Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

«Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных» по курсу «Технологии машинного обучения» Лабораторная работа №3

Выполнил:
студент группы ИУ5 – 62Б
Карягин А.Д.
подпись, дата

Проверил:
преподаватель кафедры ИУ5
Гапанюк Ю.Е.
подпись, дата

2020 г.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="whitegrid")
```

Загрузка и первичный анализ набора данных¶

Для обработки пропусков в числовых данных будем использовать набор данных об автомобилях. Набор данных состоит из спецификации автомобилей с описанием различных технических параметров, а также двух специальных показателей:

"Символизирование" (symboling) - оценка страхового риска. Показывает степень, с которой
автомобиль более "рискованный"("опасный"), чем показывает его цена. Автомобилю изначально назначают символ фактора риска, связанный с его ценой. Далее, если он более (или менее) рискованный, то символ смещается вверх (или вниз) по шкале. Актуарии (специалисты по страховой математике) называют этот процесс "символизированием". Оценка "+3" означает, что авто "рискованное", "-3" - что оно достаточно безопасное.

Нормализованные потери (normalized-losses) - относительная средняя сумма возмещения убытков за год застрахованного автомобиля. Этот показатель нормализуется для всех автомобилей внутри определенной классификации по размеру ("двухдверные маленькие" (two-door small), "универсалы" (station wagons), "спортивные/особенные" (sports/speciality), и т.д.) и определеяет средние потери на автомобиль в год.

Колонки:

symboling - символизирование {-3, -2, -1, 0, 1, 2, 3}

normalized-losses - нормализованные потери (от 65 до 256)

make - марка {alfa-romero, audi, bmw, chevrolet, dodge, honda, ... renault, saab, subaru, toyota, volkswagen, volvo}

fuel-type - тип топлива {diesel, gas}

aspiration - наддув {std, turbo}

num-of-doors - кол-во дверей {four, two}

body-style - тип кузова {hardtop, wagon, sedan, hatchback, convertible}

drive-wheels - привод {4wd, fwd, rwd}

engine-location - расположение двигателя {front, rear}

wheel-base - размер колесной базы (от 86.6 до 120.9)

length - длина авто (от 141.1 до 208.1)

width - ширина авто (от 60.3 до 72.3)

height - высота авто (от 47.8 до 59.8) curb-weight -

снаряжённая масса (от 1488 до 4066) engine-type - тип

двигателя {dohc, dohcv, l, ohc, ohcf, ohcv, rotor}

num-of-cylinders - кол-во цилинлров {eight, five, four, six, three, twelve, two}

engine-size - размер двигателя (от 61 до 326)

fuel-system - топливная система {1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi}

bore - диаметр цилиндра (от 2.54 до 3.94) stroke - ход поршня (от 2.07

до 4.17)

compression-ratio - степень сжатия (от 7 до 23)

horsepower - кол-во лошадиных сил (48 до 288)

peak-rpm - макс. число оборотов в минуту (4150 до 6600)

city-mpg - расход топлива в городе (от 13 до 49)

highway-mpg - расход топлива на шоссе (от 16 до 54)

price - цена (от 5118 до 45400)

In [51]:

```
headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-do
ors","body-style",

         "drive-wheels","engine-location","wheel-base", "length","width","height","cur
b-weight","engine-type",

         "num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-
ratio","horsepower",

         "peak-rpm","city-mpg","highway-mpg","price"]

data = pd.read_csv('data/imports-85.data', names = headers)

data.head()
```

Out[51]:

	symboling	normalized-losses	
	03	?	alfa-romero
	13	?	alfa-romero
	21	?	alfa-romero
	32	164	audi
	42	164	audi

5 rows × 26 columns

In [4]:

```
  # Размер набора данных (строки, колонки)
data.shape
```

Out[4]:

```
(205, 26)
```

In [5]:

```
  # Типы данных в колонках
data.dtypes
```

Out[5]:

```
symboling
normalized-losses
make
fuel-type
aspiration
num-of-doors
body-style
drive-wheels
engine-location
wheel-base
length
width
height
curb-weight
engine-type
num-of-cylinders
engine-size
fuel-system
bore
stroke
```

```
compression-ratio
horsepower
peak-rpm
city-mpg
highway-mpg
price
dtype: object
```

Пропущенные значения¶

Вывод первых пяти строк показал, что пропущенные значения обозначены в наборе данных ?. Метод .isnull().sum() работать не будет, как и другие методы, воспринимающие пропущенные значения в виде NaN.

In [52]:

```
  # Проверка пропущенных значений без преобразования "?" в NaN
data.isnull().sum()
```

Out[52]:

```
symboling
normalized-losses
make
fuel-type
aspiration
num-of-doors
body-style
drive-wheels
engine-location
wheel-base
length
width
height
curb-weight
engine-type
num-of-cylinders
engine-size
fuel-system
bore
stroke
compression-ratio
horsepower
peak-rpm
city-mpg
highway-mpg
price
dtype: int64
```

Преобразуем "?" в NaN и выведем первые пять строк набора данных.

In [53]:

```
data.replace("?", np.NaN, inplace = True)
data.head()
```

Out[53]:

	symboling	normalized-losses	
	03	NaN	alfa-rom ero
	13	NaN	alfa-rom ero
			alfa-

	21	NaN

rom

ero

	symboling	normalized-losses	
	32	164	aud
	42	164	aud

5 rows × 26 columns

Кол-во пропущенных значений¶

In [54]:

```
data.isnull().sum()
```

Out[54]:

```
symboling
normalized-losses
make
fuel-type
aspiration
num-of-doors
body-style
drive-wheels
engine-location
wheel-base
length
width
height
curb-weight
engine-type
num-of-cylinders
engine-size
fuel-system
bore
stroke
compression-ratio
horsepower
peak-rpm
city-mpg
highway-mpg
price
dtype: int64
```

1. Обработка пропусков в данных¶

1.1. Простые стратегии - удаление или заполнение нулями¶

In [55]:

```
# Удаление колонок, содержащих пустые значения
```

```
data_new_1 = data.dropna(axis=1, how='any')
(data.shape, data_new_1.shape)
```

Out[55]:

```
((205, 26), (205, 19))
```

In [56]:

```
  # Удаление строк, содержащих пустые значения
data_new_2 = data.dropna(axis=0, how='any')
(data.shape, data_new_2.shape)
```

Out[56]:

```
((205, 26), (159, 26))
```

In [57]:

```
  # Заполнение нулями
data_new_3 = data.fillna(0)
data_new_3.head()
```

Out[57]:

	symboling	normalized-losses	
	03	0	alfa-romero
	13	0	alfa-romero
	21	0	alfa-romero
	32	164	audi
	42	164	audi

5 rows × 26 columns

1.2. "Внедрение значений" - импьютация (imputation)¶

1.2.1. Обработка пропусков в числовых данных¶

Преобразование типов данных в колонках¶

Перед обработкой требуется преобразовать типы соответствующих колонок в числовые. Скорее всего эти колонки изначально загрузились как object из-за использования строкового символа ? в качестве пропуска в данных.

Число дверей num-of-doors в данном наборе данных записывается в виде строковых данных {two, four}. Тип этой колонки по-хорошему требуется преобразовать в строковый, однако для упрощения работы я преобразую уже содержащиеся в этой колонке данные в числовые (NaN воспринимается как тип float) и преобразую эту колонку вместе с остальными в цикле.

In [58]:

