72 kmpl', '14.2 kmpl',
       '12.98 kmpl', '23.01 kmpl', '16.12 kmpl', '9.3 kmpl', '15.85 kmpl',
       nan, '17.88 kmpl', '10.6 kmpl', '11.78 kmpl', '7.94 kmpl',
       '25.01 kmpl', '8.1 kmpl', '13.9 kmpl', '11.62 kmpl', '20.62 kmpl',
       '15.11 kmpl', '10.37 kmpl', '18.59 kmpl', '9.74 kmpl',
       '14.81 kmpl', '8.2 kmpl', '12.97 kmpl', '7.5 kmpl', '30.46 km/kg',
       '6.4 kmpl', '12.85 kmpl', '18.69 kmpl', '17.24 kmpl'], dtype=object)
```

Заметим, что для машин на бензине/дизеле и для машин на газовом топливе единица измерения строго закреплена, а для электрических машин расход не указан вообще.

In [13]:

```
for row in lab2_array:
    if (row[3] == "Electric" and isinstance(row[6], float)):
        print(row[6])
    if (row[3] == "CNG" and row[6][-5:] != "km/kg"):
        print("False")
        break;
    if (row[3] == "LPG" and row[6][-5:] != "km/kg"):
        print("False")
        break;
    if (row[3] == "Petrol" and row[6][-4:] != "kmpl"):
        print("False")
        break;
    if (row[3] == "Diesel" and row[6][-4:] != "kmpl"):
        print("False")
        break;
else:
    print("True")
```

```
nan
nan
True
```

В связи с этим сделаем следующее:

1. Удалим две записи с электрическими автомобилями, поскольку этого все равно недостаточно для обучения и есть риск сбить модель с толку;
2. Удалим описание единицы измерения из записей про автомобили на углеводородах.

In [14]:

```
electric_indexes = []

for rowNum in range(lab2_array.shape[0]):
    if (lab2_array[rowNum][3] == "Electric"):
        electric_indexes.append(rowNum)
    elif (lab2_array[rowNum][3] == "CNG" or lab2_array[rowNum][3] == "LPG"):
        lab2_array[rowNum][6] = float(lab2_array[rowNum][6][:-5])
    elif (lab2_array[rowNum][3] == "Petrol" or lab2_array[rowNum][3] == "Diesel"):
        lab2_array[rowNum][6] = float(lab2_array[rowNum][6][:-4])

lab2_array = np.delete(lab2_array, electric_indexes, 0)
```

Engine и Power

Здесь единицы измерения полностью согласованы. Необходимо отрезать их от значения признака и получить численное представление

In [15]:

```
for rowNum in range(lab2_array.shape[0]):
    if (isinstance(lab2_array[rowNum][7], str)):
        if (lab2_array[rowNum][7] != "nan"):
            lab2_array[rowNum][7] = float(lab2_array[rowNum][7][:-2])
        else:
            lab2_array[rowNum][7] = float('nan')

    if (isinstance(lab2_array[rowNum][8], str)):
        if (lab2_array[rowNum][8] != "null bhp"):
            lab2_array[rowNum][8] = float(lab2_array[rowNum][8][:-3])
        else:
            lab2_array[rowNum][8] = float('nan')
```

Seats

Здесь только преобразовать в int

In [16]:

```
for rowNum in range(lab2_array.shape[0]):
    if not (np.isnan(lab2_array[rowNum][9])):
        lab2_array[rowNum][9] = int(lab2_array[rowNum][9])
```

Получение датафрейма

Посмотрим на то, какие значения теперь принимают столбцы:

In [17]:

```
for i in range(lab2_array.shape[1]):
    print(str(i) + ":", lab2_array[:, i])
```

```
0: ['Mumbai' 'Pune' 'Chennai' ... 'Jaipur' 'Kolkata' 'Hyderabad']
1: [2010 2015 2011 ... 2012 2013 2011]
2: [72000 41000 46000 ... 55000 46000 47000]
3: ['CNG' 'Diesel' 'Petrol' ... 'Diesel' 'Petrol' 'Diesel']
4: ['Manual' 'Manual' 'Manual' ... 'Manual' 'Manual' 'Manual']
5: [1 1 1 ... 2 1 1]
6: [26.6 19.67 18.2 ... 14.0 18.9 25.44]
7: [998.0 1582.0 1199.0 ... 2498.0 998.0 936.0]
8: [58.16 126.2 88.7 ... 112.0 67.1 57.6]
9: [5 5 5 ... 8 5 5]
10: [1.75 12.5 4.5 ... 2.9 2.65 2.5]
```

In [18]:

```
new_columns = lab2_data.columns.delete([12, 1, 0])
#print(new_columns)
lab2_data_new = pd.DataFrame(lab2_array, columns = new_columns)
```

In [19]:

```
lab2_data_new.head(20)
```

Out[19]:

	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price
0	Mumbai	2010	72000	CNG	Manual	1	26.6	998	58.16	5	1.75
1	Pune	2015	41000	Diesel	Manual	1	19.67	1582	126.2	5	12.5
2	Chennai	2011	46000	Petrol	Manual	1	18.2	1199	88.7	5	4.5
3	Chennai	2012	87000	Diesel	Manual	1	20.77	1248	88.76	7	6
4	Coimbatore	2013	40670	Diesel	Automatic	2	15.2	1968	140.8	5	17.74
5	Hyderabad	2012	75000	LPG	Manual	1	21.1	814	55.2	5	2.35
6	Jaipur	2013	86999	Diesel	Manual	1	23.08	1461	63.1	5	3.5
7	Mumbai	2016	36000	Diesel	Automatic	1	11.36	2755	171.5	8	17.5
8	Pune	2013	64430	Diesel	Manual	1	20.54	1598	103.6	5	5.2
9	Chennai	2012	65932	Diesel	Manual	2	22.3	1248	74	5	1.95
10	Kochi	2018	25692	Petrol	Manual	1	21.56	1462	103.25	5	9.95
11	Kolkata	2012	60000	Petrol	Automatic	1	16.8	1497	116.3	5	4.49
12	Jaipur	2015	64424	Diesel	Manual	1	25.2	1248	74	5	5.6
13	Delhi	2014	72000	Diesel	Automatic	1	12.7	2179	187.7	5	27
14	Pune	2012	85000	Diesel	Automatic	2	0	2179	115	5	17.5
15	Delhi	2014	110000	Diesel	Manual	1	13.5	2477	175.56	7	15
16	Kochi	2016	58950	Diesel	Manual	1	25.8	1498	98.6	5	5.4
17	Jaipur	2017	25000	Diesel	Manual	1	28.4	1248	74	5	5.99
18	Kochi	2014	77469	Diesel	Manual	1	20.45	1461	83.8	5	6.34
19	Bangalore	2014	78500	Diesel	Automatic	1	14.84	2143	167.62	5	28

In [20]:

```
PrintDatasetInfo(lab2_data_new)
```

```
Столбец Location (тип object) имеет 0 пропусков (индекс 0)

Столбец Year (тип object) имеет 0 пропусков (индекс 1)

Столбец Kilometers_Driven (тип object) имеет 0 пропусков (индекс 2)

Столбец Fuel_Type (тип object) имеет 0 пропусков (индекс 3)

Столбец Transmission (тип object) имеет 0 пропусков (индекс 4)

Столбец Owner_Type (тип object) имеет 0 пропусков (индекс 5)

Столбец Mileage (тип object) имеет 0 пропусков (индекс 6)

Столбец Engine (тип object) имеет 36 пропусков (индекс 7)

Столбец Power (тип object) имеет 143 пропусков (индекс 8)

Столбец Seats (тип object) имеет 42 пропусков (индекс 9)

Столбец Price (тип object) имеет 0 пропусков (индекс 10)
```

У числовых столбцов не указан тип. Проведём доп. преобразование:

In [21]:

```
data = lab2_data_new.infer_objects()
print(data.dtypes)
```

```
Location             object
Year                  int64
Kilometers_Driven     int64
Fuel_Type            object
Transmission         object
Owner_Type            int64
Mileage             float64
Engine              float64
Power               float64
Seats               float64
Price               float64
dtype: object
```

In [22]:

```
data.head(20)
```

Out[22]:

	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price
0	Mumbai	2010	72000	CNG	Manual	1	26.60	998.0	58.16	5.0	1.75
1	Pune	2015	41000	Diesel	Manual	1	19.67	1582.0	126.20	5.0	12.50
2	Chennai	2011	46000	Petrol	Manual	1	18.20	1199.0	88.70	5.0	4.50
3	Chennai	2012	87000	Diesel	Manual	1	20.77	1248.0	88.76	7.0	6.00
4	Coimbatore	2013	40670	Diesel	Automatic	2	15.20	1968.0	140.80	5.0	17.74
5	Hyderabad	2012	75000	LPG	Manual	1	21.10	814.0	55.20	5.0	2.35
6	Jaipur	2013	86999	Diesel	Manual	1	23.08	1461.0	63.10	5.0	3.50
7	Mumbai	2016	36000	Diesel	Automatic	1	11.36	2755.0	171.50	8.0	17.50
8	Pune	2013	64430	Diesel	Manual	1	20.54	1598.0	103.60	5.0	5.20
9	Chennai	2012	65932	Diesel	Manual	2	22.30	1248.0	74.00	5.0	1.95
10	Kochi	2018	25692	Petrol	Manual	1	21.56	1462.0	103.25	5.0	9.95
11	Kolkata	2012	60000	Petrol	Automatic	1	16.80	1497.0	116.30	5.0	4.49
12	Jaipur	2015	64424	Diesel	Manual	1	25.20	1248.0	74.00	5.0	5.60
13	Delhi	2014	72000	Diesel	Automatic	1	12.70	2179.0	187.70	5.0	27.00
14	Pune	2012	85000	Diesel	Automatic	2	0.00	2179.0	115.00	5.0	17.50
15	Delhi	2014	110000	Diesel	Manual	1	13.50	2477.0	175.56	7.0	15.00
16	Kochi	2016	58950	Diesel	Manual	1	25.80	1498.0	98.60	5.0	5.40
17	Jaipur	2017	25000	Diesel	Manual	1	28.40	1248.0	74.00	5.0	5.99
18	Kochi	2014	77469	Diesel	Manual	1	20.45	1461.0	83.80	5.0	6.34
19	Bangalore	2014	78500	Diesel	Automatic	1	14.84	2143.0	167.62	5.0	28.00

3. Обработка пропусков в данных

In [23]:

```
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
```

In [24]:

```
data.shape
```

Out[24]:

