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 «Системы обработки информации и управления»

Отчет по лабораторной работе №2

по теме «*Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных*.»

по дисциплине «Технологии машинного обучения»

Выполнил:
Студент группы ИУ5-63Б Лебедева С.К.

Проверил:
к.т.н., доц., Гапанюк Ю.Е.

2024 г.

Цкль:

Изучение способов предварительной обработки данных для дальнейшего формирования моделей.

Задание:

1. Выбрать набор данных (датасет), содержащий категориальные признаки и пропуски в данных. Для выполнения следующих пунктов можно использовать несколько различных наборов данных (один для обработки пропусков, другой для категориальных признаков и т.д.)
2. Для выбранного датасета (датасетов) на основе материалов лекции решить следующие задачи:
    - обработку пропусков в данных;
    - кодирование категориальных признаков;
    - масштабирование данных.

Текст программы и экранные формы:

Ссылка на Colab: https://colab.research.google.com/drive/1aIMRzW7xNzNt-GV5IYyDgkK47LANLDNr?usp=sharing

Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных.

Мы научимся обрабатывать пропуски в данных для количественных (числовых) и категориальных признаков и масштабировать данные. Также мы научимся преобразовывать категориальные признаки в числовые.

В чем состоит проблема?

- Если в данных есть пропуски, то большинство алгоритмов машинного обучения не будут с ними работать. Даже корреляционная матрица не будет строиться корректно.
- Большинство алгоритмов машинного обучения требуют явного перекодирования категориальных признаков в числовые. Даже если алгоритм не требует этого явно, такое перекодирование возможно стоит попробовать, чтобы повысить качество модели.
- Большинство алгоритмов показывает лучшее качество на масштабированных признаках, в особенности алгоритмы, использующие методы градиентного спуска.

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

Загрузка и первичный анализ данных

https://www.kaggle.com/datasets/syedanwarafridi/vehicle-sales-data

```
# Будем использовать только обучающую выборку
data = pd.read_csv('car_prices.csv', sep=",")
```

```
[ ] # размер набора данных
data.shape
```

```
(458983, 16)
```

```
[ ] # типы колонок
data.dtypes
```

```
year              int64
make             object
model            object
trim             object
body             object
transmission     object
vin              object
state            object
condition       float64
odometer        float64
color            object
interior         object
seller           object
mmr             float64
sellingprice    float64
saledate         object
dtype: object
```

```
[ ] # проверим есть ли пропущенные значения
data.isnull().sum()
```

```
year              0
make           8202
model          8274
trim           8531
body          11075
transmission  52143
vin               0
state             0
condition      11802
odometer          90
color            643
interior         643
seller             0
mmr               18
sellingprice       9
saledate          10
dtype: int64
```

```
# Первые 5 строк датасета
data.head()
```

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	mmr	sellingprice	saledate
0	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	5.0	16639.0	white	black	kia motors america inc	20500.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
1	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg561319	ca	5.0	9393.0	white	beige	kia motors america inc	20800.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2	2014	BMW	3 Series	328i SULEV	Sedan	automatic	wba3c1c51ek116351	ca	45.0	1331.0	gray	black	financial services remarketing (lease)	31900.0	30000.0	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
3	2015	Volvo	S60	T5	Sedan	automatic	yv1612tb4f1310987	ca	41.0	14282.0	white	black	volvo na rep/world omni	27500.0	27750.0	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
4	2014	BMW	6 Series Gran Coupe	650i	Sedan	automatic	wba6b2c57ed129731	ca	43.0	2641.0	gray	black	financial services remarketing (lease)	66000.0	67000.0	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)

```
[ ] total_count = data.shape[0]
print('Всего строк: {}'.format(total_count))
```

```
Всего строк: 458983
```

Обработка пропусков в данных

Простые стратегии - удаление или заполнение нулями

Удаление колонок, содержащих пустые значения `res = data.dropna(axis=1, how='any')`

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

Документация

Удаление может производиться для группы строк или колонок.

```
[ ] # Удаление колонок, содержащих пустые значения
    data_new_1 = data.dropna(axis=1, how='any')
    (data.shape, data_new_1.shape)
```

```
((458983, 16), (458983, 4))
```

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

```
((458983, 16), (387537, 16))
```

```
data.head()
```

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	mmr	sellingprice	saledate
0	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	5.0	16639.0	white	black	kia motors america inc	20500.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
1	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg561319	ca	5.0	9393.0	white	beige	kia motors america inc	20800.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2	2014	BMW	3 Series	328i SULEV	Sedan	automatic	wba3c1c51ek116351	ca	45.0	1331.0	gray	black	financial services remarketing (lease)	31900.0	30000.0	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
3	2015	Volvo	S60	T5	Sedan	automatic	yv1612tb4f1310987	ca	41.0	14282.0	white	black	volvo na rep/world omni	27500.0	27750.0	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
4	2014	BMW	6 Series Gran Coupe	650i	Sedan	automatic	wba6b2c57ed129731	ca	43.0	2641.0	gray	black	financial services remarketing (lease)	66000.0	67000.0	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)

```
[ ] # Заполнение всех пропущенных значений нулями
    # В данном случае это некорректно, так как нулями заполняются в том числе категориальные колонки
    data_new_3 = data.fillna(0)
    data_new_3.head()
```

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	mmr	sellingprice	saledate
0	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	5.0	16639.0	white	black	kia motors america inc	20500.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
1	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg561319	ca	5.0	9393.0	white	beige	kia motors america inc	20800.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2	2014	BMW	3 Series	328i SULEV	Sedan	automatic	wba3c1c51ek116351	ca	45.0	1331.0	gray	black	financial services remarketing (lease)	31900.0	30000.0	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
3	2015	Volvo	S60	T5	Sedan	automatic	yv1612tb4f1310987	ca	41.0	14282.0	white	black	volvo na rep/world omni	27500.0	27750.0	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
4	2014	BMW	6 Series Gran Coupe	650i	Sedan	automatic	wba6b2c57ed129731	ca	43.0	2641.0	gray	black	financial services remarketing (lease)	66000.0	67000.0	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)