```
  # Преобразование кол-ва дверей в число
row2_index = data[data['num-of-doors'] == 'two'].index.tolist()
row4_index = data[data['num-of-doors'] == 'four'].index.tolist()
for row_index in data.index.values:
    if row_index in row2_index:
        data.at[row_index, 'num-of-doors'] = 2
    if row_index in row4_index:
        data.at[row_index, 'num-of-doors'] = 4
data['num-of-doors']
```

Out[58]:

```
       1   2
       2   2
       3   2
       4   4
       5   4
      ..
       200 4
       201 4
       202 4
       203 4
       204 4
Name: num-of-doors, Length: 205, dtype: object
```

In [59]:

```
  # Преобразование типа колонок с пропущенными значениями в числовой
for col in data.columns:
    if data[col].isnull().sum() > 0:
        data[col] = data[col].apply(pd.to_numeric)
data.dtypes
```

Out[59]:

```
symboling
normalized-losses
make
fuel-type
aspiration
num-of-doors
body-style
drive-wheels
engine-location
wheel-base
length
width
height
curb-weight
engine-type
num-of-cylinders
engine-size
fuel-system
bore
stroke
compression-ratio
horsepower
peak-rpm
city-mpg
highway-mpg
price
dtype: object
```

Статистика по колонкам с пропущенными значениями¶

In [60]:

```
  # Выберем числовые колонки с пропущенными значениями
  # Цикл по колонкам датасета
total_count = data.shape[0]
num_cols = []
for col in data.columns:
      #    Количество пустых значений
    temp_null_count = data[data[col].isnull()].shape[0]
    dt = str(data[col].dtype)
    if temp_null_count>0 and (dt=='float64' or dt=='int64'):
        num_cols.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(
col, dt, temp_null_count, temp_perc))
```

```
Колонка normalized-losses. Тип данных float64. Количество пустых значений 41, 20.0%.
Колонка num-of-doors. Тип данных float64. Количество пустых значений 2, 0.98%.
Колонка bore. Тип данных float64. Количество пустых значений 4, 1.95%.
Колонка stroke. Тип данных float64. Количество пустых значений 4, 1.95%.
Колонка horsepower. Тип данных float64. Количество пустых значений 2, 0.98%.
Колонка peak-rpm. Тип данных float64. Количество пустых значений 2, 0.98%.
Колонка price. Тип данных float64. Количество пустых значений 4, 1.95%.
```

Применение способов импьютации, описанных в лекции¶

In [61]:

```
  # Фильтр по колонкам с пропущенными значениями
data_num = data[num_cols]
data_num
```

Out[61]:

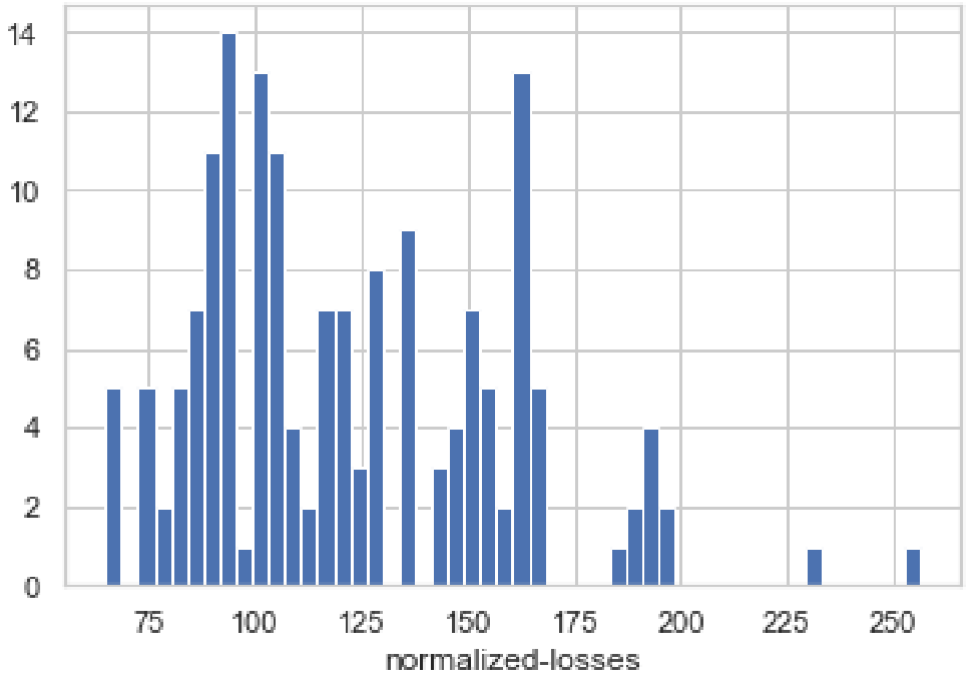
		normalized-lossesnum-of-		
0		NaN		
1		NaN		
2		NaN		
3		164.0		
4		164.0		
...		...		
200	95.0			
201	95.0			
202	95.0			
203	95.0			
204	95.0			

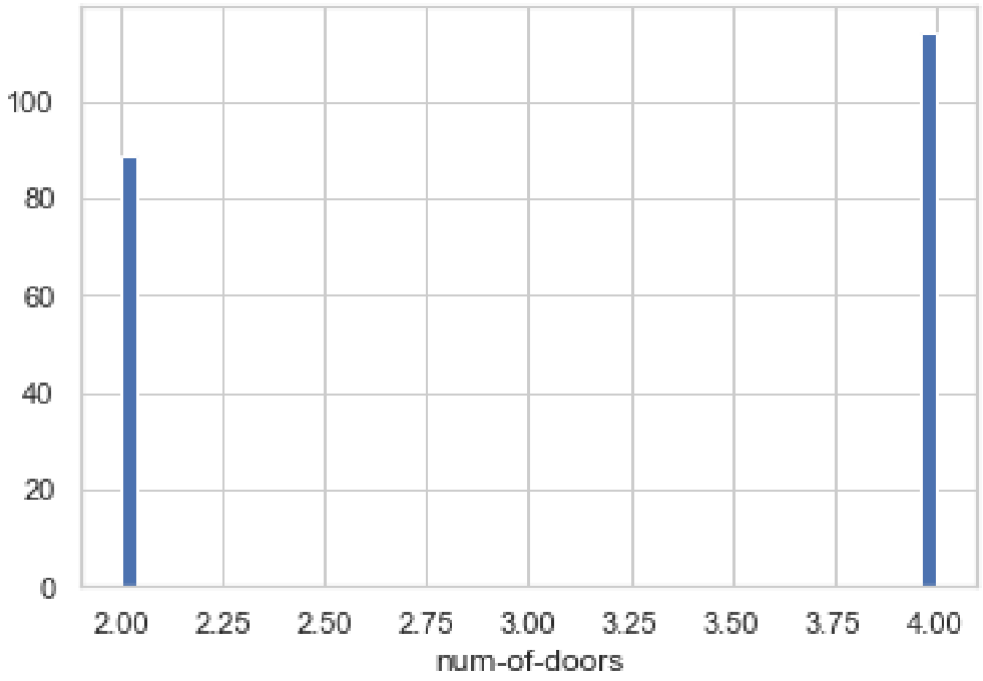
205 rows × 7 columns

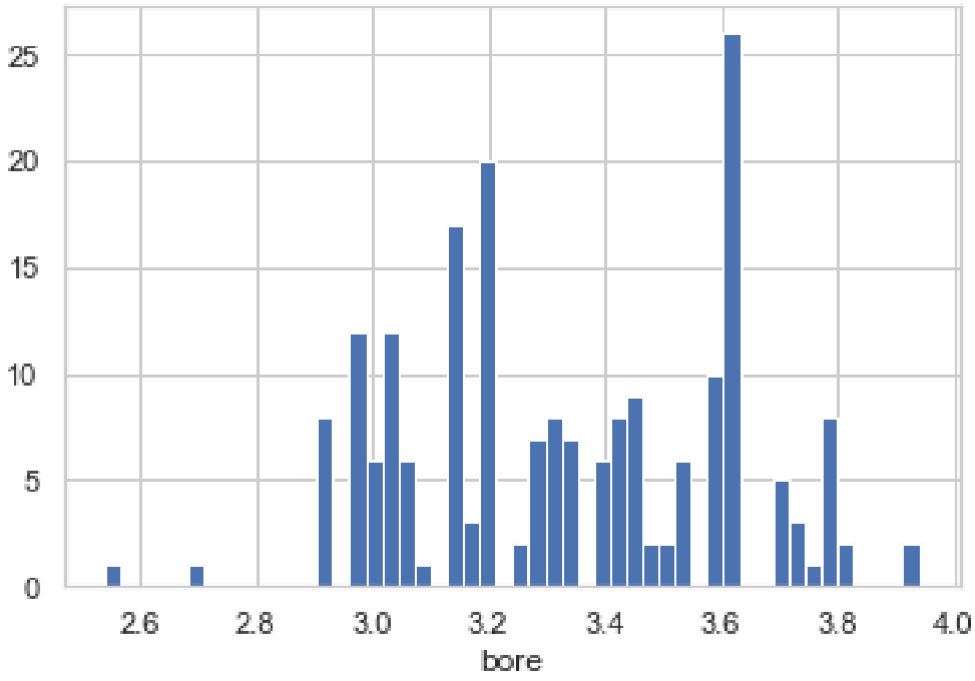
In [62]:

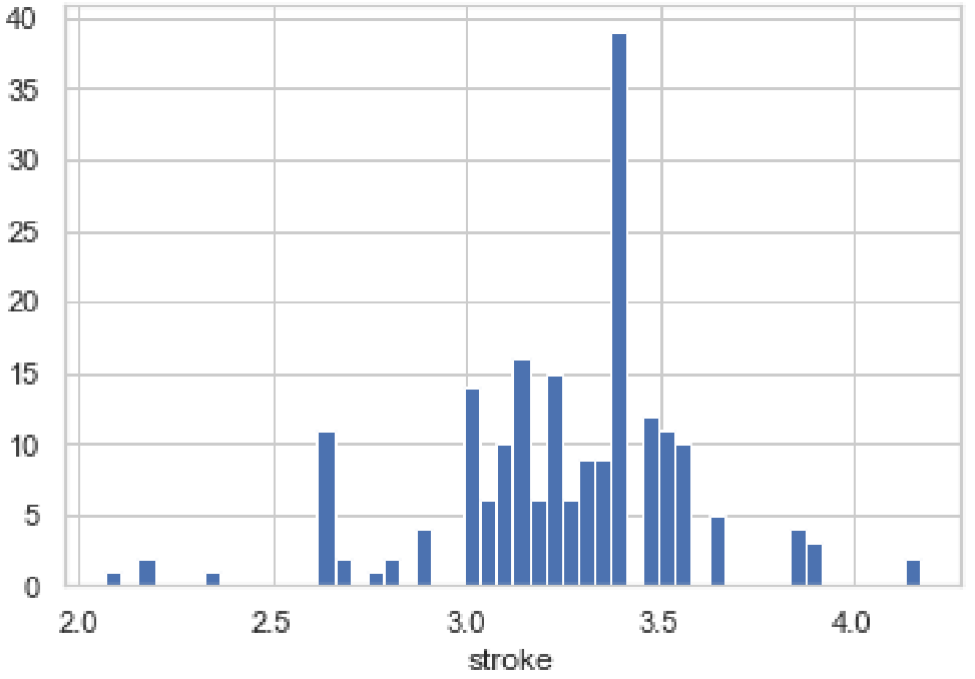
```
  # Гистограмма по признакам
for col in data_num:
    plt.hist(data[col], 50)
    plt.xlabel(col)
    plt.show()
```

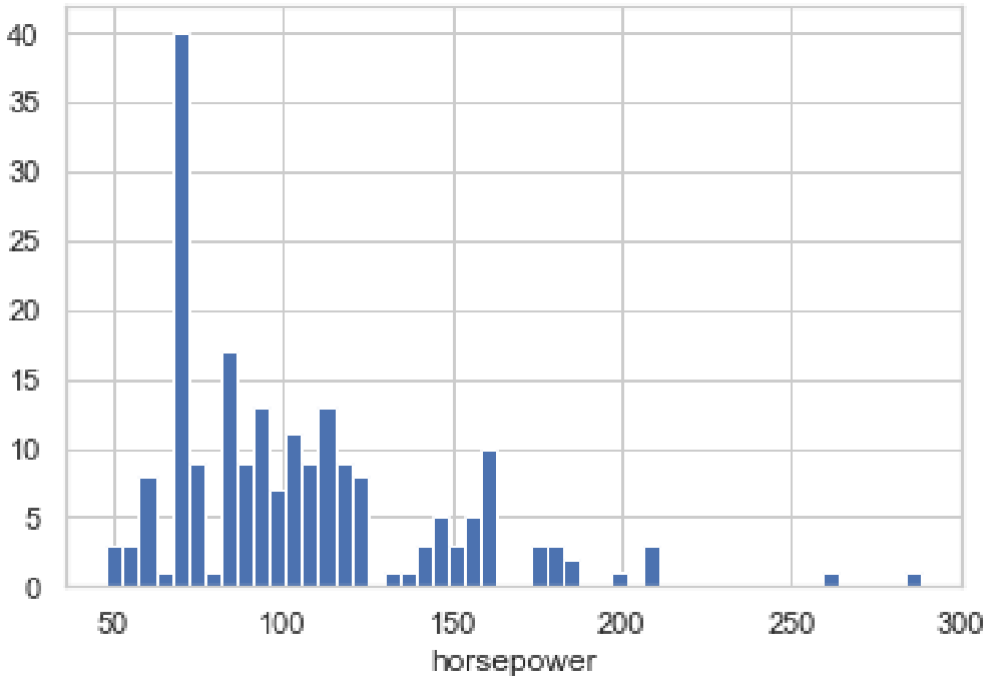
```
C:\Users\miair\Anaconda3\lib\site-packages\numpy\lib\histograms.py:824: RuntimeWarning
: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
C:\Users\miair\Anaconda3\lib\site-packages\numpy\lib\histograms.py:825: RuntimeWarning
  : invalid value encountered in
  less_equal keep &= (tmp_a <= last_edge)
```

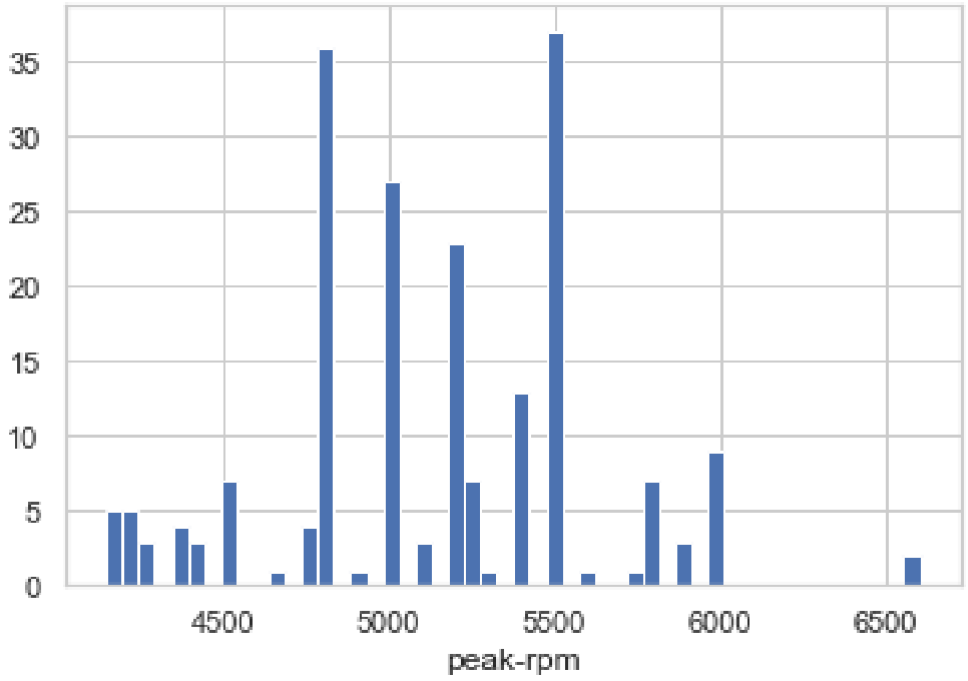


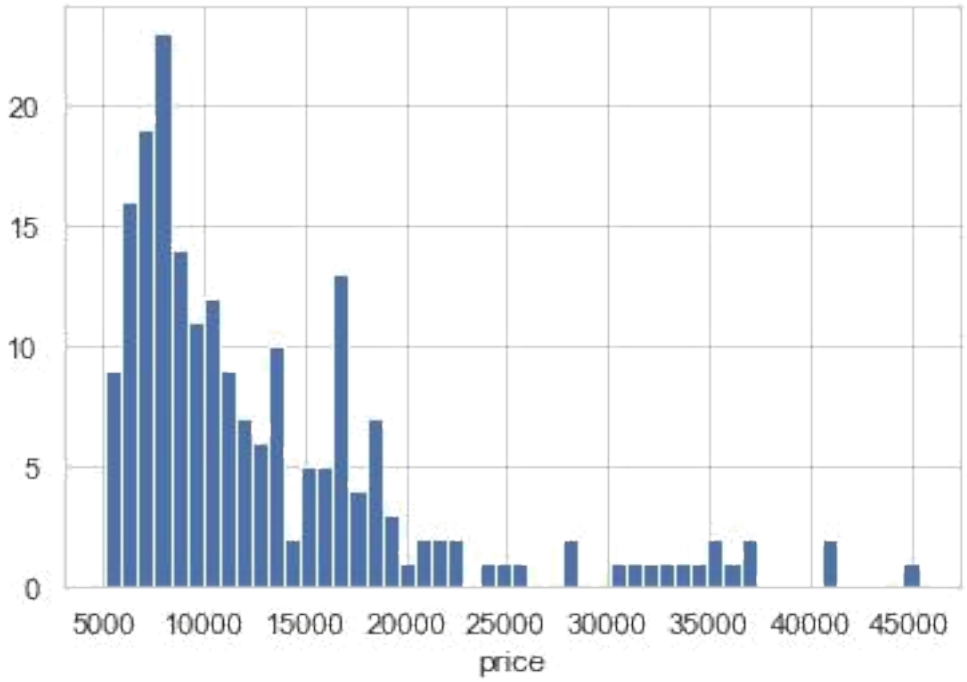



25
20
15
10
5
0
2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0
bore


40
35
30
25
20
15
10
5
0
2.0
2.5
3.0
3.5
4.0
stroke


40
35
30
25
20
15
10
5
0
50
100
150
200
250
300
horsepower


35
30
25
20
15
10
5
0
4500
5000
5500
6000
6500
peak-rpm



In [63]:

```
# Фильтр по пустым значениям поля normalized-losses
data[data['normalized-losses'].isnull()].head(10)
```

Out[63]:

	symboling	normalized-losses	
0	3	NaN	alfa-ron erc
1	3	NaN	alfa-ron erc
2	1	NaN	alfa-ron erc
5	2	NaN	aud i
7	1	NaN	aud i
9	0	NaN	aud i

14	1	NaN	bmw
15	0	NaN	bmw

	symboling	normalized-losses	
16	0	NaN	bmw
17	0	NaN	bmw

10 rows × 26 columns

In [64]:

```
  # Запоминаем индексы строк с пустыми значениями
flt_index = data[data['normalized-
losses'].isnull()].index flt_index
```

Out[64]:

```
Int64Index([

            192, 193],
           dtype='int64')
```

In [65]:

```
  # Проверяем что выводятся нужные строки
data[data.index.isin(flt_index)].head(10)
```

Out[65]:

	symboling	normalized-losses	
0	3	NaN	alfa-rom erc
1	3	NaN	alfa-rom erc
2	1	NaN	alfa-rom erc
5	2	NaN	audi
7	1	NaN	audi

9	0	NaN	audi
14	1	NaN	bmw

	symboling	normalized-losses	
15	0	NaN	bmw
16	0	NaN	bmw
17	0	NaN	bmw

10 rows × 26 columns

In [66]:

```
  # фильтр по колонке
data_num[data_num.index.isin(flt_index)]['normalized-losses']
```

Out[66]:

```
1    NaN
2    NaN
3    NaN

7    NaN
9    NaN
14   NaN
15   NaN
16   NaN
17   NaN
43   NaN
44   NaN
45   NaN
46   NaN
48   NaN
49   NaN
63   NaN
66   NaN
71   NaN
73   NaN
74   NaN
75   NaN
82   NaN
83   NaN
84   NaN
109  NaN
110  NaN
113  NaN
114  NaN
124  NaN
126  NaN
127  NaN
128  NaN
129  NaN
130  NaN
131  NaN
181  NaN
189  NaN
191  NaN
192  NaN
193  NaN
Name: normalized-losses, dtype: float64
```

Будем использовать встроенные средства импьютации библиотеки scikit-learn - https://scikit-

learn.org/stable/modules/impute.html#impute

In [67]:

```
data_num_norm_loss = data_num[['normalized-losses']]
data_num_norm_loss.head()
```

Out[67]:

	normalized-losses
0	NaN
1	NaN
2	NaN
3	164.0
4	164.0

In [68]:

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

In [69]:

```
  # Фильтр для проверки заполнения пустых значений
indicator = MissingIndicator()
mask_missing_values_only = indicator.fit_transform(data_num_norm_loss)
mask_missing_values_only
```

Out[69]:

```
array([[ True],
       [ True],
       [ True],
       [False],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [ True],
       [ True],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
```

```
[False],
[False],
[False],
[False],
[ True],
[ True],
[ True],
[ True],
[False],
[ True],
[ True],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[ True],
[False],
[False],
[ True],
[False],
[False],
[False],
[False],
[ True],
[False],
[ True],
[ True],
[ True],
[False],
[False],
[False],
[False],
[False],
[False],
[ True],
[ True],
[ True],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[ True],
```

```
[ True],
[False],
[False],
[ True],
[ True],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[ True],
[False],
[ True],
[ True],
[ True],
[ True],
[ True],
[ True],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
```

```
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [ True],
       [ True],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False]])
```

С помощью класса SimpleImputer проведем импьютацию с различными показателями центра распределения ("среднее", "медиана", "самое частое").

In [70]:

```
strategies=['mean', 'median','most_frequent']
```

In [71]:

```
def test_num_impute(strategy_param):
    imp_num = SimpleImputer(strategy=strategy_param)
    data_num_imp = imp_num.fit_transform(data_num_norm_loss)
    return data_num_imp[mask_missing_values_only]
```

In [72]:

```
strategies[0], test_num_impute(strategies[0])
```

Out[72]:

```
('mean',
       array([122., 122., 122., 122., 122., 122., 122., 122., 122., 122.,
  122., 122., 122., 122., 122., 122., 122., 122., 122., 122., 122., 122.,
       122., 122., 122., 122., 122., 122., 122., 122., 122., 122., 122.,
       122., 122., 122., 122., 122., 122., 122., 122.]))
```

In [73]:

```
strategies[1], test_num_impute(strategies[1])
```

Out[73]:

```
('median',
       array([115., 115., 115., 115., 115., 115., 115., 115., 115., 115.,
  115., 115., 115., 115., 115., 115., 115., 115., 115., 115., 115., 115.,
       115., 115., 115., 115., 115., 115., 115., 115., 115., 115., 115.,
       115., 115., 115., 115., 115., 115., 115., 115.]))
```

In [74]:

```
strategies[2], test_num_impute(strategies[2])
```

Out[74]:

```
('most_frequent',
       array([161., 161., 161., 161., 161., 161., 161., 161., 161., 161.,
  161., 161., 161., 161., 161., 161., 161., 161., 161., 161., 161., 161.,
```

```
        161., 161., 161., 161., 161., 161., 161., 161., 161., 161., 161.,
        161., 161., 161., 161., 161., 161., 161., 161.]))
```

In [75]:

```
  # Более сложная функция, которая позволяет задавать колонку и вид импьютации
def test_num_impute_col(dataset, column, strategy_param):
    temp_data = dataset[[column]]

    indicator = MissingIndicator()
    mask_missing_values_only = indicator.fit_transform(temp_data)

    imp_num = SimpleImputer(strategy=strategy_param)
    data_num_imp = imp_num.fit_transform(temp_data)

    filled_data = data_num_imp[mask_missing_values_only]

    return column, strategy_param, filled_data.size, filled_data[0], filled_data[fille
d_data.size-1]
```

In [76]:

```
data[['normalized-losses']].describe()
```

Out[76]:

count	
mean	122.000000
std	
min	
25%	
50%	
75%	
max	

In [77]:

```
test_num_impute_col(data, 'normalized-losses', strategies[0])
```

Out[77]:

```
('normalized-losses', 'mean', 41, 122.0, 122.0)
```

In [78]:

```
test_num_impute_col(data, 'normalized-losses', strategies[1])
```

Out[78]:

```
('normalized-losses', 'median', 41, 115.0, 115.0)
```

In [79]:

```
test_num_impute_col(data, 'normalized-losses', strategies[2])
```

Out[79]:

```
('normalized-losses', 'most_frequent', 41, 161.0, 161.0)
```

Из описания набора данных известно, что значат все признаки. Учитывая их взаимосвязь между собой, можно попробовать приблизительно посчитать возможные значения пропущенных данных.

Попробуем это сделать на примере колонки с наибольшим количеством пропусков normalized-losses.
Раз этот показатель вычисляется среди автомобилей определнной классификации, то мы можем попробовать заменить пропуски данных в этой колонке средним значением этого показателя машин одного класса. Выберем в качестве классификации тип кузова. Например, если машина с пропущенной normalized-value имеет тип кузова sedan, то посчитаем среднее для всех sedan и вставим в пропуск.
Алгоритм написан таким образом, что работает с tuples для итерации по циклу. Из-за технических особенностей кортежей потребуется сначала немного переименовать колонки: везде поменять - на _.

In [80]:

```
data.columns = data.columns.str.replace('-', '_')
data.isnull().sum()
```

Out[80]:

```
symboling
normalized_losses
make
fuel_type
aspiration
num_of_doors
body_style
drive_wheels
engine_location
wheel_base
length
width
height
curb_weight
engine_type
num_of_cylinders
engine_size
fuel_system
bore
stroke
compression_ratio
horsepower
peak_rpm
city_mpg
highway_mpg
price
dtype: int64
```

In [81]:

```
test_data = data.copy()
for row in test_data.itertuples():
     if np.isnan(row.normalized_losses):
            body_style_data = test_data.loc[test_data['body_style']== row.body_style]
            test_data.at[row.Index, 'normalized_losses'] = body_style_data['normalized
_losses'].mean()
test_data.isnull().sum()
```

Out[81]:

```
symboling           0
normalized_losses   0
make                0
fuel_type           0
aspiration          0
num_of_doors        2
body_style          0
```

```
drive_wheels
engine_location
wheel_base
length
width
height
curb_weight
engine_type
num_of_cylinders
engine_size
fuel_system
bore
stroke
compression_ratio
horsepower
peak_rpm
city_mpg
highway_mpg
price
dtype: int64
```

Таким образом, мы убрали все пропуски в normalized_losses. Но можно сделать эти данные точнее.

Будем считать normalized-losses не просто для машин с одним кузовом, но только для тех, у кого кол-во лошадиных сил приблизительно совпадает с этим кол-вом у машины с пропуском данных.

Для этого сначала уберем пропуски в колонке horsepower. Обычно чем больше двигатель, тем больше л.с. он имеет. Поэтому будем искать дргуие автомобили с хотя бы похожим размером двигателя engine_size.

In [82]:

```
  # Оценим стандартное отклонение размеров двигателя
data[['engine_size']].std()
```

Out[82]:

```
engine_size    41.642693
dtype: float64
```

In [83]:

```
for row in data.itertuples():
     if np.isnan(row.horsepower):
            #Возьмём разборс размеров двигателей, равным половине от стандратного откл
онения
            engine_size_data = data.loc[((row.engine_size - 20) < data['engine_size'])
& (data['engine_size'] < (row.engine_size + 20))]
            data.at[row.Index, 'horsepower'] = engine_size_data['horsepower'].mean()
data['horsepower'].isnull().sum()
```

Out[83]:

```
0
```

Теперь усложним наш алгоритм подсчета normalized-value. Будем среди машин с одинаковым кузовом искать те, у которых отличается кол-во л.с. максимум на 30 единиц.

Оказалось, что могут попадаться машины, кол-во л.с. которых будет сильно отличаться от машин того же кузова. Для таких машин будем вести подсчет только среди машин с тем же кузовом, без учета кол-ва л.с.

In [84]:

```
for row in data.itertuples():
```

```
    if np.isnan(row.normalized_losses):
            body_style_data = data.loc[data['body_style']== row.body_style]
            hp_and_body_data = body_style_data.loc[((row.horsepower - 30) < data['hors
epower']) & (data['horsepower'] < (row.horsepower + 30))]
            if hp_and_body_data.shape[0] <= 1:
                data.at[row.Index, 'normalized_losses'] = body_style_data['normalized_
losses'].mean()
            else:
                data.at[row.Index, 'normalized_losses'] = hp_and_body_data['normalized
_losses'].mean()
              #   Если и это не помогло, то просто считаем среднее по колонке:
            if np.isnan(data.at[row.Index, 'normalized_losses']):
                data.at[row.Index, 'normalized_losses'] = data['normalized_losses'].me
an()
data.isnull().sum()
```

Out[84]:

```
symboling
normalized_losses
make
fuel_type
aspiration
num_of_doors
body_style
drive_wheels
engine_location
wheel_base
length
width
height
curb_weight
engine_type
num_of_cylinders
engine_size
fuel_system
bore
stroke
compression_ratio
horsepower
peak_rpm
city_mpg
highway_mpg
price
dtype: int64
```

Для кол-ва дверей выберем для пропусков то кол-во, которое чаще повторяется среди машин того же кузова.

In [85]:

```
for row in data.itertuples():
    if np.isnan(row.num_of_doors):
            data_body_doors = data.loc[data['body_style'] == row.body_style, 'num_of_d
oors']
            data.at[row.Index, 'num_of_doors'] = data_body_doors.value_counts().idxmax
()
data['num_of_doors'].isnull().sum()
```

Out[85]:

```
0
```

Обработаем остальные пропуски данных¶

bore - диаметр цилиндра
stroke - ход поршня Эти параметры вероятнее всего зависят от размера двигателя.
Используем этот факт для более точного заполнения пропусков.

In [86]:

```
for row in data.itertuples():
     if np.isnan(row.bore):
            #Возьмём разборс размеров двигателей, равным половине от стандратного откл
онения
            engine_size_data = data.loc[((row.engine_size - 20) < data['engine_size'])
& (data['engine_size'] < (row.engine_size + 20))]
            data.at[row.Index, 'bore'] = engine_size_data['bore'].mean()
     if np.isnan(row.stroke):
            engine_size_data = data.loc[((row.engine_size - 20) < data['engine_size'])
& (data['engine_size'] < (row.engine_size + 20))]
            data.at[row.Index, 'stroke'] = engine_size_data['stroke'].mean()
print('Кол-во пропусков в bore:', data['bore'].isnull().sum())
print('Кол-во пропусков в stroke:', data['stroke'].isnull().sum())
```

```
Кол-во пропусков в bore: 0
Кол-во пропусков в stroke: 0
```

peak-rpm - макс. число оборотов двигателя в минуту Скорее всего зависит от кол-ва лошадиных сил в двигателе.

In [87]:

```
for row in data.itertuples():
     if np.isnan(row.peak_rpm):
            #Возьмём разборс размеров двигателей, равным половине от стандратного откл
онения
            hp_data = data.loc[((row.horsepower - 30) < data['horsepower']) & (data['h
orsepower'] < (row.horsepower + 30))]
            data.at[row.Index, 'peak_rpm'] = engine_size_data['peak_rpm'].mean()
print('Кол-во пропусков в peak_rpm:', data['peak_rpm'].isnull().sum())
```

```
Кол-во пропусков в peak_rpm: 0
```

Для заполнения пропусков в ценах автомобилей применим тот же метод, что и для заполнения normalized-value.

In [88]:

```
for row in data.itertuples():
     if np.isnan(row.price):
            body_style_data = data.loc[data['body_style']== row.body_style]
            hp_and_body_data = body_style_data.loc[((row.horsepower - 30) < data['hors
epower']) & (data['horsepower'] < (row.horsepower + 30))]
            if hp_and_body_data.shape[0] <= 1:
                data.at[row.Index, 'price'] = body_style_data['price'].mean()
            else:
                data.at[row.Index, 'price'] = hp_and_body_data['price'].mean()
            # Если и это не помогло, то просто считаем среднее по колонке:
```

```
            if np.isnan(data.at[row.Index, 'price']):
                data.at[row.Index, 'price'] = data['price'].mean()
print('Кол-во пропусков в price:', data['price'].isnull().sum())
```

```
Кол-во пропусков в price: 0
```

In [89]:

```
data.isnull().sum()
```

Out[89]:

```
symboling
normalized_losses
make
fuel_type
aspiration
num_of_doors
body_style
drive_wheels
engine_location
wheel_base
length
width
height
curb_weight
engine_type
num_of_cylinders
engine_size
fuel_system
bore
stroke
compression_ratio
horsepower
peak_rpm
city_mpg
highway_mpg
price
dtype: int64
```

Добились удаления всех пропусков.

1.2.2. Обработка пропусков в категориальных данных¶

Используем датасет о домах в Мельбурне.

```
data = pd.read_csv('data/melb_data.csv', sep=',')

data.head()
```

	Suburb	Address	om
	Abbotsford	85 Turner St	2
	Abbotsfor	25 Bloombur	2

	d	g St	
2	Abbotsford	5 Charles St	3

	Suburb	Address	Rooms
3	Abbotsford	40 Federation La	3
4	Abbotsford	55a Park St	4

5 rows × 21 columns

In [93]:

```
data.shape
```

Out[93]:

```
(13580,
```

```
data.dtypes
```

```
Suburb
Address
Rooms
Type
Price
Method
SellerG
Date
Distance
Postcode
Bedroom2
Bathroom
Car
Landsize
BuildingArea
YearBuilt
CouncilArea
Lattitude
Longtitude
Regionname
Propertycount
dtype: object
```

In [95]:

```
data.isnull().sum()
```

Out[95]:

```
Suburb
Address
Rooms
Type
Price
Method
SellerG
Date
Distance
Postcode
Bedroom2
Bathroom
```

Car

```
Landsize            0
BuildingArea     6450
YearBuilt        5375
CouncilArea      1369
Lattitude           0
Longtitude          0
Regionname          0
Propertycount       0
dtype: int64
```

Рассмотрим колонку CouncilArea.

In [96]:

```
  # Выберем категориальные колонки с пропущенными значениями

  # Цикл по колонкам датасета

cat_cols = []

total_count = data.shape[0]

for col in data.columns:

      #    Количество пустых значений

    temp_null_count = data[data[col].isnull()].shape[0]

    dt = str(data[col].dtype)

    if temp_null_count>0 and (dt=='object'):

        cat_cols.append(col)

        temp_perc = round((temp_null_count / total_count) * 100.0, 2)