```
(6017, 11)
```

Метод для сведений о встречающихся значениях в столбце:

In [25]:

```
import math

def GetValuesInfo(data, columnName):
    value_count_list = list()
    print("Для датафрейма {0}, столбец {1}:".format(data.name, columnName))
    sum = 0
    uniqueCount = 0

    # Формируем список из кортежей
    for value in data[columnName].unique():
        uniqueCount = uniqueCount + 1
        if (isinstance(value, float) and math.isnan(value)):
            temp = data[pd.isna(data[columnName])]
        else:
            temp = data[data[columnName] == value]
            sum = sum + temp.shape[0]

        value_count_list.append((value, temp.shape[0]))

    # Сортируем по убыванию на основе второго поля кортежа
    value_count_list = sorted(value_count_list, reverse = True, key = lambda x: x[1])

    # Смотрим на результат
```

```
    for element in value_count_list:
        print(element[0], "->", element[1])
    print("Заполненных значений:", sum, "(из {})".format(data.shape[0]))
    print("Уникальных значений:", uniqueCount)
```

Метод для заполнения пропусков:

In [26]:

```
def ColumnImputer(data, columnName, strategyName):
    data_column = data[[columnName]]
    mask = MissingIndicator().fit_transform(data_column)
    imputer = SimpleImputer(strategy = strategyName)
    column_imputed = imputer.fit_transform(data_column)
    return column_imputed
```

Seats

In [27]:

```
data.name = "data"
GetValuesInfo(data, "Seats")
```

```
Для датафрейма data, столбец Seats:
5.0 -> 5012
7.0 -> 674
8.0 -> 134
4.0 -> 99
nan -> 42
6.0 -> 31
2.0 -> 16
10.0 -> 5
9.0 -> 3
0.0 -> 1
Заполненных значений: 5975 (из 6017)
Уникальных значений: 10
```

У авто не может быть нецелое количество кресел, поэтому будет заполнять медиану, а не среднее выборочное:

In [28]:

```
new_seats = ColumnImputer(data, "Seats", "median")
data["Seats"] = new_seats
GetValuesInfo(data, "Seats")
```

```
Для датафрейма data, столбец Seats:
5.0 -> 5054
7.0 -> 674
8.0 -> 134
4.0 -> 99
6.0 -> 31
2.0 -> 16
10.0 -> 5
9.0 -> 3
0.0 -> 1
Заполненных значений: 6017 (из 6017)
Уникальных значений: 9
```

Исправлено!

Power

In [29]:

```
GetValuesInfo(data, "Power")
new_power = ColumnImputer(data, "Power", "median")
data["Power"] = new_power
```

```
Для датафрейма data, столбец Power:
74.0 -> 235
nan -> 143
98.6 -> 131
73.9 -> 125
140.0 -> 123
88.5 -> 112
78.9 -> 111
67.1 -> 107
67.04 -> 107
82.0 -> 101
117.3 -> 93
118.0 -> 90
121.3 -> 88
85.8 -> 82
190.0 -> 79
```

126.2 -> 78
170.0 -> 77
88.7 -> 75
70.0 -> 75
80.0 -> 74
86.8 -> 74
174.33 -> 71
81.86 -> 71
103.6 -> 69
81.83 -> 68
68.0 -> 62
68.05 -> 61
184.0 -> 58
102.0 -> 57
120.0 -> 57
88.8 -> 57
55.2 -> 56
100.0 -> 55
90.0 -> 52
108.45 -> 50
126.32 -> 49
88.73 -> 48
47.3 -> 48
168.5 -> 47
88.76 -> 45
67.0 -> 45
138.1 -> 44
81.8 -> 44
83.1 -> 44
86.7 -> 43
46.3 -> 42
75.0 -> 40
89.84 -> 40
241.4 -> 40
83.8 -> 39
100.6 -> 39
258.0 -> 38
108.5 -> 37
103.2 -> 35
187.7 -> 34
138.03 -> 34
103.5 -> 30
62.1 -> 30
171.0 -> 29
147.51 -> 29
53.3 -> 28
91.1 -> 28
181.0 -> 28
73.75 -> 28
110.0 -> 26
103.52 -> 26
136.0 -> 26
116.3 -> 25
130.0 -> 25
171.5 -> 23
167.62 -> 23
105.0 -> 23
147.8 -> 22
73.94 -> 22
71.0 -> 21
187.74 -> 21
157.75 -> 21
93.7 -> 21
141.0 -> 21
204.0 -> 21
112.0 -> 19
177.0 -> 19
79.4 -> 19
64.0 -> 18
78.0 -> 18
99.0 -> 18
84.8 -> 18
174.5 -> 18
35.0 -> 17
73.0 -> 17
194.3 -> 17
66.1 -> 17

203.0 -> 17
94.0 -> 17
174.3 -> 17
69.0 -> 16
57.6 -> 16
62.0 -> 16
60.0 -> 15
63.1 -> 14
177.6 -> 14
98.59 -> 14
73.97 -> 14
58.16 -> 13
186.0 -> 13
215.0 -> 13
245.0 -> 13
83.14 -> 13
153.86 -> 13
88.2 -> 13
270.9 -> 12
37.48 -> 12
125.0 -> 12
37.0 -> 12
110.4 -> 12
163.7 -> 11
152.0 -> 11
103.3 -> 11
138.0 -> 11
218.0 -> 10
107.3 -> 10
85.0 -> 10
224.0 -> 10
66.0 -> 10
198.5 -> 10
147.9 -> 10
197.0 -> 10
58.2 -> 10
126.24 -> 10
84.0 -> 10
157.7 -> 10
175.56 -> 9
37.5 -> 9
82.85 -> 9
150.0 -> 9
143.0 -> 9
67.06 -> 9
235.0 -> 9
87.2 -> 9
63.0 -> 9
98.96 -> 9
120.7 -> 8
102.5 -> 8
177.46 -> 8
254.79 -> 8
169.0 -> 8
88.0 -> 8
113.98 -> 8
126.3 -> 8
165.0 -> 8
69.01 -> 8
140.8 -> 7
115.0 -> 7
179.5 -> 7
254.8 -> 7
82.9 -> 7
201.1 -> 7
106.0 -> 7
65.0 -> 7
189.08 -> 7
183.0 -> 7
201.15 -> 7
154.0 -> 7
107.2 -> 7
34.2 -> 7
104.68 -> 7
97.7 -> 7
180.0 -> 7
210.0 -> 7

142.0 -> 7
83.0 -> 7
77.0 -> 6
132.0 -> 6
178.0 -> 6
76.8 -> 6
123.24 -> 6
47.0 -> 6
335.2 -> 6
306.0 -> 6
92.7 -> 6
241.38 -> 6
130.2 -> 6
121.4 -> 6
121.36 -> 6
265.0 -> 6
255.0 -> 6
203.2 -> 6
272.0 -> 6
103.25 -> 5
194.0 -> 5
76.0 -> 5
313.0 -> 5
141.1 -> 5
99.6 -> 5
160.0 -> 5
197.2 -> 5
167.6 -> 5
53.64 -> 5
158.2 -> 5
158.0 -> 5
157.8 -> 5
123.7 -> 5
56.3 -> 5
167.7 -> 5
74.9 -> 5
300.0 -> 5
149.92 -> 4
75.94 -> 4
53.5 -> 4
67.05 -> 4
282.0 -> 4
108.4 -> 4
148.0 -> 4
108.62 -> 4
163.5 -> 4
91.72 -> 4
40.3 -> 4
101.0 -> 4
63.12 -> 4
147.5 -> 4
134.0 -> 4
147.6 -> 3
64.1 -> 3
177.5 -> 3
164.7 -> 3
73.8 -> 3
82.5 -> 3
364.9 -> 3
185.0 -> 3
66.7 -> 3
116.6 -> 3
118.3 -> 3
181.03 -> 3
308.0 -> 3
86.79 -> 3
158.8 -> 3
80.46 -> 3
138.08 -> 3
91.2 -> 3
97.6 -> 3
73.74 -> 3
105.5 -> 3
94.68 -> 3
163.2 -> 3
241.0 -> 3
55.0 -> 3

86.76 -> 3
162.0 -> 3
82.4 -> 3
224.34 -> 3
92.0 -> 3
64.08 -> 3
177.01 -> 2
362.07 -> 2
444.0 -> 2
246.7 -> 2
122.0 -> 2
114.0 -> 2
191.34 -> 2
108.49 -> 2
138.02 -> 2
187.4 -> 2
207.8 -> 2
155.0 -> 2
38.4 -> 2
57.5 -> 2
65.3 -> 2
93.0 -> 2
91.7 -> 2
189.0 -> 2
104.0 -> 2
148.31 -> 2
250.0 -> 2
102.57 -> 2
102.53 -> 2
240.0 -> 2
108.495 -> 2
320.0 -> 2
70.02 -> 2
261.49 -> 2
168.7 -> 2
55.23 -> 2
200.0 -> 2
271.23 -> 2
85.7 -> 2
308.43 -> 2
118.6 -> 2
83.83 -> 2
262.6 -> 2
163.0 -> 2
58.33 -> 2
394.3 -> 2
335.3 -> 2
198.25 -> 2
53.0 -> 2
193.1 -> 2
450.0 -> 2
52.8 -> 2
139.01 -> 2
208.0 -> 2
98.82 -> 2
112.2 -> 1
500.0 -> 1
362.9 -> 1
227.0 -> 1
192.0 -> 1
246.74 -> 1
367.0 -> 1
98.79 -> 1
156.0 -> 1
181.43 -> 1
237.4 -> 1
226.6 -> 1
321.0 -> 1
35.5 -> 1
114.4 -> 1
270.88 -> 1
236.0 -> 1
254.0 -> 1
112.4 -> 1
116.9 -> 1
550.0 -> 1
152.88 -> 1

```
95.0 -> 1
199.3 -> 1
201.0 -> 1
76.9 -> 1
174.57 -> 1
301.73 -> 1
68.1 -> 1
80.9 -> 1
340.0 -> 1
120.33 -> 1
231.1 -> 1
333.0 -> 1
402.0 -> 1
261.0 -> 1
61.0 -> 1
144.0 -> 1
71.01 -> 1
271.72 -> 1
135.1 -> 1
261.5 -> 1
123.37 -> 1
175.67 -> 1
110.5 -> 1
178.4 -> 1
395.0 -> 1
48.21 -> 1
421.0 -> 1
89.75 -> 1
387.3 -> 1
130.3 -> 1
281.61 -> 1
503.0 -> 1
168.0 -> 1
139.07 -> 1
83.11 -> 1
74.93 -> 1
382.0 -> 1
74.96 -> 1
552.0 -> 1
127.0 -> 1
560.0 -> 1
116.4 -> 1
161.6 -> 1
488.1 -> 1
103.0 -> 1
181.04 -> 1
Заполненных значений: 5874 (из 6017)
Уникальных значений: 369
```

In [30]:

```
GetValuesInfo(data, "Power")
```

```
Для датафрейма data, столбец Power:
74.0 -> 235
97.7 -> 150
98.6 -> 131
73.9 -> 125
140.0 -> 123
88.5 -> 112
78.9 -> 111
67.1 -> 107
67.04 -> 107
82.0 -> 101
117.3 -> 93
118.0 -> 90
121.3 -> 88
85.8 -> 82
190.0 -> 79
126.2 -> 78
170.0 -> 77
88.7 -> 75
70.0 -> 75
80.0 -> 74
86.8 -> 74
174.33 -> 71
81.86 -> 71
103.6 -> 69
81.83 -> 68
```

68.0 -> 62
68.05 -> 61
184.0 -> 58
102.0 -> 57
120.0 -> 57
88.8 -> 57
55.2 -> 56
100.0 -> 55
90.0 -> 52
108.45 -> 50
126.32 -> 49
88.73 -> 48
47.3 -> 48
168.5 -> 47
88.76 -> 45
67.0 -> 45
138.1 -> 44
81.8 -> 44
83.1 -> 44
86.7 -> 43
46.3 -> 42
75.0 -> 40
89.84 -> 40
241.4 -> 40
83.8 -> 39
100.6 -> 39
258.0 -> 38
108.5 -> 37
103.2 -> 35
187.7 -> 34
138.03 -> 34
103.5 -> 30
62.1 -> 30
171.0 -> 29
147.51 -> 29
53.3 -> 28
91.1 -> 28
181.0 -> 28
73.75 -> 28
110.0 -> 26
103.52 -> 26
136.0 -> 26
116.3 -> 25
130.0 -> 25
171.5 -> 23
167.62 -> 23
105.0 -> 23
147.8 -> 22
73.94 -> 22
71.0 -> 21
187.74 -> 21
157.75 -> 21
93.7 -> 21
141.0 -> 21
204.0 -> 21
112.0 -> 19
177.0 -> 19
79.4 -> 19
64.0 -> 18
78.0 -> 18
99.0 -> 18
84.8 -> 18
174.5 -> 18
35.0 -> 17
73.0 -> 17
194.3 -> 17
66.1 -> 17
203.0 -> 17
94.0 -> 17
174.3 -> 17
69.0 -> 16
57.6 -> 16
62.0 -> 16
60.0 -> 15
63.1 -> 14
177.6 -> 14
98.59 -> 14

73.97 -> 14
58.16 -> 13
186.0 -> 13
215.0 -> 13
245.0 -> 13
83.14 -> 13
153.86 -> 13
88.2 -> 13
270.9 -> 12
37.48 -> 12
125.0 -> 12
37.0 -> 12
110.4 -> 12
163.7 -> 11
152.0 -> 11
103.3 -> 11
138.0 -> 11
218.0 -> 10
107.3 -> 10
85.0 -> 10
224.0 -> 10
66.0 -> 10
198.5 -> 10
147.9 -> 10
197.0 -> 10
58.2 -> 10
126.24 -> 10
84.0 -> 10
157.7 -> 10
175.56 -> 9
37.5 -> 9
82.85 -> 9
150.0 -> 9
143.0 -> 9
67.06 -> 9
235.0 -> 9
87.2 -> 9
63.0 -> 9
98.96 -> 9
120.7 -> 8
102.5 -> 8
177.46 -> 8
254.79 -> 8
169.0 -> 8
88.0 -> 8
113.98 -> 8
126.3 -> 8
165.0 -> 8
69.01 -> 8
140.8 -> 7
115.0 -> 7
179.5 -> 7
254.8 -> 7
82.9 -> 7
201.1 -> 7
106.0 -> 7
65.0 -> 7
189.08 -> 7
183.0 -> 7
201.15 -> 7
154.0 -> 7
107.2 -> 7
34.2 -> 7
104.68 -> 7
180.0 -> 7
210.0 -> 7
142.0 -> 7
83.0 -> 7
77.0 -> 6
132.0 -> 6
178.0 -> 6
76.8 -> 6
123.24 -> 6
47.0 -> 6
335.2 -> 6
306.0 -> 6
92.7 -> 6

241.38 -> 6
130.2 -> 6
121.4 -> 6
121.36 -> 6
265.0 -> 6
255.0 -> 6
203.2 -> 6
272.0 -> 6
103.25 -> 5
194.0 -> 5
76.0 -> 5
313.0 -> 5
141.1 -> 5
99.6 -> 5
160.0 -> 5
197.2 -> 5
167.6 -> 5
53.64 -> 5
158.2 -> 5
158.0 -> 5
157.8 -> 5
123.7 -> 5
56.3 -> 5
167.7 -> 5
74.9 -> 5
300.0 -> 5
149.92 -> 4
75.94 -> 4
53.5 -> 4
67.05 -> 4
282.0 -> 4
108.4 -> 4
148.0 -> 4
108.62 -> 4
163.5 -> 4
91.72 -> 4
40.3 -> 4
101.0 -> 4
63.12 -> 4
147.5 -> 4
134.0 -> 4
147.6 -> 3
64.1 -> 3
177.5 -> 3
164.7 -> 3
73.8 -> 3
82.5 -> 3
364.9 -> 3
185.0 -> 3
66.7 -> 3
116.6 -> 3
118.3 -> 3
181.03 -> 3
308.0 -> 3
86.79 -> 3
158.8 -> 3
80.46 -> 3
138.08 -> 3
91.2 -> 3
97.6 -> 3
73.74 -> 3
105.5 -> 3
94.68 -> 3
163.2 -> 3
241.0 -> 3
55.0 -> 3
86.76 -> 3
162.0 -> 3
82.4 -> 3
224.34 -> 3
92.0 -> 3
64.08 -> 3
177.01 -> 2
362.07 -> 2
444.0 -> 2
246.7 -> 2
122.0 -> 2

114.0 -> 2
191.34 -> 2
108.49 -> 2
138.02 -> 2
187.4 -> 2
207.8 -> 2
155.0 -> 2
38.4 -> 2
57.5 -> 2
65.3 -> 2
93.0 -> 2
91.7 -> 2
189.0 -> 2
104.0 -> 2
148.31 -> 2
250.0 -> 2
102.57 -> 2
102.53 -> 2
240.0 -> 2
108.495 -> 2
320.0 -> 2
70.02 -> 2
261.49 -> 2
168.7 -> 2
55.23 -> 2
200.0 -> 2
271.23 -> 2
85.7 -> 2
308.43 -> 2
118.6 -> 2
83.83 -> 2
262.6 -> 2
163.0 -> 2
58.33 -> 2
394.3 -> 2
335.3 -> 2
198.25 -> 2
53.0 -> 2
193.1 -> 2
450.0 -> 2
52.8 -> 2
139.01 -> 2
208.0 -> 2
98.82 -> 2
112.2 -> 1
500.0 -> 1
362.9 -> 1
227.0 -> 1
192.0 -> 1
246.74 -> 1
367.0 -> 1
98.79 -> 1
156.0 -> 1
181.43 -> 1
237.4 -> 1
226.6 -> 1
321.0 -> 1
35.5 -> 1
114.4 -> 1
270.88 -> 1
236.0 -> 1
254.0 -> 1
112.4 -> 1
116.9 -> 1
550.0 -> 1
152.88 -> 1
95.0 -> 1
199.3 -> 1
201.0 -> 1
76.9 -> 1
174.57 -> 1
301.73 -> 1
68.1 -> 1
80.9 -> 1
340.0 -> 1
120.33 -> 1
231.1 -> 1

```
333.0 -> 1
402.0 -> 1
261.0 -> 1
61.0 -> 1
144.0 -> 1
71.01 -> 1
271.72 -> 1
135.1 -> 1
261.5 -> 1
123.37 -> 1
175.67 -> 1
110.5 -> 1
178.4 -> 1
395.0 -> 1
48.21 -> 1
421.0 -> 1
89.75 -> 1
387.3 -> 1
130.3 -> 1
281.61 -> 1
503.0 -> 1
168.0 -> 1
139.07 -> 1
83.11 -> 1
74.93 -> 1
382.0 -> 1
74.96 -> 1
552.0 -> 1
127.0 -> 1
560.0 -> 1
116.4 -> 1
161.6 -> 1
488.1 -> 1
103.0 -> 1
181.04 -> 1
Заполненных значений: 6017 (из 6017)
Уникальных значений: 368
```
Исправлено!

Engine

In [31]:

```
GetValuesInfo(data, "Engine")
new_engine = ColumnImputer(data, "Engine", "median")
data["Engine"] = new_engine
```

```
Для датафрейма data, столбец Engine:
1197.0 -> 606
1248.0 -> 512
1498.0 -> 304
998.0 -> 259
2179.0 -> 240
1497.0 -> 229
1198.0 -> 227
1968.0 -> 216
1995.0 -> 183
1461.0 -> 152
2143.0 -> 149
1582.0 -> 145
1199.0 -> 143
1598.0 -> 141
1396.0 -> 139
796.0 -> 129
2494.0 -> 121
1086.0 -> 108
1591.0 -> 94
2993.0 -> 90
1399.0 -> 88
2982.0 -> 86
1798.0 -> 84
2987.0 -> 67
2967.0 -> 61
814.0 -> 59
1120.0 -> 54
1196.0 -> 50
1493.0 -> 47
1373.0 -> 47
```

1364.0 -> 47
2354.0 -> 41
1298.0 -> 39
2755.0 -> 38
799.0 -> 36
nan -> 36
1991.0 -> 33
1799.0 -> 32
1896.0 -> 32
1061.0 -> 30
999.0 -> 29
1998.0 -> 28
624.0 -> 25
1796.0 -> 25
1496.0 -> 25
2393.0 -> 24
1193.0 -> 23
1586.0 -> 23
936.0 -> 21
1997.0 -> 21
1499.0 -> 21
1794.0 -> 20
2148.0 -> 19
1405.0 -> 19
1999.0 -> 18
1984.0 -> 17
2199.0 -> 17
2523.0 -> 17
2498.0 -> 16
1495.0 -> 15
3198.0 -> 15
993.0 -> 14
1956.0 -> 14
1186.0 -> 14
2499.0 -> 14
2696.0 -> 13
2497.0 -> 12
1599.0 -> 12
3498.0 -> 12
2477.0 -> 10
995.0 -> 10
1368.0 -> 10
1299.0 -> 10
2835.0 -> 10
1595.0 -> 9
2400.0 -> 9
1388.0 -> 9
1341.0 -> 9
1950.0 -> 8
4134.0 -> 8
2953.0 -> 7
1150.0 -> 7
2198.0 -> 7
2489.0 -> 7
1462.0 -> 6
2698.0 -> 6
2996.0 -> 6
2979.0 -> 6
2496.0 -> 5
1596.0 -> 5
1590.0 -> 5
2609.0 -> 5
4367.0 -> 5
4806.0 -> 4
1390.0 -> 4
793.0 -> 4
1343.0 -> 4
2446.0 -> 4
1047.0 -> 4
5461.0 -> 3
2956.0 -> 3
1395.0 -> 3
2362.0 -> 3
1969.0 -> 3
1172.0 -> 3
1194.0 -> 3
2359.0 -> 3