"Внедрение значений" - импьютация (imputation)

Обработка пропусков в числовых данных

```
# Выберем числовые колонки с пропущенными значениями
# Цикл по колонкам датасета
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 condition. Тип данных float64. Количество пустых значений 10929, 27.27%.
Колонка odometer. Тип данных float64. Количество пустых значений 68, 0.17%.
Колонка mmr. Тип данных float64. Количество пустых значений 1, 0.0%.
Колонка sellingprice. Тип данных float64. Количество пустых значений 1, 0.0%.
```


"Внедрение значений" - импьютация (imputation)

Обработка пропусков в числовых данных

```
# Выберем числовые колонки с пропущенными значениями
# Цикл по колонкам датасета
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))
```

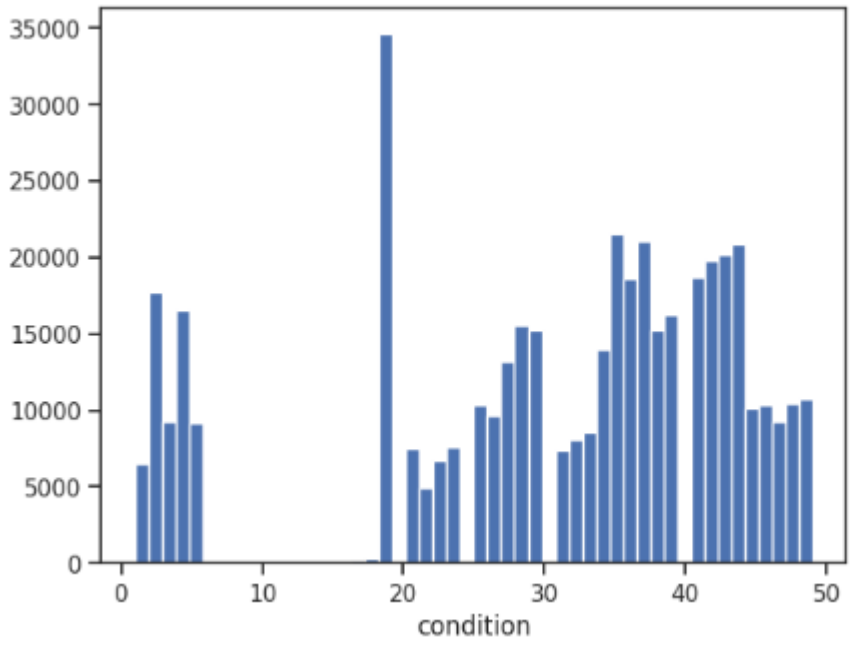
```
Колонка condition. Тип данных float64. Количество пустых значений 10929, 27.27%.
Колонка odometer. Тип данных float64. Количество пустых значений 68, 0.17%.
Колонка mmr. Тип данных float64. Количество пустых значений 1, 0.0%.
Колонка sellingprice. Тип данных float64. Количество пустых значений 1, 0.0%.
```

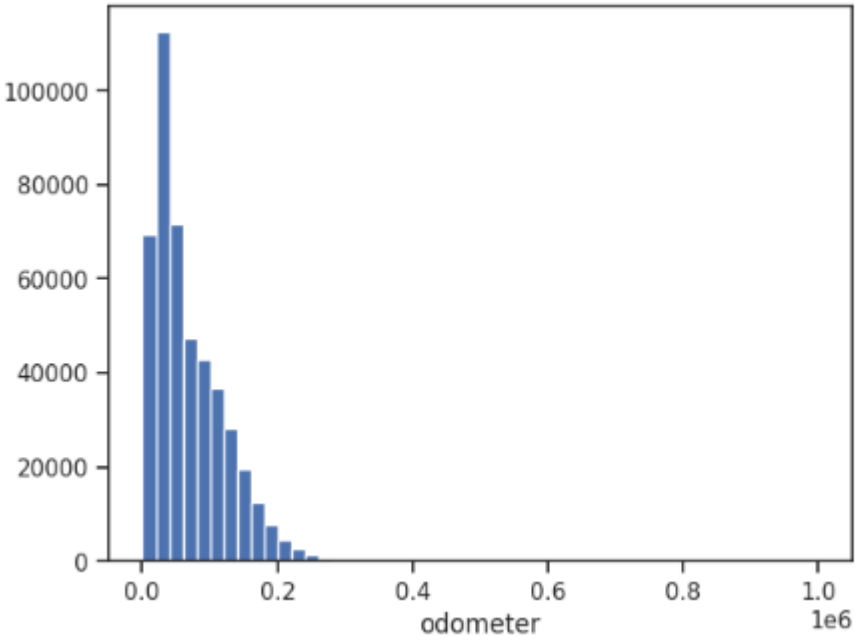

```
[ ] # Фильтр по колонкам с пропущенными значениями
    data_num = data[num_cols]
    data_num
```

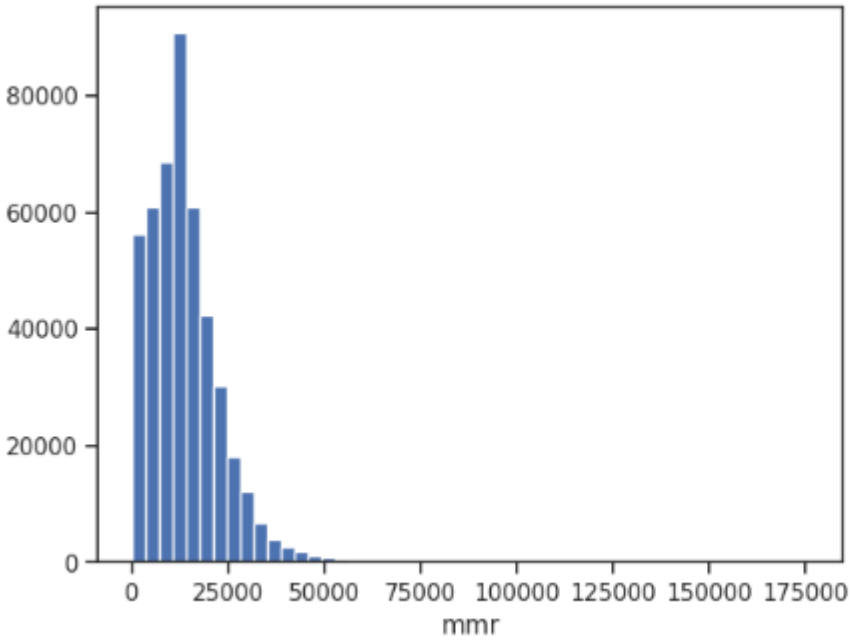
	condition	odometer	mmr	sellingprice
0	5.0	16639.0	20500.0	21500.0
1	5.0	9393.0	20800.0	21500.0
2	45.0	1331.0	31900.0	30000.0
3	41.0	14282.0	27500.0	27750.0
4	43.0	2641.0	66000.0	67000.0
...	...	...	...	...
458978	39.0	28670.0	19450.0	20300.0
458979	37.0	75525.0	10700.0	11000.0
458980	35.0	90725.0	14700.0	12687.0
458981	31.0	32808.0	23700.0	23800.0
458982	2.0	169959.0	11600.0	10100.0