        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(
col, dt, temp_null_count, temp_perc))
```

```
Колонка CouncilArea. Тип данных object. Количество пустых значений 1369, 10.08%.
```

In [97]:

```
  # Выберем даннее только из этой колонки

cat_temp_data = data[['CouncilArea']]

cat_temp_data.head()
```

Out[97]:

	CouncilArea
0	Yarra
1	Yarra
2	Yarra
3	Yarra
4	Yarra

In [98]:

```
  # Названия всех муниципалитетов (все уникальные значения колонки)

cat_temp_data['CouncilArea'].unique()
```

Out[98]:

```
array(['Yarra', 'Moonee Valley', 'Port Phillip', 'Darebin', 'Hobsons Bay',
       'Stonnington', 'Boroondara', 'Monash', 'Glen Eira', 'Whitehorse',
       'Maribyrnong', 'Bayside', 'Moreland', 'Manningham', 'Banyule',
       'Melbourne', 'Kingston', 'Brimbank', 'Hume', nan, 'Knox',
       'Maroondah', 'Casey', 'Melton', 'Greater Dandenong', 'Nillumbik',
       'Whittlesea', 'Frankston', 'Macedon Ranges', 'Yarra Ranges',
       'Wyndham', 'Cardinia', 'Unavailable', 'Moorabool'], dtype=object)
```

In [99]:

```
  # Размер колонки
```

```
cat_temp_data[cat_temp_data['CouncilArea'].isnull()].shape
```

Out[99]:

```
(1369, 1)
```

In [100]:

```
  # Импьютация наиболее частыми значениями
imp2 = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
data_imp2 = imp2.fit_transform(cat_temp_data) data_imp2
```

Out[100]:

```
array([['Yarra'],
       ['Yarra'],
       ['Yarra'],
       ...,
       ['Moreland'],
       ['Moreland'],
       ['Moreland']], dtype=object)
```

In [101]:

```
  # Пустые значения отсутствуют
np.unique(data_imp2)
```

Out[101]:

```
array(['Banyule', 'Bayside', 'Boroondara', 'Brimbank', 'Cardinia', 'Casey',
       'Darebin', 'Frankston', 'Glen Eira', 'Greater Dandenong', 'Hobsons
       Bay', 'Hume', 'Kingston', 'Knox', 'Macedon Ranges', 'Manningham',
       'Maribyrnong', 'Maroondah', 'Melbourne', 'Melton', 'Monash', 'Moonee
       Valley', 'Moorabool', 'Moreland', 'Nillumbik', 'Port Phillip',
       'Stonnington', 'Unavailable', 'Whitehorse', 'Whittlesea', 'Wyndham',
       'Yarra', 'Yarra Ranges'], dtype=object)
```

Избавились от пропусков в категориальных данных.

2.Преобразование категориальных признаков в числовые¶

Преобразуем названия муниципалитетов в числовые значения.

In [102]:

```
cat_enc = pd.DataFrame({'c1':data_imp2.T[0]})
cat_enc
```

Out[102]:

0	
1	
2	
3	
4	
...	
13575	Moreland
13576	Moreland
13577	Moreland
13578	Moreland
13579	Moreland

13580 rows × 1 columns

In [103]:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

Для преобразования данных в числовые будем использовать LabelEncoder.

In [104]:

```
le = LabelEncoder()
cat_enc_le = le.fit_transform(cat_enc['c1'])
```

In [105]:

```
cat_enc['c1'].unique()
```

Out[105]:

```
array(['Yarra', 'Moonee Valley', 'Port Phillip', 'Darebin', 'Hobsons Bay',
       'Stonnington', 'Boroondara', 'Monash', 'Glen Eira', 'Whitehorse',
       'Maribyrnong', 'Bayside', 'Moreland', 'Manningham', 'Banyule',
       'Melbourne', 'Kingston', 'Brimbank', 'Hume', 'Knox', 'Maroondah',
       'Casey', 'Melton', 'Greater Dandenong', 'Nillumbik', 'Whittlesea',
       'Frankston', 'Macedon Ranges', 'Yarra Ranges', 'Wyndham',
       'Cardinia', 'Unavailable', 'Moorabool'], dtype=object)
```

За каждым муниципалитетом теперь закреплен номер.

In [106]:

```
np.unique(cat_enc_le)
```

Out[106]:

```
array([
```

```
0,
17,
```

In [107]:

```
le.inverse_transform([0, 1, 2, 3, 4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32])
```

Out[107]:

```
array(['Banyule', 'Bayside', 'Boroondara', 'Brimbank', 'Cardinia', 'Casey',
       'Darebin', 'Frankston', 'Glen Eira', 'Greater Dandenong', 'Hobsons
       Bay', 'Hume', 'Kingston', 'Knox', 'Macedon Ranges', 'Manningham',
       'Maribyrnong', 'Maroondah', 'Melbourne', 'Melton', 'Monash', 'Moonee
       Valley', 'Moorabool', 'Moreland', 'Nillumbik', 'Port Phillip',
       'Stonnington', 'Unavailable', 'Whitehorse', 'Whittlesea', 'Wyndham',
       'Yarra', 'Yarra Ranges'], dtype=object)
```

Перед кодированием признаки сортируются в алфавитном порядке.

2.2. Кодирование категорий наборами бинарных значений - one-hot encoding¶

In [108]:

```
ohe = OneHotEncoder()
cat_enc_ohe = ohe.fit_transform(cat_enc[['c1']])
```

In [109]:

```
  # Исходный размер данных. Просто колонка.
cat_enc.shape
```

Out[109]:

```
(13580, 1)
```

Каждое уникальное значение было закодировано набором из 33 единиц и нулей.

In [110]:

```
cat_enc_ohe.shape
```

Out[110]:

```
(13580, 33)
```

In [111]:

```
cat_enc_ohe
```

Out[111]:

```
<13580x33 sparse matrix of type '<class 'numpy.float64'>'
	with 13580 stored elements in Compressed Sparse Row format>
```

In [112]:

```
cat_enc_ohe.todense()[504:509]
```

Out[112]:

```
matrix([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
         0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
         0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0.]])
```

In [113]:

```
cat_enc[504:509]
```

Out[113]:

	c1
504	Port Phillip
505	Port Phillip
506	Boroondara
507	Boroondara
508	Boroondara

Использование Pandas get_dummies - быстрого варианта one-hot кодирования¶

In [114]:

```
pd.get_dummies(cat_enc)[504:509]
```

Out[114]:

	c1_Banyule	c1_Bayside	c1_Bayndara
5004	0	0	0
5005	0	0	0
5006	0	0	1
5007	0	0	1
5008	0	0	1

5 rows × 33 columns

In [115]:

```
pd.get_dummies(cat_temp_data, dummy_na=True)[504:509]
```

Out[115]:

	Council Area_Banyule	Council Area_Bayside	Coun
504	0	0	0
505	0	0	0
506	0	0	1
507	0	0	1
508	0	0	1

5 rows × 34 columns

3. Масштабирование данных¶

In [116]:

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler, Normalizer
```

In [117]:

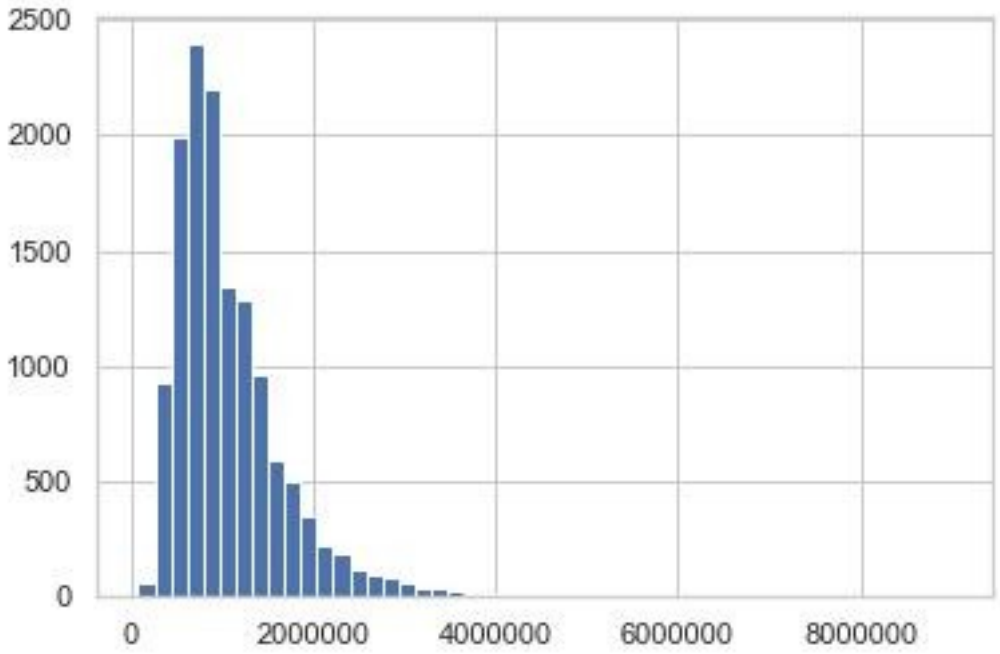
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[['Price']])
```

Применим масштабирование данных на колонке Price используемого набора данных.

3.1 MinMax-масштабирование¶

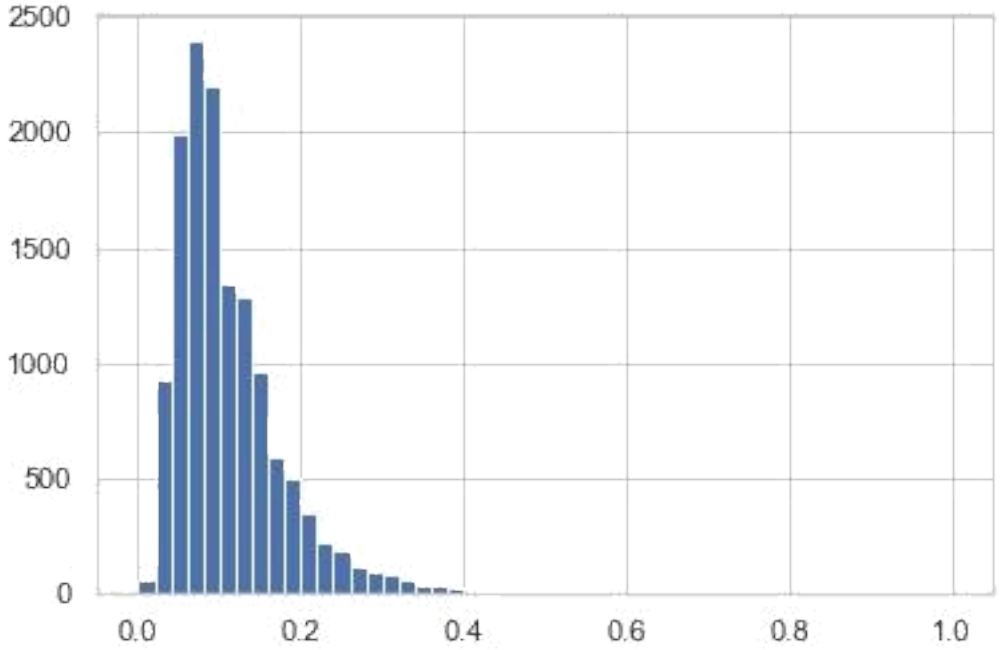
In [118]:

```
  # До масштабирования:
plt.hist(data['Price'], 50)
plt.show()
```



In [119]:

```
  # После
plt.hist(sc1_data, 50)
plt.show()
```



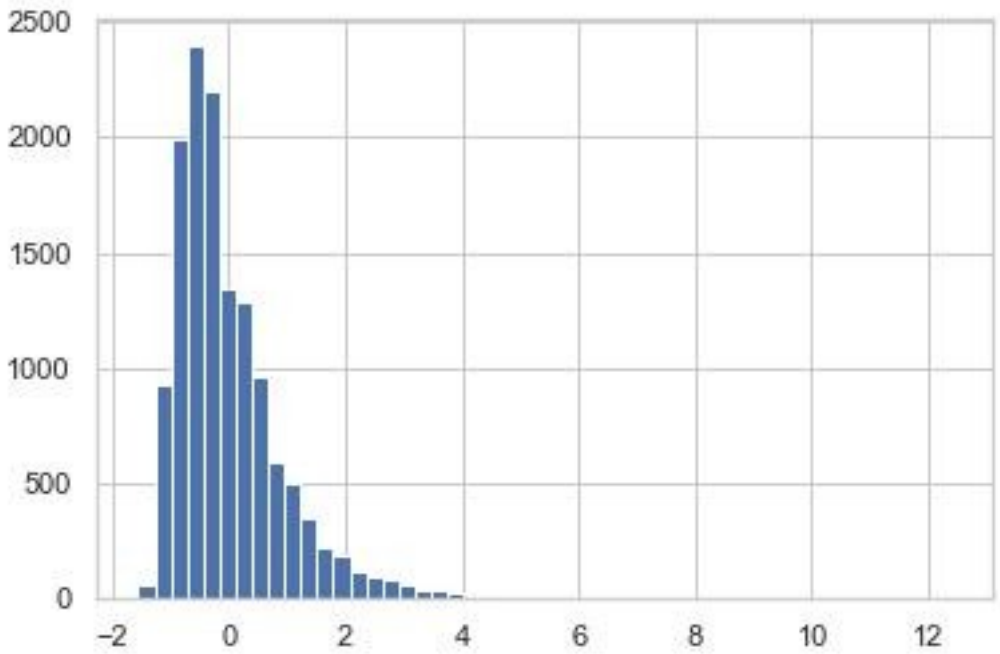
3.2. Масштабирование данных на основе Z-оценки - StandardScale¶

In [120]:

```
sc2 = StandardScaler()
sc2_data = sc2.fit_transform(data[['Price']])
```

In [121]:

```
plt.hist(sc2_data, 50)
plt.show()
```



3.3. Нормализация данных¶

In [122]:

```
sc3 = Normalizer()
sc3_data = sc3.fit_transform(data[['Price']])
```

In [123]:

```
plt.hist(sc3_data, 50)
plt.show()
```