```
4395.0 -> 3
2894.0 -> 2
2360.0 -> 2
3436.0 -> 2
1242.0 -> 2
1781.0 -> 2
3597.0 -> 2
1985.0 -> 2
2997.0 -> 2
1597.0 -> 2
2771.0 -> 2
2200.0 -> 2
5000.0 -> 2
2694.0 -> 1
1978.0 -> 1
2706.0 -> 1
1422.0 -> 1
1489.0 -> 1
2495.0 -> 1
3200.0 -> 1
2773.0 -> 1
2147.0 -> 1
2999.0 -> 1
2995.0 -> 1
1948.0 -> 1
2349.0 -> 1
2720.0 -> 1
1468.0 -> 1
3197.0 -> 1
2487.0 -> 1
4951.0 -> 1
970.0 -> 1
2925.0 -> 1
2149.0 -> 1
5998.0 -> 1
2092.0 -> 1
5204.0 -> 1
2112.0 -> 1
1797.0 -> 1
Заполненных значений: 5981 (из 6017)
Уникальных значений: 146
```

In [32]:

```
GetValuesInfo(data, "Engine")
```

```
Для датафрейма data, столбец Engine:
1197.0 -> 606
1248.0 -> 512
1498.0 -> 304
998.0 -> 259
2179.0 -> 240
1497.0 -> 229
1198.0 -> 227
1968.0 -> 216
1995.0 -> 183
1461.0 -> 152
2143.0 -> 149
1582.0 -> 145
1199.0 -> 143
1598.0 -> 141
1396.0 -> 139
796.0 -> 129
2494.0 -> 121
1086.0 -> 108
1591.0 -> 94
2993.0 -> 90
1399.0 -> 88
2982.0 -> 86
1798.0 -> 84
1493.0 -> 83
2987.0 -> 67
2967.0 -> 61
814.0 -> 59
1120.0 -> 54
1196.0 -> 50
1373.0 -> 47
1364.0 -> 47
2354.0 -> 41
```

1298.0 -> 39
2755.0 -> 38
799.0 -> 36
1991.0 -> 33
1799.0 -> 32
1896.0 -> 32
1061.0 -> 30
999.0 -> 29
1998.0 -> 28
624.0 -> 25
1796.0 -> 25
1496.0 -> 25
2393.0 -> 24
1193.0 -> 23
1586.0 -> 23
936.0 -> 21
1997.0 -> 21
1499.0 -> 21
1794.0 -> 20
2148.0 -> 19
1405.0 -> 19
1999.0 -> 18
1984.0 -> 17
2199.0 -> 17
2523.0 -> 17
2498.0 -> 16
1495.0 -> 15
3198.0 -> 15
993.0 -> 14
1956.0 -> 14
1186.0 -> 14
2499.0 -> 14
2696.0 -> 13
2497.0 -> 12
1599.0 -> 12
3498.0 -> 12
2477.0 -> 10
995.0 -> 10
1368.0 -> 10
1299.0 -> 10
2835.0 -> 10
1595.0 -> 9
2400.0 -> 9
1388.0 -> 9
1341.0 -> 9
1950.0 -> 8
4134.0 -> 8
2953.0 -> 7
1150.0 -> 7
2198.0 -> 7
2489.0 -> 7
1462.0 -> 6
2698.0 -> 6
2996.0 -> 6
2979.0 -> 6
2496.0 -> 5
1596.0 -> 5
1590.0 -> 5
2609.0 -> 5
4367.0 -> 5
4806.0 -> 4
1390.0 -> 4
793.0 -> 4
1343.0 -> 4
2446.0 -> 4
1047.0 -> 4
5461.0 -> 3
2956.0 -> 3
1395.0 -> 3
2362.0 -> 3
1969.0 -> 3
1172.0 -> 3
1194.0 -> 3
2359.0 -> 3
4395.0 -> 3
2894.0 -> 2
2360.0 -> 2

```
3436.0 -> 2
1242.0 -> 2
1781.0 -> 2
3597.0 -> 2
1985.0 -> 2
2997.0 -> 2
1597.0 -> 2
2771.0 -> 2
2200.0 -> 2
5000.0 -> 2
2694.0 -> 1
1978.0 -> 1
2706.0 -> 1
1422.0 -> 1
1489.0 -> 1
2495.0 -> 1
3200.0 -> 1
2773.0 -> 1
2147.0 -> 1
2999.0 -> 1
2995.0 -> 1
1948.0 -> 1
2349.0 -> 1
2720.0 -> 1
1468.0 -> 1
3197.0 -> 1
2487.0 -> 1
4951.0 -> 1
970.0 -> 1
2925.0 -> 1
2149.0 -> 1
5998.0 -> 1
2092.0 -> 1
5204.0 -> 1
2112.0 -> 1
1797.0 -> 1
Заполненных значений: 6017 (из 6017)
Уникальных значений: 145
```

Промежуточный итог

In [33]:

```
PrintDatasetInfo(data)
```

```
Столбец Location (тип object) имеет 0 пропусков (индекс 0)

Столбец Year (тип int64) имеет 0 пропусков (индекс 1)

Столбец Kilometers_Driven (тип int64) имеет 0 пропусков (индекс 2)

Столбец Fuel_Type (тип object) имеет 0 пропусков (индекс 3)

Столбец Transmission (тип object) имеет 0 пропусков (индекс 4)

Столбец Owner_Type (тип int64) имеет 0 пропусков (индекс 5)

Столбец Mileage (тип float64) имеет 0 пропусков (индекс 6)

Столбец Engine (тип float64) имеет 0 пропусков (индекс 7)

Столбец Power (тип float64) имеет 0 пропусков (индекс 8)

Столбец Seats (тип float64) имеет 0 пропусков (индекс 9)

Столбец Price (тип float64) имеет 0 пропусков (индекс 10)
```

Все пропуски заполнены.

4. Кодирование категориальных признаков

In [34]:

```
GetValuesInfo(data, "Location")
```

```
Для датафрейма data, столбец Location:
Mumbai -> 789
Hyderabad -> 742
Kochi -> 651
Coimbatore -> 636
Pune -> 622
Delhi -> 554
Kolkata -> 535
Chennai -> 493
Jaipur -> 413
Bangalore -> 358
Ahmedabad -> 224
Заполненных значений: 6017 (из 6017)
Уникальных значений: 11
```

In [35]:

```
data = pd.get_dummies(data)
```

In [36]:

```
PrintDatasetInfo(data)
```

```
Столбец Year (тип int64) имеет 0 пропусков (индекс 0)

Столбец Kilometers_Driven (тип int64) имеет 0 пропусков (индекс 1)

Столбец Owner_Type (тип int64) имеет 0 пропусков (индекс 2)

Столбец Mileage (тип float64) имеет 0 пропусков (индекс 3)

Столбец Engine (тип float64) имеет 0 пропусков (индекс 4)

Столбец Power (тип float64) имеет 0 пропусков (индекс 5)

Столбец Seats (тип float64) имеет 0 пропусков (индекс 6)

Столбец Price (тип float64) имеет 0 пропусков (индекс 7)

Столбец Location_Ahmedabad (тип uint8) имеет 0 пропусков (индекс 8)

Столбец Location_Bangalore (тип uint8) имеет 0 пропусков (индекс 9)

Столбец Location_Chennai (тип uint8) имеет 0 пропусков (индекс 10)

Столбец Location_Coimbatore (тип uint8) имеет 0 пропусков (индекс 11)

Столбец Location_Delhi (тип uint8) имеет 0 пропусков (индекс 12)

Столбец Location_Hyderabad (тип uint8) имеет 0 пропусков (индекс 13)

Столбец Location_Jaipur (тип uint8) имеет 0 пропусков (индекс 14)

Столбец Location_Kochi (тип uint8) имеет 0 пропусков (индекс 15)

Столбец Location_Kolkata (тип uint8) имеет 0 пропусков (индекс 16)

Столбец Location_Mumbai (тип uint8) имеет 0 пропусков (индекс 17)

Столбец Location_Pune (тип uint8) имеет 0 пропусков (индекс 18)

Столбец Fuel_Type_CNG (тип uint8) имеет 0 пропусков (индекс 19)

Столбец Fuel_Type_Diesel (тип uint8) имеет 0 пропусков (индекс 20)

Столбец Fuel_Type_LPG (тип uint8) имеет 0 пропусков (индекс 21)

Столбец Fuel_Type_Petrol (тип uint8) имеет 0 пропусков (индекс 22)

Столбец Transmission_Automatic (тип uint8) имеет 0 пропусков (индекс 23)