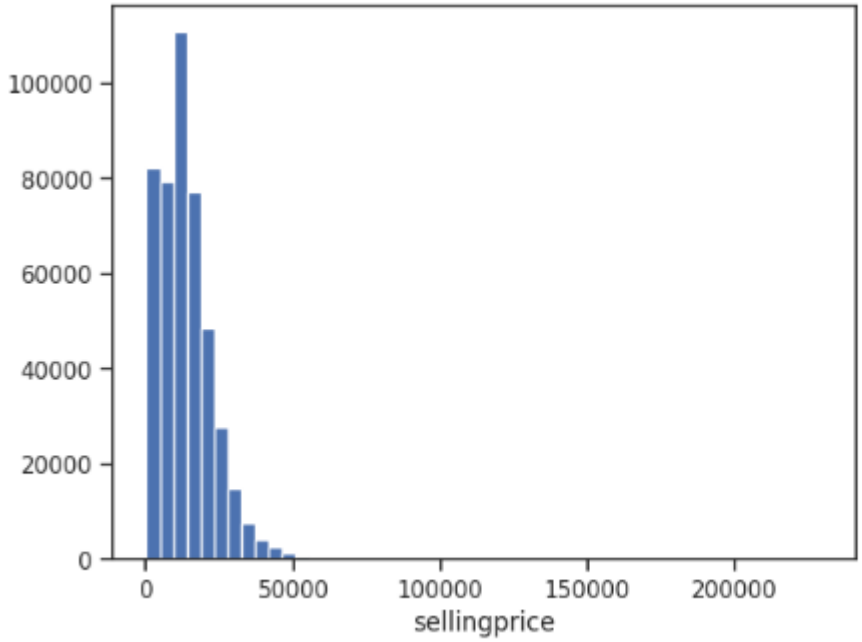
458983 rows × 4 columns

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


35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
50
condition


100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0
odometer
1e6


80000
60000
40000
20000
0
0
25000
50000
75000
100000
125000
150000
175000
mmr


100000
80000
60000
40000
20000
0
0
50000
100000
150000
200000
sellingprice

```
[ ] # 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[filled_data.size-1]
```

```
data[['mmr']].describe()
```

	mmr
count	458965.000000
mean	13599.871014
std	9488.783680
min	25.000000
25%	6925.000000
50%	12200.000000
75%	18150.000000
max	176000.000000

```
[ ] data[['sellingprice']].describe()
```

	sellingprice
count	458974.000000
mean	13428.317367
std	9562.206694
min	1.000000
25%	6700.000000
50%	12000.000000
75%	18000.000000
max	230000.000000

```
data[['odometer']].describe()
```

	odometer
count	458893.000000
mean	68885.740214
std	53876.979148
min	1.000000
25%	28422.000000
50%	52646.000000
75%	100230.000000
max	999999.000000

```
[ ] data[['condition']].describe()
```

	condition
count	447181.000000
mean	30.528549
std	13.532865
min	1.000000
25%	23.000000
50%	34.000000
75%	41.000000
max	49.000000

```
[ ] clean_n_data =  data.copy()
```

```
[ ] for col in num_cols:
        clean_n_data[col].fillna(clean_n_data[col].median(), inplace = True)
        print(clean_n_data[col].isna().sum())
```

```
0
0
0
0
```

```
[ ] test_num_impute_col(data, 'condition', strategies[0])
```

```
('condition', 'mean', 11802, 30.528548842638664, 30.528548842638664)
```

```
[ ] test_num_impute_col(data, 'condition', strategies[1])
```

```
('condition', 'median', 11802, 34.0, 34.0)
```

```
[ ] test_num_impute_col(data, 'condition', strategies[2])
```

```
('condition', 'most_frequent', 11802, 19.0, 19.0)
```

```
[ ] test_num_impute_col(data, 'odometer', strategies[0])
```

```
('odometer', 'mean', 90, 68885.74021394966, 68885.74021394966)
```

```
[ ] test_num_impute_col(data, 'odometer', strategies[1])
```

```
('odometer', 'median', 90, 52646.0, 52646.0)
```

```
[ ] test_num_impute_col(data, 'odometer', strategies[2])
```

```
('odometer', 'most_frequent', 90, 1.0, 1.0)
```

```
clean_n_data.isnull().sum()
```

```
year              0
make           8202
model          8274
trim           8531
body          11075
transmission  52143
vin               0
state             0
condition         0
odometer          0
color           643
interior        643
seller            0
mmr               0
sellingprice      0
saledate         10
dtype: int64
```

```
[ ]
```

Обработка пропусков в категориальных данных

```
# Выберем категориальные колонки с пропущенными значениями
# Цикл по колонкам датасета
cat_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=='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 make. Тип данных object. Количество пустых значений 8202, 1.79%.
Колонка model. Тип данных object. Количество пустых значений 8274, 1.8%.
Колонка trim. Тип данных object. Количество пустых значений 8531, 1.86%.
Колонка body. Тип данных object. Количество пустых значений 11075, 2.41%.
Колонка transmission. Тип данных object. Количество пустых значений 52143, 11.36%.
Колонка color. Тип данных object. Количество пустых значений 643, 0.14%.
Колонка interior. Тип данных object. Количество пустых значений 643, 0.14%.
Колонка saledate. Тип данных object. Количество пустых значений 10, 0.0%.
```

+ Код + Текст

```
[ ] cat_cols
```

```
['make',
 'model',
 'trim',
 'body',
 'transmission',
 'color',
 'interior',
 'saledate']
```

```
[ ] data_cat = data[cat_cols]
    data_cat
```

	make	model	trim	body	transmission	color	interior	saledate
0	Kia	Sorento	LX	SUV	automatic	white	black	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
1	Kia	Sorento	LX	SUV	automatic	white	beige	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2	BMW	3 Series	328i SULEV	Sedan	automatic	gray	black	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
3	Volvo	S60	T5	Sedan	automatic	white	black	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
4	BMW	6 Series Gran Coupe	650i	Sedan	automatic	gray	black	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)
...	...	...	...	...	...	...	...	...
458978	Subaru	Outback	2.5i Premium	Wagon	automatic	green	tan	Wed May 27 2015 03:30:00 GMT-0700 (PDT)
458979	Toyota	Camry	LE	Sedan	automatic	silver	gray	Tue May 26 2015 06:00:00 GMT-0700 (PDT)
458980	Toyota	Camry Hybrid	XLE	Sedan	automatic	red	gray	Wed May 27 2015 02:30:00 GMT-0700 (PDT)
458981	Toyota	Highlander	Base	SUV	automatic	gray	black	Wed May 27 2015 02:00:00 GMT-0700 (PDT)
458982	Toyota	Camry	SE	Sedan	automatic	blue	black	NaN

Какие из этих колонок Вы бы выбрали или не выбрали для построения модели?

Класс SimpleImputer можно использовать для категориальных признаков со стратегиями "most_frequent" или "constant".

```
cat_temp_data = data[['make']]
cat_temp_data.head()
```

	make
0	Kia
1	Kia
2	BMW
3	Volvo
4	BMW

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

```
array(['Kia', 'BMW', 'Volvo', 'Nissan', 'Chevrolet', 'Audi', 'Ford',
       'Hyundai', 'Buick', 'Cadillac', 'Acura', 'Lexus', 'Infiniti',
       'Jeep', 'Mercedes-Benz', 'Mitsubishi', 'Mazda', 'MINI',
       'Land Rover', 'Lincoln', 'lincoln', 'Jaguar', 'Volkswagen',
       'Toyota', 'Subaru', 'Scion', 'Porsche', nan, 'bmw', 'Dodge',
       'FIAT', 'Chrysler', 'ford', 'Ferrari', 'Honda', 'GMC',
       'mitsubishi', 'Ram', 'smart', 'chevrolet', 'Bentley', 'chrysler',
       'pontiac', 'Pontiac', 'Saturn', 'Maserati', 'Mercury', 'HUMMER',
       'landrover', 'cadillac', 'land rover', 'mercedes', 'mazda',
       'toyota', 'lexus', 'gmc truck', 'honda', 'nissan', 'porsche',
       'Saab', 'Suzuki', 'dodge', 'subaru', 'Oldsmobile', 'oldsmobile',
       'hyundai', 'jeep', 'Isuzu', 'dodge tk', 'Geo', 'acura',
       'volkswagen', 'suzuki', 'kia', 'audi', 'Rolls-Royce', 'gmc',
       'maserati', 'mazda tk', 'mercury', 'buick', 'hyundai tk',
       'mercedes-b', 'vw', 'Daewoo', 'chev truck', 'ford tk', 'plymouth',
       'Plymouth', 'ford truck', 'Tesla', 'airstream', 'dot',
       'Aston Martin', 'Fisker', 'Lamborghini', 'Lotus'], dtype=object)
```

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

```
(8202, 1)
```

```
cat_temp_data[['make']].describe()
```

	make
count	450781
unique	96
top	Ford
freq	78104

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

```
array([['Kia'],
       ['Kia'],
       ['BMW'],
       ...,
       ['Toyota'],
       ['Toyota'],
       ['Toyota']], dtype=object)
```