Столбец Transmission_Manual (тип uint8) имеет 0 пропусков (индекс 24)
```

С ходу непонятно, имело ли смысл оставлять колонку "Location".

5. Масштабирование данных

Видим, что одни колонки имеют значения в единицах и десятках, а другие - в десятках тысяч.

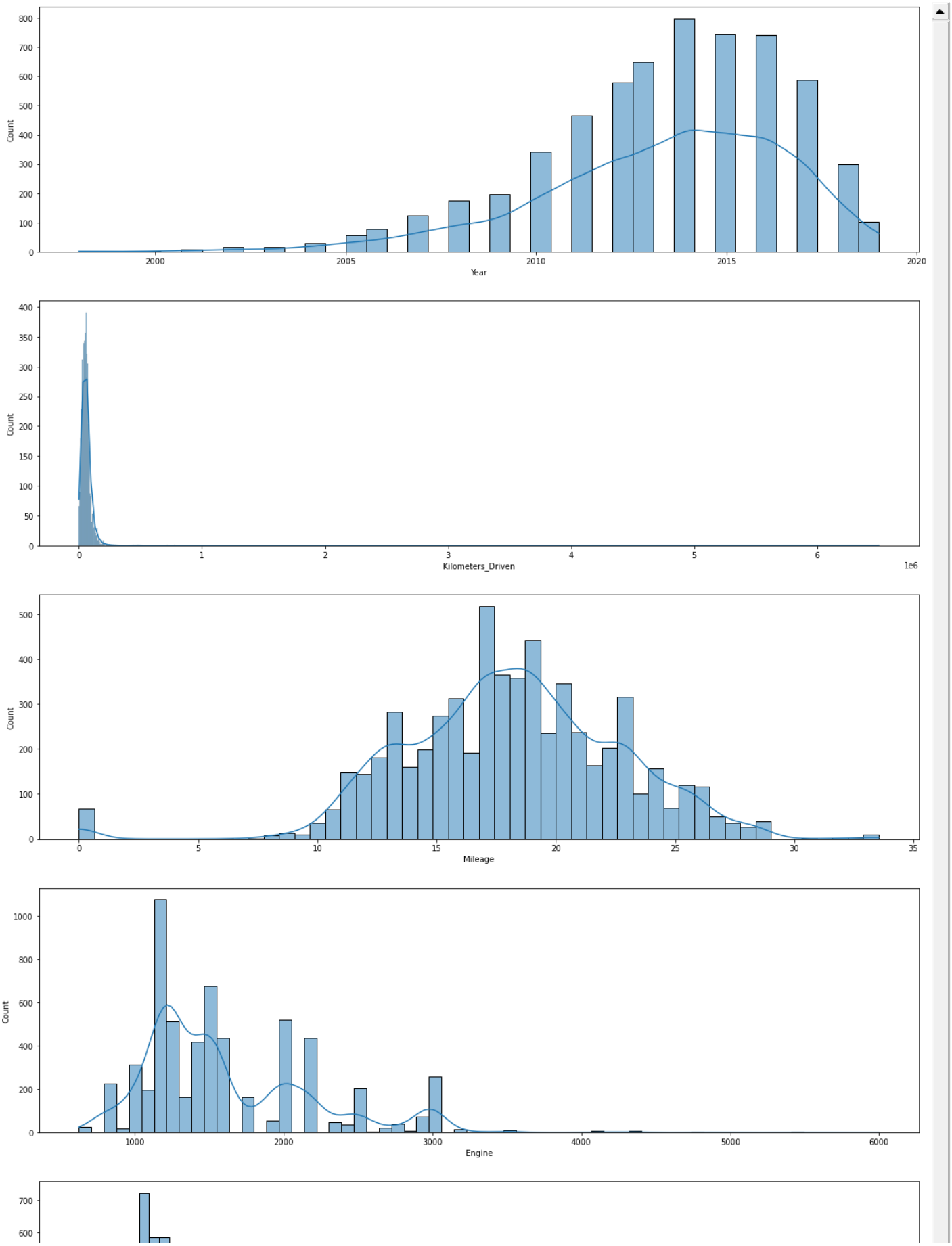
In [37]:

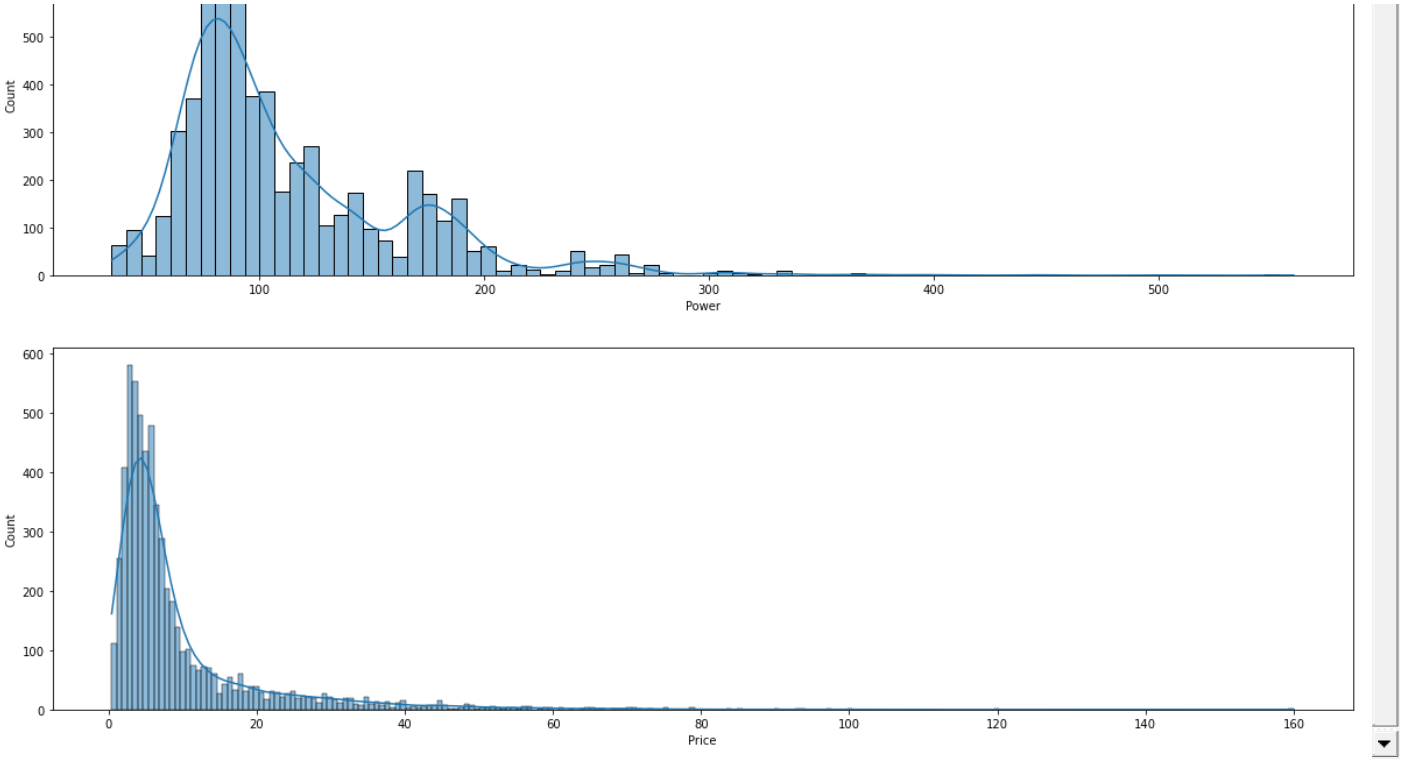
```
data.head(10)
```

```
data_unscaled = data.copy()
```

In [38]:

```
import matplotlib.pyplot as plt

columnsToScale = ["Year", "Kilometers_Driven", "Mileage", "Engine", "Power","Price"]
fig, ax = plt.subplots(len(columnsToScale), 1, figsize=(20,40))
for i in range(len(columnsToScale)):
    sns.histplot(data[columnsToScale[i]], ax=ax[i], kde = True)
```





In [39]:

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
minMaxScaler = MinMaxScaler()
zScaler = StandardScaler()

def ColumnScaler(data, columnName, scaler):
    return scaler.fit_transform(data[[columnName]])
```

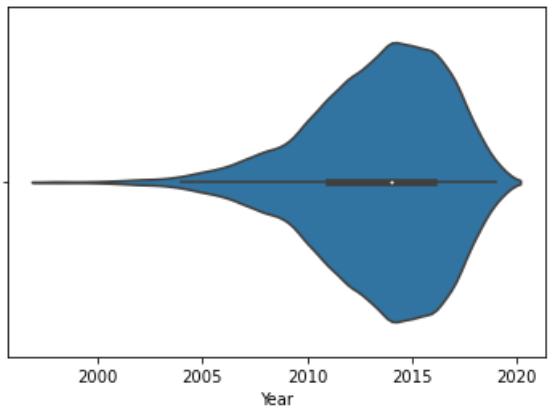
Year

In [40]:

```
sns.violinplot(data=data, x="Year")
```

Out[40]:

```
<AxesSubplot:xlabel='Year'>
```



Кривая соответствует нормальному распределению с небольшим перекосом, поэтому применим MinMaxScaler

In [41]:

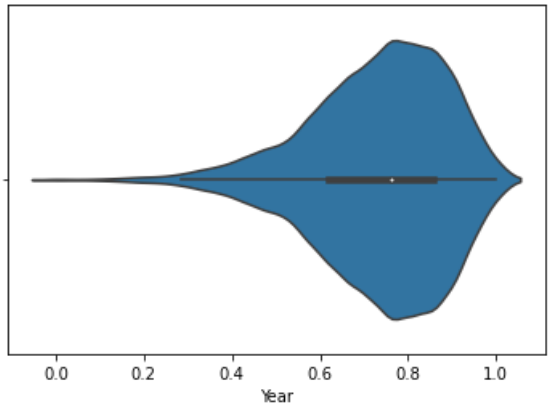
```
new_year = ColumnScaler(data, "Year", minMaxScaler)
data["Year"] = new_year
```

In [42]:

```
sns.violinplot(data=data, x="Year")
```

Out[42]:

```
<AxesSubplot:xlabel='Year'>
```



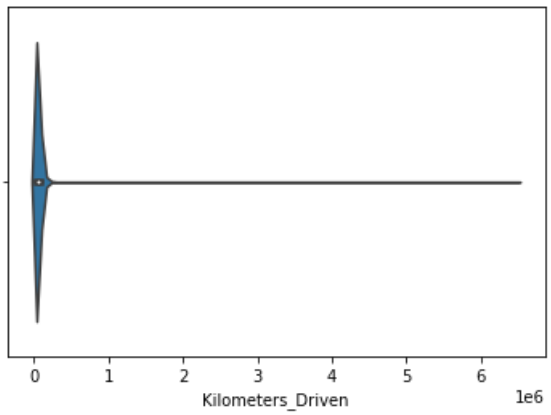
Kilometers_Driven

In [43]:

```
sns.violinplot(data=data, x="Kilometers_Driven", kde = True)
```

Out[43]:

```
<AxesSubplot:xlabel='Kilometers_Driven'>
```



Видим длиннющий хвост, поэтому применим Z-оценку

In [44]:

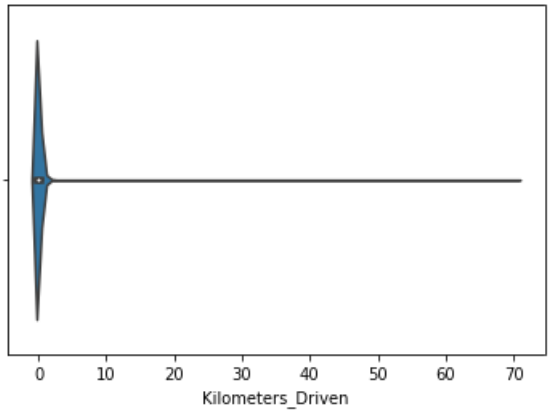
```
new_kmDriven = ColumnScaler(data, "Kilometers_Driven", zScaler)
data["Kilometers_Driven"] = new_kmDriven
```

In [45]:

```
sns.violinplot(data=data, x="Kilometers_Driven", kde = True)
```

Out[45]:

```
<AxesSubplot:xlabel='Kilometers_Driven'>
```



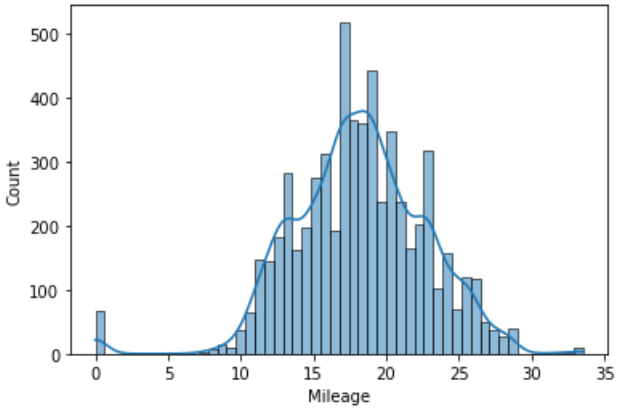
Mileage

In [46]:

```
sns.histplot(data=data, x="Mileage", kde = True)
```

Out[46]:

```
<AxesSubplot:xlabel='Mileage', ylabel='Count'>
```



Видим небольшой перекос выборки на нулевых значениях, но в целом кривая соответствует нормальному распределению, поэтому применим MinMaxScaler

In [47]:

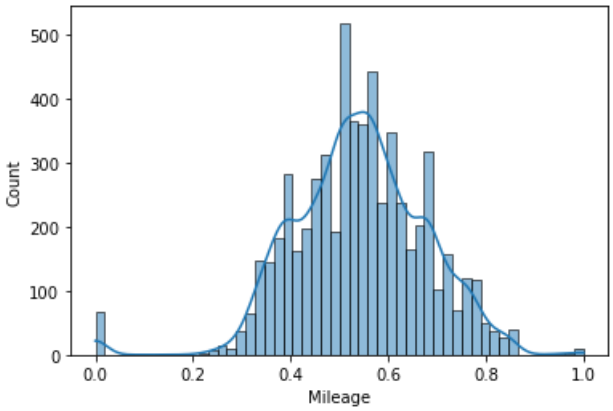
```
new_mileage = ColumnScaler(data, "Mileage", minMaxScaler)
data["Mileage"] = new_mileage
```

In [48]:

```
sns.histplot(data=data, x="Mileage", kde = True)
```

Out[48]:

```
<AxesSubplot:xlabel='Mileage', ylabel='Count'>
```



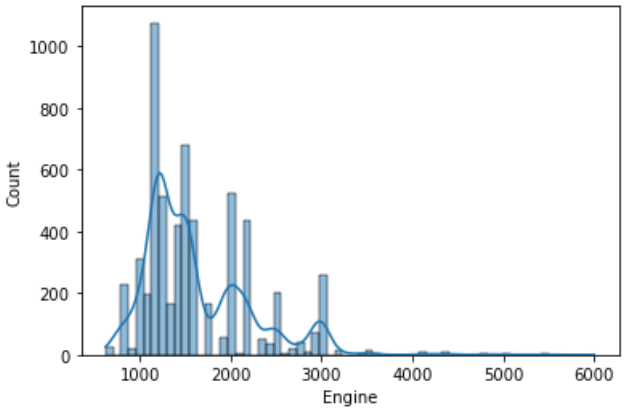
Engine

In [49]:

```
sns.histplot(data=data, x="Engine", kde = True)
```

Out[49]:

```
<AxesSubplot:xlabel='Engine', ylabel='Count'>
```



In [50]:

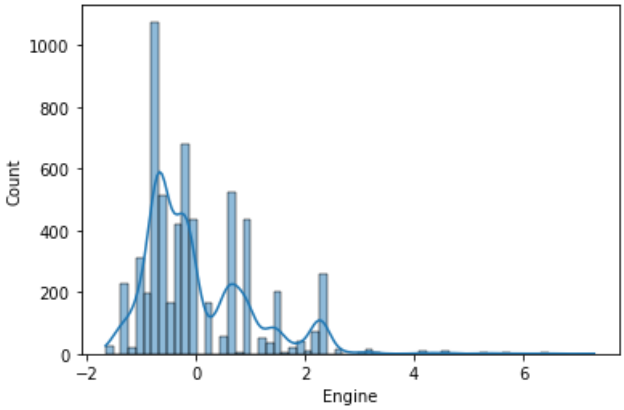
```
new_engine = ColumnScaler(data, "Engine", zScaler)
data["Engine"] = new_engine
```

In [51]:

```
sns.histplot(data=data, x="Engine", kde = True)
```

Out[51]:

```
<AxesSubplot:xlabel='Engine', ylabel='Count'>
```



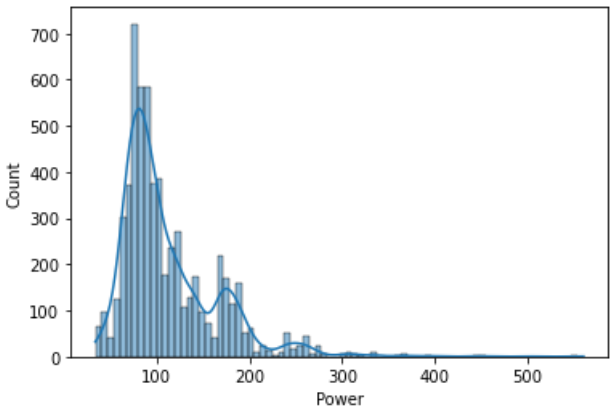
Power

In [52]:

```
sns.histplot(data=data, x="Power", kde = True)
```

Out[52]:

```
<AxesSubplot:xlabel='Power', ylabel='Count'>
```



In [53]:

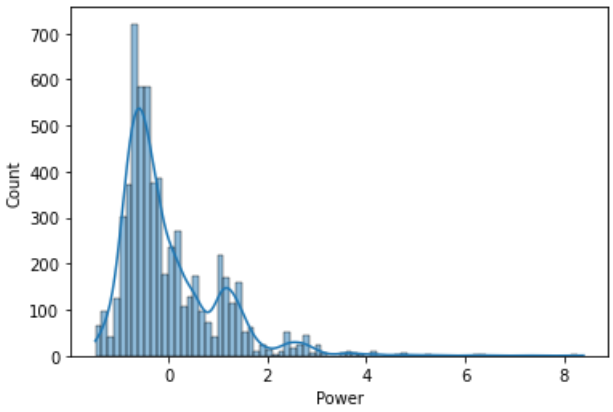
```
new_power = ColumnScaler(data, "Power", zScaler)
data["Power"] = new_power
```

In [54]:

```
sns.histplot(data=data, x="Power", kde = True)
```

Out[54]:

```
<AxesSubplot:xlabel='Power', ylabel='Count'>
```



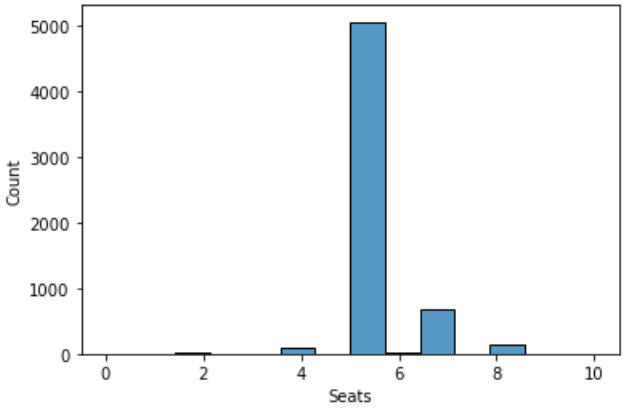
Seats

In [55]:

```
sns.histplot(data=data, x="Seats")
```

Out[55]:

```
<AxesSubplot:xlabel='Seats', ylabel='Count'>
```



In [56]:

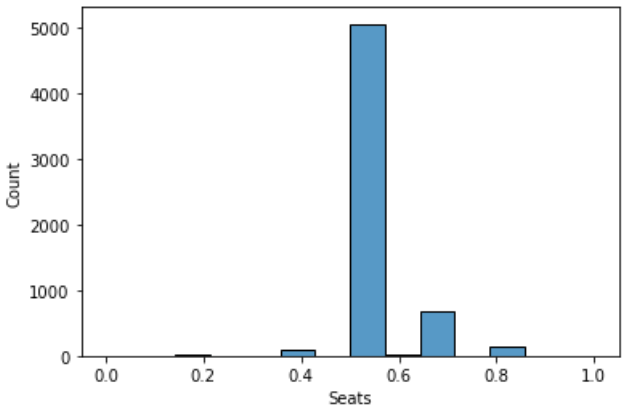
```
new_seats = ColumnScaler(data, "Seats", minMaxScaler)
data["Seats"] = new_seats
```

In [57]:

```
sns.histplot(data=data, x="Seats")
```

Out[57]:

```
<AxesSubplot:xlabel='Seats', ylabel='Count'>
```



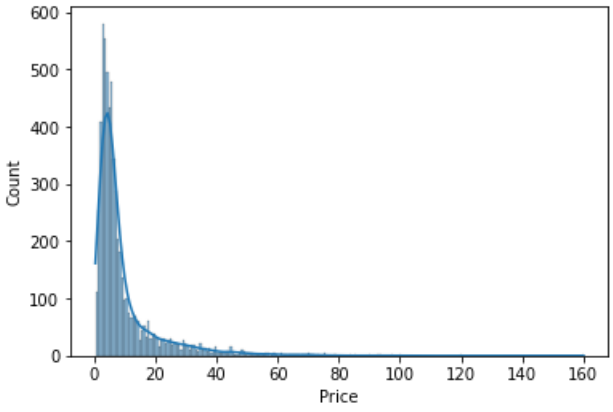
Price

In [58]:

```
sns.histplot(data=data, x="Price", kde = True)
```

Out[58]:

```
<AxesSubplot:xlabel='Price', ylabel='Count'>
```



In [59]:

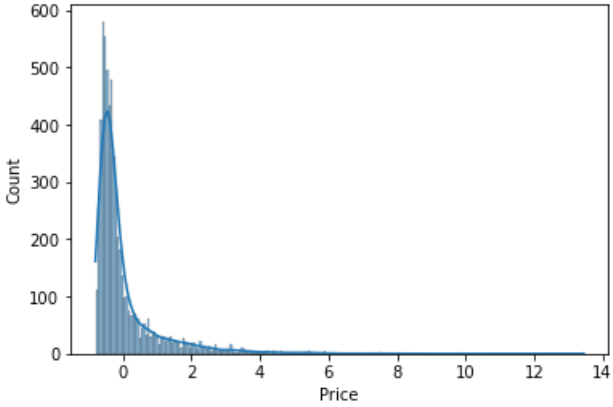
```
new_price = ColumnScaler(data, "Price", zScaler)
data["Price"] = new_price
```

In [60]:

```
sns.histplot(data=data, x="Price", kde = True)
```

Out[60]:

```
<AxesSubplot:xlabel='Price', ylabel='Count'>
```



Итоговый вид набора данных

In [61]:

```
data.head(30)
```

Out[61]:

	Year	Kilometers_Driven	Owner_Type	Mileage	Engine	Power	Seats	Price	Location_Ahmedabad	Location_Bangalore	...	Location_
0	0.571429	0.145248	1	0.793083	-1.039023	-1.027489	0.5	-0.690729	0	0	...	
1	0.809524	-0.194380	1	0.586464	-0.064632	0.249596	0.5	0.270064	0	0	...	
2	0.619048	-0.139601	1	0.542636	-0.703659	-0.454264	0.5	-0.444945	0	0	...	
3	0.666667	0.309585	1	0.619261	-0.621904	-0.453138	0.7	-0.310880	0	0	...	
4	0.714286	-0.197996	2	0.453190	0.579401	0.523633	0.5	0.738395	0	0	...	
5	0.666667	0.178116	1	0.629100	-1.346023	-1.083047	0.5	-0.637103	0	0	...	
6	0.714286	0.309574	1	0.688134	-0.266518	-0.934767	0.5	-0.534321	0	0	...	
7	0.857143	-0.249159	1	0.338700	1.892493	1.099860	0.8	0.716945	0	0	...	
8	0.714286	0.062313	1	0.612403	-0.037936	-0.174597	0.5	-0.382381	0	0	...	
9	0.666667	0.078769	2	0.664878	-0.621904	-0.730178	0.5	-0.672854	0	0	...	
10	0.952381	-0.362091	1	0.642815	-0.264849	-0.181167	0.5	0.042155	0	0	...	
11	0.666667	0.013779	1	0.500894	-0.206453	0.063777	0.5	-0.445838	0	0	...	
12	0.809524	0.062248	1	0.751342	-0.621904	-0.730178	0.5	-0.346631	0	0	...	
13	0.761905	0.145248	1	0.378652	0.931450	1.403928	0.5	1.566017	0	0	...	
14	0.666667	0.287673	2	0.000000	0.931450	0.039377	0.5	0.716945	0	0	...	
15	0.761905	0.561567	1	0.402504	1.428656	1.176065	0.7	0.493504	0	0	...	
16	0.857143	0.002276	1	0.769231	-0.204784	-0.268445	0.5	-0.364506	0	0	...	
17	0.904762	-0.369672	1	0.846750	-0.621904	-0.730178	0.5	-0.311774	0	0	...	
18	0.761905	0.205165	1	0.609720	-0.266518	-0.546236	0.5	-0.280493	0	0	...	
19	0.761905	0.216461	1	0.442457	0.871384	1.027034	0.5	1.655394	0	1	...	
20	0.761905	-0.282224	1	0.676506	0.624450	1.447098	0.5	0.810789	0	0	...	
21	0.809524	-0.036705	2	0.705128	-0.621904	-0.458018	0.5	-0.109784	0	1	...	
22	0.809524	-0.030208	1	0.403399	0.606096	1.203281	0.5	1.253201	0	0	...	
23	0.571429	-0.141716	1	0.551580	-0.706996	-0.617560	0.5	-0.680004	0	0	...	
24	0.571429	-0.282026	1	0.429338	-0.037936	-0.174597	0.5	-0.592415	0	0	...	
25	0.666667	-0.074743	1	0.500894	-0.206453	0.063777	0.5	-0.467289	0	0	...	
26	0.666667	-0.051955	1	0.688134	-0.266518	-0.934767	0.5	-0.467289	0	0	...	
27	0.714286	-0.051955	2	0.623733	-1.039023	-0.859688	0.5	-0.601353	0	0	...	
28	0.952381	-0.233492	1	0.521765	-0.703659	-0.454264	0.5	0.037686	0	0	...	
29	0.428571	2.226843	4	0.381634	1.457020	-0.204629	0.7	-0.489633	0	0	...	

30 rows × 25 columns

In [62]:

```
data_scaled = data.copy()
```

Лабораторная Работа №3

"Подготовка обучающей и тестовой выборок. Кросс-валидация. Подбор гиперпараметров для метода kNN."

Что изменилось во второй работе:

1. Удалил две строки для электрических машин, как и планировал;
2. На момент демонстрации я не удалил столбец ID и не нормализовал его. Я про это благополучно забыл, из-за чего на масштабированных данных оптимальным числом соседей становились числа от 110 до 160, а к-т детерминации (R2) был меньше нуля. Посмотрим, как поведёт себя исправленная модель;

Поработаем над масштабированным и немасштабированным наборами данных из предыдущей работы.

In [63]:

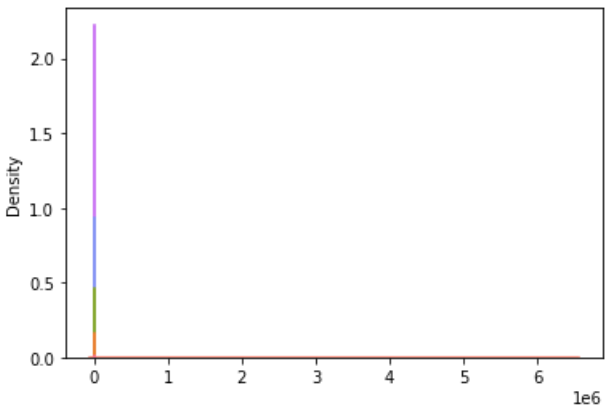
```
# Функция вернёт обновлённый список колонок
def MoveColumnToEnd(data, column_name):
    columns = data.columns.tolist()
    column = columns.pop(columns.index(column_name))
    columns.append(column)
    return columns

# Переместим целевой признак в конец
new_columns = MoveColumnToEnd(data_unscaled, "Price")

# Поскольку колонки у обоих наборов одинаковые, результат будет общим.
data_unscaled = data_unscaled[new_columns]
data_scaled = data_scaled[new_columns]
```

In [64]:

```
sns.kdeplot(data = data_unscaled, legend = False)
```

Out[64]:

```
<AxesSubplot:ylabel='Density'>
```



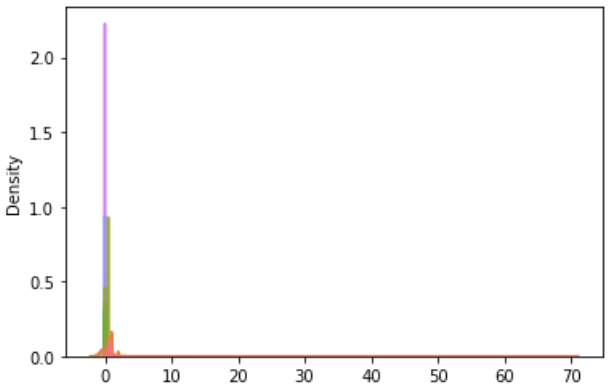
до исправления ошибки в виде столбца ID на масштабированных д-х был идентичный график, разве что вместо e+6 было e+3

In [65]:

```
sns.kdeplot(data = data_scaled, legend = False)
```

Out[65]:

```
<AxesSubplot:ylabel='Density'>
```



1. Разделение набора данных

In [66]:

```
from sklearn.model_selection import train_test_split


# Для определённости будем всегда подставлять в random_state одно значение
RANDOM_STATE_GLOBAL = 16
# Указываем, где целевой признак, а где - набор данных
y_column = "Price"
x_columns = data_unscaled.columns.tolist()
x_columns.pop(x_columns.index(y_column))


data_unscaled_x_train, data_unscaled_x_test, data_unscaled_y_train, data_unscaled_y_test = train_test_split(da
                                                                                                             da
                                                                                                             te
                                                                                                             ra


data_scaled_x_train, data_scaled_x_test, data_scaled_y_train, data_scaled_y_test = train_test_split(data_scale
                                                                                                             da
                                                                                                             te
                                                                                                             ra
```

2. Получение произвольной модели

In [67]:

```
from sklearn.neighbors import KNeighborsRegressor


random_param = 22

knn_unscaled = KNeighborsRegressor(n_neighbors = random_param)
knn_scaled = KNeighborsRegressor(n_neighbors = random_param)

knn_unscaled.fit(data_unscaled_x_train, data_unscaled_y_train)
knn_scaled.fit(data_scaled_x_train, data_scaled_y_train)

knn_unscaled_prediction = knn_unscaled.predict(data_unscaled_x_test)
knn_scaled_prediction = knn_scaled.predict(data_scaled_x_test)
```

Получили модель, теперь - оценка качества:

In [68]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, median_absolute_error, r2_score
from sklearn.model_selection import ShuffleSplit, cross_val_score, cross_validate


def PrintRegressionMetrics(y_test, y_predicted):
    return "-Средняя абсолютная ошибка = {0};\
          \n-Медианная абсолютная ошибка = {1};\
          \n-Среднеквадратичная ошибка = {2};\
          \n-Коэффициент детерминации = {3}.".format(mean_absolute_error(y_test, y_predicted),
                                                     median_absolute_error(y_test, y_predicted),
                                                     mean_squared_error(y_test, y_predicted, squared = Fals
                                                     r2_score(y_test, y_predicted))
```

In [69]:

```
random_model_unscaled_data_results = PrintRegressionMetrics(data_unscaled_y_test, knn_unscaled_prediction)
print("Для немасштабированных данных:\n" + random_model_unscaled_data_results)
```