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

```
array(['Acura', 'Aston Martin', 'Audi', 'BMW', 'Bentley', 'Buick',
       'Cadillac', 'Chevrolet', 'Chrysler', 'Daewoo', 'Dodge', 'FIAT',
       'Ferrari', 'Fisker', 'Ford', 'GMC', 'Geo', 'HUMMER', 'Honda',
       'Hyundai', 'Infiniti', 'Isuzu', 'Jaguar', 'Jeep', 'Kia',
       'Lamborghini', 'Land Rover', 'Lexus', 'Lincoln', 'Lotus', 'MINI',
       'Maserati', 'Mazda', 'Mercedes-Benz', 'Mercury', 'Mitsubishi',
       'Nissan', 'Oldsmobile', 'Plymouth', 'Pontiac', 'Porsche', 'Ram',
       'Rolls-Royce', 'Saab', 'Saturn', 'Scion', 'Subaru', 'Suzuki',
       'Tesla', 'Toyota', 'Volkswagen', 'Volvo', 'acura', 'airstream',
       'audi', 'bmw', 'buick', 'cadillac', 'chev truck', 'chevrolet',
       'chrysler', 'dodge', 'dodge tk', 'dot', 'ford', 'ford tk',
       'ford truck', 'gmc', 'gmc truck', 'honda', 'hyundai', 'hyundai tk',
       'jeep', 'kia', 'land rover', 'landrover', 'lexus', 'lincoln',
       'maserati', 'mazda', 'mazda tk', 'mercedes', 'mercedes-b',
       'mercury', 'mitsubishi', 'nissan', 'oldsmobile', 'plymouth',
       'pontiac', 'porsche', 'smart', 'subaru', 'suzuki', 'toyota',
       'volkswagen', 'vw'], dtype=object)
```

```
# Импьютация константой
imp3 = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value='NA')
data_imp3 = imp3.fit_transform(cat_temp_data)
data_imp3
```

```
array([['Kia'],
       ['Kia'],
       ['BMW'],
       ...,
       ['Toyota'],
       ['Toyota'],
       ['Toyota']], dtype=object)
```

```
[ ] np.unique(data_imp3)
```

```
array(['Acura', 'Aston Martin', 'Audi', 'BMW', 'Bentley', 'Buick',
       'Cadillac', 'Chevrolet', 'Chrysler', 'Daewoo', 'Dodge', 'FIAT',
       'Ferrari', 'Fisker', 'Ford', 'GMC', 'Geo', 'HUMMER', 'Honda',
       'Hyundai', 'Infiniti', 'Isuzu', 'Jaguar', 'Jeep', 'Kia',
       'Lamborghini', 'Land Rover', 'Lexus', 'Lincoln', 'Lotus', 'MINI',
       'Maserati', 'Mazda', 'Mercedes-Benz', 'Mercury', 'Mitsubishi',
       'NA', 'Nissan', 'Oldsmobile', 'Plymouth', 'Pontiac', 'Porsche',
       'Ram', 'Rolls-Royce', 'Saab', 'Saturn', 'Scion', 'Subaru',
       'Suzuki', 'Tesla', 'Toyota', 'Volkswagen', 'Volvo', 'acura',
       'airstream', 'audi', 'bmw', 'buick', 'cadillac', 'chev truck',
       'chevrolet', 'chrysler', 'dodge', 'dodge tk', 'dot', 'ford',
       'ford tk', 'ford truck', 'gmc', 'gmc truck', 'honda', 'hyundai',
       'hyundai tk', 'jeep', 'kia', 'land rover', 'landrover', 'lexus',
       'lincoln', 'maserati', 'mazda', 'mazda tk', 'mercedes',
       'mercedes-b', 'mercury', 'mitsubishi', 'nissan', 'oldsmobile',
       'plymouth', 'pontiac', 'porsche', 'smart', 'subaru', 'suzuki',
       'toyota', 'volkswagen', 'vw'], dtype=object)
```

```
[ ] data_imp3[data_imp3=='NA'].size
```

```
8202
```

```
from sklearn.impute import SimpleImputer

# Создание экземпляра SimpleImputer с параметром strategy='most_frequent'
imputer = SimpleImputer(strategy='most_frequent')

# Преобразование данных с заменой пропущенных значений для каждого столбца
imputed_data = imputer.fit_transform(data_cat)

# Преобразованные данные в DataFrame
df_imputed = pd.DataFrame(imputed_data, columns=data_cat.columns)

df_imputed
```

	make	model	trim	body	transmission	color	interior	saledate
0	Kia	Sorento	LX	SUV	automatic	white	black	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
1	Kia	Sorento	LX	SUV	automatic	white	beige	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2	BMW	3 Series	328i SULEV	Sedan	automatic	gray	black	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
3	Volvo	S60	T5	Sedan	automatic	white	black	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
4	BMW	6 Series Gran Coupe	650i	Sedan	automatic	gray	black	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)
...	...	...	...	...	...	...	...	...
458978	Subaru	Outback	2.5i Premium	Wagon	automatic	green	tan	Wed May 27 2015 03:30:00 GMT-0700 (PDT)
458979	Toyota	Camry	LE	Sedan	automatic	silver	gray	Tue May 26 2015 06:00:00 GMT-0700 (PDT)
458980	Toyota	Camry Hybrid	XLE	Sedan	automatic	red	gray	Wed May 27 2015 02:30:00 GMT-0700 (PDT)
458981	Toyota	Highlander	Base	SUV	automatic	gray	black	Wed May 27 2015 02:00:00 GMT-0700 (PDT)
458982	Toyota	Camry	SE	Sedan	automatic	blue	black	Tue Feb 10 2015 01:30:00 GMT-0800 (PST)

458983 rows × 8 columns

```
[ ] clean_n_data.shape
```

```
(458983, 16)
```

```
[ ] df_imputed.shape
```

```
(458983, 8)
```

```
df_imputed.isnull().sum()
```

```
make            0
model           0
trim            0
body            0
transmission    0
color           0
interior        0
saledate        0
dtype: int64
```

```
[ ] clean_n_data.isnull().sum()
```

```
year                0
make             8202
model            8274
trim             8531
body            11075
transmission    52143
vin                 0
state               0
condition           0
odometer            0
color             643
interior          643
seller              0
mmr                 0
sellingprice        0
saledate           10
dtype: int64
```

```
# Замена столбцов 'b' и 'c' в pd1 на столбцы 'b' и 'c' из pd2
clean_n_data.update(df_imputed)

clean_n_data
```

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	mmr	sellingprice	saledate
0	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	5.0	16639.0	white	black	kia motors america inc	20500.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
1	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg561319	ca	5.0	9393.0	white	beige	kia motors america inc	20800.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2	2014	BMW	3 Series	328i SULEV	Sedan	automatic	wba3c1c51ek116351	ca	45.0	1331.0	gray	black	financial services remarketing (lease)	31900.0	30000.0	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
3	2015	Volvo	S60	T5	Sedan	automatic	yv1612tb4f1310987	ca	41.0	14282.0	white	black	volvo na rep/world omni	27500.0	27750.0	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
4	2014	BMW	6 Series Gran Coupe	650i	Sedan	automatic	wba6b2c57ed129731	ca	43.0	2641.0	gray	black	financial services remarketing (lease)	66000.0	67000.0	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
458978	2012	Subaru	Outback	2.5i Premium	Wagon	automatic	4s4brcgc4c3208123	mn	39.0	28670.0	green	tan	jpmorgan chase bank n.a.	19450.0	20300.0	Wed May 27 2015 03:30:00 GMT-0700 (PDT)
458979	2012	Toyota	Camry	LE	Sedan	automatic	4t1bf1fk9cu575017	ga	37.0	75525.0	silver	gray	nextgear capital	10700.0	11000.0	Tue May 26 2015 06:00:00 GMT-0700 (PDT)
458980	2012	Toyota	Camry Hybrid	XLE	Sedan	automatic	4t1bd1fk1cu016762	nj	35.0	90725.0	red	gray	marano & sons auto sales inc	14700.0	12687.0	Wed May 27 2015 02:30:00 GMT-0700 (PDT)
458981	2012	Toyota	Highlander	Base	SUV	automatic	5tdbk3eh3cs134716	nj	31.0	32808.0	gray	black	leaserite leasing and sales	23700.0	23800.0	Wed May 27 2015 02:00:00 GMT-0700 (PDT)
458982	2012	Toyota	Camry	SE	Sedan	automatic	4t1bf1fk4cu158520	ga	2.0	169959.0	blue	black	nalley toyota stonecrest	11600.0	10100.0	Tue Feb 10 2015 01:30:00 GMT-0800 (PST)

458983 rows × 16 columns

```
clean_n_data.isnull().sum()
```

```
year            0
make            0
model           0
trim            0
body            0
transmission    0
vin             0
state           0
condition       0
odometer        0
color           0
interior        0
seller          0
mmr             0
sellingprice    0
saledate        0
dtype: int64
```

```
[ ]
```

Преобразование категориальных признаков в числовые

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

	c1
0	Kia
1	Kia
2	BMW
3	Volvo
4	BMW
...	...
458978	Subaru
458979	Toyota
458980	Toyota
458981	Toyota
458982	Toyota

458983 rows × 1 columns

Кодирование категорий целочисленными значениями (label encoding)

В этом случае уникальные значения категориального признака кодируются целыми числами.

В scikit-learn для такого кодирования используется два класса :

- LabelEncoder - который ориентирован на применение к одному признаку. Этот класс прежде всего предназначен для кодирования целевого признака, но может быть также использован для последовательного кодирования отдельных нецелевых признаков.
- OrdinalEncoder - который ориентирован на применение к матрице объект-признак, то есть для кодирования матрицы нецелевых признаков.

Использование LabelEncoder

```
[ ] from sklearn.preprocessing import LabelEncoder
```

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

```
array(['Kia', 'BMW', 'Volvo', 'Nissan', 'Chevrolet', 'Audi', 'Ford',
       'Hyundai', 'Buick', 'Cadillac', 'Acura', 'Lexus', 'Infiniti',
       'Jeep', 'Mercedes-Benz', 'Mitsubishi', 'Mazda', 'MINI',
       'Land Rover', 'Lincoln', 'lincoln', 'Jaguar', 'Volkswagen',
       'Toyota', 'Subaru', 'Scion', 'Porsche', 'bmw', 'Dodge', 'FIAT',
       'Chrysler', 'ford', 'Ferrari', 'Honda', 'GMC', 'mitsubishi', 'Ram',
       'smart', 'chevrolet', 'Bentley', 'chrysler', 'pontiac', 'Pontiac',
       'Saturn', 'Maserati', 'Mercury', 'HUMMER', 'landrover', 'cadillac',
       'land rover', 'mercedes', 'mazda', 'toyota', 'lexus', 'gmc truck',
       'honda', 'nissan', 'porsche', 'Saab', 'Suzuki', 'dodge', 'subaru',
       'Oldsmobile', 'oldsmobile', 'hyundai', 'jeep', 'Isuzu', 'dodge tk',
       'Geo', 'acura', 'volkswagen', 'suzuki', 'kia', 'audi',
       'Rolls-Royce', 'gmc', 'maserati', 'mazda tk', 'mercury', 'buick',
       'hyundai tk', 'mercedes-b', 'vw', 'Daewoo', 'chev truck',
       'ford tk', 'plymouth', 'Plymouth', 'ford truck', 'Tesla',
       'airstream', 'dot', 'Aston Martin', 'Fisker', 'Lamborghini',
       'Lotus'], dtype=object)
```

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

```
[ ] # Наименования категорий в соответствии с порядковыми номерами

    # Свойство называется classes, потому что предполагается что мы решаем
    # задачу классификации и каждое значение категории соответствует
    # какому-либо классу целевого признака

    le.classes_
```

```
array(['Acura', 'Aston Martin', 'Audi', 'BMW', 'Bentley', 'Buick',
       'Cadillac', 'Chevrolet', 'Chrysler', 'Daewoo', 'Dodge', 'FIAT',
       'Ferrari', 'Fisker', 'Ford', 'GMC', 'Geo', 'HUMMER', 'Honda',
       'Hyundai', 'Infiniti', 'Isuzu', 'Jaguar', 'Jeep', 'Kia',
       'Lamborghini', 'Land Rover', 'Lexus', 'Lincoln', 'Lotus', 'MINI',
       'Maserati', 'Mazda', 'Mercedes-Benz', 'Mercury', 'Mitsubishi',
       'Nissan', 'Oldsmobile', 'Plymouth', 'Pontiac', 'Porsche', 'Ram',
       'Rolls-Royce', 'Saab', 'Saturn', 'Scion', 'Subaru', 'Suzuki',
       'Tesla', 'Toyota', 'Volkswagen', 'Volvo', 'acura', 'airstream',
       'audi', 'bmw', 'buick', 'cadillac', 'chev truck', 'chevrolet',
       'chrysler', 'dodge', 'dodge tk', 'dot', 'ford', 'ford tk',
       'ford truck', 'gmc', 'gmc truck', 'honda', 'hyundai', 'hyundai tk',
       'jeep', 'kia', 'land rover', 'landrover', 'lexus', 'lincoln',
       'maserati', 'mazda', 'mazda tk', 'mercedes', 'mercedes-b',
       'mercury', 'mitsubishi', 'nissan', 'oldsmobile', 'plymouth',
       'pontiac', 'porsche', 'smart', 'subaru', 'suzuki', 'toyota',
       'volkswagen', 'vw'], dtype=object)
```

```
[ ] cat_enc_le
```

```
array([24, 24,  3, ..., 49, 49, 49])
```

```
[ ] np.unique(cat_enc_le)
```

```
array([ 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, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95])
```

```
[ ] # В этом примере видно, что перед кодированием
    # уникальные значения признака сортируются в лексикографическом порядке
    le.inverse_transform([0, 1, 2, 3])
```

```
array(['Acura', 'Aston Martin', 'Audi', 'BMW'], dtype=object)
```