```
Для немасштабированных данных:
-Средняя абсолютная ошибка = 5.4108360086112475;
-Медианная абсолютная ошибка = 3.1079545454545463;
-Среднеквадратичная ошибка = 9.720584012504426;
-Коэффициент детерминации = 0.24164131017945278.
```

In [70]:

```
random_model_scaled_data_results = PrintRegressionMetrics(data_scaled_y_test, knn_scaled_prediction)
print("Для масштабированных данных:\n" + random_model_scaled_data_results)
```

```
Для масштабированных данных:
-Средняя абсолютная ошибка = 0.2788384178937498;
-Медианная абсолютная ошибка = 0.13642040783353254;
-Среднеквадратичная ошибка = 0.5245404461146183;
-Коэффициент детерминации = 0.7235572218632615.
```

Проведём кросс-валидацию и посмотрим, насколько одно случайное разбиение соответствует среднему показателю:

In [71]:

```
def PrintDictionary(dict):
    for key, value in dict.items():
        print("\n{0} -> {1}".format(key, value))


scoring_strategies = ["neg_root_mean_squared_error", "r2"]

# Кросс-валидация по стратегии ShuffleSplit
data_unscaled_cv_scores = cross_validate(KNeighborsRegressor(n_neighbors = random_param), data_unscaled[x_colu
data_scaled_cv_scores = cross_validate(KNeighborsRegressor(n_neighbors = random_param), data_scaled[x_columns]

print("Кросс-валидация для немасштабированных данных:")
PrintDictionary(data_unscaled_cv_scores)
print("\n\nКросс-валидация для масштабированных данных:")
PrintDictionary(data_scaled_cv_scores)
```

```
Кросс-валидация для немасштабированных данных:

fit_time -> [0.01599717 0.01499629 0.01499581 0.01399708 0.01399803 0.01399708
 0.01499724 0.01499701]

score_time -> [0.06098843 0.06198907 0.05798936 0.05798936 0.05898762 0.0579896
 0.0579896  0.05798841]

test_neg_root_mean_squared_error -> [ -9.72058401 -10.03781859  -9.63748914  -9.66133394  -9.85531
  -9.49133028  -9.55642826  -9.84120328]

test_r2 -> [0.24164131 0.23014713 0.25633304 0.25878841 0.22987474 0.27061136
 0.26106281 0.24382612]


Кросс-валидация для масштабированных данных:

fit_time -> [0.01399684 0.01499701 0.01499629 0.01499605 0.01399732 0.01399732
 0.01399708 0.01399732]

score_time -> [0.29394341 0.32693887 0.31893992 0.2979424  0.29394341 0.27794695
 0.27794719 0.28294587]

test_neg_root_mean_squared_error -> [-0.52454045 -0.5679027  -0.54367736 -0.54377808 -0.53673244 -0.53171855
 -0.54593    -0.55029173]

test_r2 -> [0.72355722 0.69151486 0.70372806 0.7060532  0.71404756 0.71343326
 0.6981108  0.70401577]
```

Масштабированные данные имеют отрицательный показатель к-та детерминации, что говорит об их ужасной обобщающей способности. При этом среднеквадратичная ошибка низка, что может свидетельствовать либо о переобучении, либо о неспособности модели обобщить такие данные в принципе.

Что есть высокая и низкая ошибки? Смотрим на графики распределения целевого признака из второй лабораторной: на немасштабированных данных значения распределны примерно от 0 до 160, то есть ошибка равняется где-то 6% длины интервала, а на масштабированных - значения примерно от -1 до 13.5, то есть ошибка - где-то 7 или 8 процентов.

Также стоит заметить, что немасштабированные данные имеют существенно больший коэффициент детерминации.

3. Получение оптимальной модели

Немасштабированные данные

Теперь найдём оптимальное значение K, используя решётчатый поиск.

In [72]:

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, KFold

# Перебирал так много из-за проблем с подбором на масштабированных данных
tested_parametres = {"n_neighbors" : np.array(range(1, 41, 3))}
```

In [73]:

```
%%time
# Кросс-валидация по стратегии KFold
randomized_grid_search = RandomizedSearchCV(KNeighborsRegressor(),
                                            tested_parametres,
                                            n_iter = 7,
                                            random_state = RANDOM_STATE_GLOBAL,
                                            cv = KFold(shuffle = True, random_state = RANDOM_STATE_GLOBAL),
                                            scoring = "neg_root_mean_squared_error")
```

```
randomized_grid_search.fit(data_unscaled[x_columns], data_unscaled[y_column])
```

```
Wall time: 2.79 s
```

Out[73]:

```
RandomizedSearchCV(cv=KFold(n_splits=5, random_state=16, shuffle=True),
                   estimator=KNeighborsRegressor(), n_iter=7,
                   param_distributions={'n_neighbors': array([ 1,  4,  7, 10, 13, 16, 19, 22, 25, 28, 31, 34,
37, 40])},
                   random_state=16, scoring='neg_root_mean_squared_error')
```

In [74]:

```
# Найденное значение:
randomized_best_param = randomized_grid_search.best_params_.get("n_neighbors")
print(randomized_best_param, randomized_grid_search.best_score_)
```

```
7 -7.446886385306317
```

Попробуем найти оптимальное решение вокруг найденного наилучшего параметра:

In [75]:

```
# Ищем решение вокруг полученного значения с радиусом 10
# Тоже перебираю так много из-за проблем с масштабированными данными
gs_parametres = {"n_neighbors" : np.array(range(randomized_best_param - 3, randomized_best_param + 4))}

# Прогоняем все решения в окрестности
grid_search = GridSearchCV(KNeighborsRegressor(), gs_parametres, scoring = "neg_root_mean_squared_error", cv =
# Обучаем оптимальную модель
grid_search.fit(data_unscaled[x_columns], data_unscaled[y_column])
```

Out[75]:

```
GridSearchCV(cv=KFold(n_splits=5, random_state=16, shuffle=True),
             estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': array([ 4,  5,  6,  7,  8,  9, 10])},
             scoring='neg_root_mean_squared_error')
```

In [76]:

```
# Итоговые "наилучшие" параметр и показатель RMSE:
best_param = grid_search.best_params_.get("n_neighbors")
print(best_param, grid_search.best_score_)
```

```
5 -7.396568464228198
```

Обучим оптимальную модель:

In [77]:

```
cv_found_knn_unscaled = KNeighborsRegressor(n_neighbors = best_param)
cv_found_knn_unscaled.fit(data_unscaled_x_train, data_unscaled_y_train)
cv_found_knn_unscaled_prediction = cv_found_knn_unscaled.predict(data_unscaled_x_test)

found_model_unscaled_data_results = PrintRegressionMetrics(data_unscaled_y_test, cv_found_knn_unscaled_predict
```

Сравним результаты:

In [78]:

```
print("Случайная модель:\n-K = {0};\n{1}\n\nОптимальная модель:\n-K = {2};\n{3}".format(random_param, random_r
```

```
Случайная модель:
-K = 22;
-Средняя абсолютная ошибка = 5.4108360086112475;
-Медианная абсолютная ошибка = 3.1079545454545463;
-Среднеквадратичная ошибка = 9.720584012504426;
-Коэффициент детерминации = 0.24164131017945278.

Оптимальная модель:
-K = 5;
-Средняя абсолютная ошибка = 4.754752804320732;
-Медианная абсолютная ошибка = 2.302;
-Среднеквадратичная ошибка = 8.684876941069014;
-Коэффициент детерминации = 0.3946350255561992.
```

Видим существенный выигрыш от решётчатого поиска: все параметры существенно "прибавили", в особенности - R2.

Масштабированные данные

Проделаем то же самое:

In [89]:

```
# Точно также прогоняем: тот же диапазон возможных значений K, тот же ключ генератора случ. чисел,
#                        та же стратегия кросс-валидации, та же метрика.
randomized_grid_search = RandomizedSearchCV(KNeighborsRegressor(),
                                            tested_parametres,
                                            n_iter = 7,
                                            random_state = RANDOM_STATE_GLOBAL,
```

```
                                                cv = KFold(shuffle = True, random_state = RANDOM_STATE_GLOBAL),
                                                scoring = "neg_root_mean_squared_error")
# Теперь работаем с масштабированными данными
randomized_grid_search.fit(data_scaled[x_columns], data_scaled[y_column])

# Найденное рабочее значение
randomized_best_param = randomized_grid_search.best_params_.get("n_neighbors")
print(randomized_best_param, randomized_grid_search.best_score_)

# Ищем оптимальное
gs_parametres = {"n_neighbors" : np.array(range(randomized_best_param - 3, randomized_best_param + 4))}
grid_search = GridSearchCV(KNeighborsRegressor(), gs_parametres, scoring = "neg_root_mean_squared_error", cv =
grid_search.fit(data_scaled[x_columns], data_scaled[y_column])

# Найденное оптимальное значение
best_param = grid_search.best_params_.get("n_neighbors")
print(best_param, grid_search.best_score_)

# Обучаем оптимальную модель
cv_found_knn_scaled = KNeighborsRegressor(n_neighbors = best_param)
cv_found_knn_scaled.fit(data_scaled_x_train, data_scaled_y_train)
cv_found_knn_scaled_prediction = cv_found_knn_scaled.predict(data_scaled_x_test)

# Смотрим на результат
found_model_scaled_data_results = PrintRegressionMetrics(data_scaled_y_test, cv_found_knn_scaled_prediction)
```

```
7 -0.5014027799466024
6 -0.49959869739819834
```

In [90]:

```
print("Случайная модель:\n-K = {0};\n{1}\n\nОптимальная модель:\n-K = {2};\n{3}".format(random_param, random_r
```

```
Случайная модель:
-K = 22;
-Средняя абсолютная ошибка = 0.2788384178937498;
-Медианная абсолютная ошибка = 0.13642040783353254;
-Среднеквадратичная ошибка = 0.5245404461146183;
-Коэффициент детерминации = 0.7235572218632615.

Оптимальная модель:
-K = 6;
-Средняя абсолютная ошибка = 0.2854839407659935;
-Медианная абсолютная ошибка = 0.12423275972452132;
-Среднеквадратичная ошибка = 0.561599142443407;
-Коэффициент детерминации = 0.6831161118747986.
```

Здесь получили ухудшение от решётчатого поиска. Явление объяснить тяжело; возможно, слишком хорошо попали изначальной точкой.

Так или иначе, преимущество обучения на масштабированных данных крайне заметно!