Использование OrdinalEncoder

```
[ ] from sklearn.preprocessing import OrdinalEncoder
```

```
data_oe = data[['make', 'model', 'transmission']]
data_oe.head()
```

	make	model	transmission
0	Kia	Sorento	automatic
1	Kia	Sorento	automatic
2	BMW	3 Series	automatic
3	Volvo	S60	automatic
4	BMW	6 Series Gran Coupe	automatic

```
[ ] imp4 = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value='NA')
    data_oe_filled = imp4.fit_transform(data_oe)
    data_oe_filled
```

```
array([['Kia', 'Sorento', 'automatic'],
       ['Kia', 'Sorento', 'automatic'],
       ['BMW', '3 Series', 'automatic'],
       ...,
       ['Toyota', 'Camry Hybrid', 'automatic'],
       ['Toyota', 'Highlander', 'automatic'],
       ['Toyota', 'Camry', 'automatic']], dtype=object)
```

```
[ ] oe = OrdinalEncoder()
    cat_enc_oe = oe.fit_transform(data_oe_filled)
    cat_enc_oe
```

```
array([[ 24., 650.,   2.],
       [ 24., 650.,   2.],
       [  3.,   9.,   2.],
       ...,
       [ 50., 149.,   2.],
       [ 50., 354.,   2.],
       [ 50., 148.,   2.]])
```

```
# Уникальные значения 1 признака
np.unique(cat_enc_oe[:, 0])
```

```
array([ 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., 33., 34., 35., 36., 37., 38.,
       39., 40., 41., 42., 43., 44., 45., 46., 47., 48., 49., 50., 51.,
       52., 53., 54., 55., 56., 57., 58., 59., 60., 61., 62., 63., 64.,
       65., 66., 67., 68., 69., 70., 71., 72., 73., 74., 75., 76., 77.,
       78., 79., 80., 81., 82., 83., 84., 85., 86., 87., 88., 89., 90.,
       91., 92., 93., 94., 95., 96.])
```

```
# Уникальные значения 2 признака
np.unique(cat_enc_oe[:, 1])
```

```
array([  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.,
        33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
        44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
        55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
        66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
        77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
        88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
        99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
       110., 111., 112., 113., 114., 115., 116., 117., 118., 119., 120.,
       121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
       132., 133., 134., 135., 136., 137., 138., 139., 140., 141., 142.,
       143., 144., 145., 146., 147., 148., 149., 150., 151., 152., 153.,
       154., 155., 156., 157., 158., 159., 160., 161., 162., 163., 164.,
       165., 166., 167., 168., 169., 170., 171., 172., 173., 174., 175.,
       176., 177., 178., 179., 180., 181., 182., 183., 184., 185., 186.,
       187., 188., 189., 190., 191., 192., 193., 194., 195., 196., 197.,
       198., 199., 200., 201., 202., 203., 204., 205., 206., 207., 208.,
       209., 210., 211., 212., 213., 214., 215., 216., 217., 218., 219.,
       220., 221., 222., 223., 224., 225., 226., 227., 228., 229., 230.,
       231., 232., 233., 234., 235., 236., 237., 238., 239., 240., 241.,
       242., 243., 244., 245., 246., 247., 248., 249., 250., 251., 252.,
       253., 254., 255., 256., 257., 258., 259., 260., 261., 262., 263.,
       264., 265., 266., 267., 268., 269., 270., 271., 272., 273., 274.,
       275., 276., 277., 278., 279., 280., 281., 282., 283., 284., 285.,
       286., 287., 288., 289., 290., 291., 292., 293., 294., 295., 296.,
       297., 298., 299., 300., 301., 302., 303., 304., 305., 306., 307.,
       308., 309., 310., 311., 312., 313., 314., 315., 316., 317., 318.,
       319., 320., 321., 322., 323., 324., 325., 326., 327., 328., 329.,
       330., 331., 332., 333., 334., 335., 336., 337., 338., 339., 340.,
       341., 342., 343., 344., 345., 346., 347., 348., 349., 350., 351.,
       352., 353., 354., 355., 356., 357., 358., 359., 360., 361., 362.,
       363., 364., 365., 366., 367., 368., 369., 370., 371., 372., 373.,
       374., 375., 376., 377., 378., 379., 380., 381., 382., 383., 384.,
       385., 386., 387., 388., 389., 390., 391., 392., 393., 394., 395.,
```

```
[ ] # Уникальные значения 3 признака
    np.unique(cat_enc_oe[:, 2])
```

```
array([0., 1., 2., 3.])
```

```
# Наименования категорий в соответствии с порядковыми номерами
oe.categories_
```

```
[array(['Acura', 'Aston Martin', 'Audi', 'BMW', 'Bentley', 'Buick',
        'Cadillac', 'Chevrolet', 'Chrysler', 'Daewoo', 'Dodge', 'FIAT',
        'Ferrari', 'Fisker', 'Ford', 'GMC', 'Geo', 'HUMMER', 'Honda',
        'Hyundai', 'Infiniti', 'Isuzu', 'Jaguar', 'Jeep', 'Kia',
        'Lamborghini', 'Land Rover', 'Lexus', 'Lincoln', 'Lotus', 'MINI',
        'Maserati', 'Mazda', 'Mercedes-Benz', 'Mercury', 'Mitsubishi',
        'NA', 'Nissan', 'Oldsmobile', 'Plymouth', 'Pontiac', 'Porsche',
        'Ram', 'Rolls-Royce', 'Saab', 'Saturn', 'Scion', 'Subaru',
        'Suzuki', 'Tesla', 'Toyota', 'Volkswagen', 'Volvo', 'acura',
        'airstream', 'audi', 'bmw', 'buick', 'cadillac', 'chev truck',
        'chevrolet', 'chrysler', 'dodge', 'dodge tk', 'dot', 'ford',
        'ford tk', 'ford truck', 'gmc', 'gmc truck', 'honda', 'hyundai',
        'hyundai tk', 'jeep', 'kia', 'land rover', 'landrover', 'lexus',
        'lincoln', 'maserati', 'mazda', 'mazda tk', 'mercedes',
        'mercedes-b', 'mercury', 'mitsubishi', 'nissan', 'oldsmobile',
        'plymouth', 'pontiac', 'porsche', 'smart', 'subaru', 'suzuki',
        'toyota', 'volkswagen', 'vw'], dtype=object),
 array(['1', '1 Series', '1500', '190-Class', '2 Series', '200', '200SX',
        '2500', '3', '3 Series', '3 Series Gran Turismo', '300',
        '300-Class', '3000GT', '300M', '300ZX', '300e', '320i', '323i',
        '328i', '350', '3500', '350Z', '350z', '370Z', '4 Series',
        '400-Class', '420-Class', '420sel', '42c', '4Runner', '5 Series',
        '5 Series Gran Turismo', '500', '500-Class', '500L', '500e', '6',
        '6 Series', '6 Series Gran Coupe', '626', '7', '7 Series', '750i',
        '750li', '750lxi', '8 Series', '850', '9-2X', '9-3', '9-5', '9-7X',
        '911', '940', '960', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'ATS',
        'Acadia', 'Accent', 'Accord', 'Accord Crosstour', 'Accord Hybrid',
        'Achieva', 'ActiveHybrid 5', 'ActiveHybrid 7', 'ActiveHybrid X6',
        'Aerio', 'Alero', 'Altima', 'Altima Hybrid', 'Amanti', 'Amigo',
        'Armada', 'Ascender', 'Aspen', 'Aspire', 'Astra', 'Astro',
        'Astro Cargo', 'Aura', 'Aura Hybrid', 'Aurora', 'Avalanche',
        'Avalon', 'Avalon Hybrid', 'Avenger', 'Aveo', 'Aviator', 'Axiom',
        'Azera', 'Aztek', 'B-Class Electric Drive', 'B-Series',
        'B-Series Pickup', 'B-Series Truck', 'B9 Tribeca', 'BRZ', 'Baja',
        'Beetle', 'Beetle Convertible', 'Black Diamond Avalanche',
        'Blackwood', 'Blazer', 'Bonneville', 'Borrego', 'Boxster',
        'Bravada', 'Breeze', 'C-Class', 'C-Max Energi', 'C-Max Hybrid',
        'C/K 1500 Series', 'C/K 2500 Series', 'C/K 3500 Series',
        'C/V Cargo Van', 'C/V Tradesman', 'C30', 'C70', 'CC', 'CL',
        'CL-Class', 'CLA-Class', 'CLK-Class', 'CLS-Class', 'CR-V', 'CR-Z',
        'CT 200h', 'CTS', 'CTS Coupe', 'CTS Wagon', 'CTS-V', 'CTS-V Coupe',
        'CTS-V Wagon', 'CV Tradesman', 'CX-5', 'CX-7', 'CX-9', 'Cabrio',
        'Cabriolet', 'Cadenza', 'Caliber', 'California', 'Camaro', 'Camry',
        'Camry Hybrid', 'Camry Solara', 'Canyon', 'Caprice',
        'Captiva Sport', 'Caravan', 'Catera', 'Cavalier', 'Cayenne',
        'Cayman', 'Cayman S', 'Celica', 'Century', 'Challenger', 'Charger',
        'Cherokee', 'Cirrus', 'Civic', 'Civic del Sol', 'Classic',
        'Cobalt', 'Colorado', 'Commander', 'Compass', 'Concorde',
        'Continental', 'Continental Flying Spur',
        'Continental Flying Spur Speed', 'Continental GT',
        'Continental GT Speed', 'Continental GTC', 'Continental GTC Speed',
        'Continental Supersports', 'Contour', 'Cooper', 'Cooper Clubman',
```

```
[ ] # Обратное преобразование
    oe.inverse_transform(cat_enc_oe)
```

```
array([['Kia', 'Sorento', 'automatic'],
       ['Kia', 'Sorento', 'automatic'],
       ['BMW', '3 Series', 'automatic'],
       ...,
       ['Toyota', 'Camry Hybrid', 'automatic'],
       ['Toyota', 'Highlander', 'automatic'],
       ['Toyota', 'Camry', 'automatic']], dtype=object)
```

Проблемы использования LabelEncoder и OrdinalEncoder

Необходимо отметить, что LabelEncoder и OrdinalEncoder могут использоваться только для категориальных признаков в номинальных шкалах (для которых отсутствует порядок), например города, страны, названия рек и т.д.

Это связано с тем, что задать какой-либо порядок при кодировании с помощью LabelEncoder и OrdinalEncoder невозможно, они сортируют категории в лексикографическом порядке.

При этом кодирование целыми числами создает фиктивное отношение порядка (1 < 2 < 3 < ...) которого не было в исходных номинальных шкалах. Данное отношение порядка может негативно повлиять на построение модели машинного обучения.

Кодирование шкал порядка

Библиотека scikit-learn не предоставляет готового решения для кодирования шкал порядка, но можно воспользоваться функцией map для отдельных объектов Series.

```
[ ] # пример шкалы порядка 'small' < 'medium' < 'large'
    sizes = ['small', 'medium', 'large', 'small', 'medium', 'large', 'small', 'medium', 'large']
```

```
pd_sizes = pd.DataFrame(data={'sizes':sizes})
pd_sizes
```

	sizes
0	small
1	medium
2	large
3	small
4	medium
5	large
6	small
7	medium
8	large

```
[ ] pd_sizes['sizes_codes'] = pd_sizes['sizes'].map({'small':1, 'medium':2, 'large':3})
    pd_sizes
```

	sizes	sizes_codes
0	small	1
1	medium	2
2	large	3
3	small	1
4	medium	2
5	large	3
6	small	1
7	medium	2
8	large	3

```
pd_sizes['sizes_decoded'] = pd_sizes['sizes_codes'].map({1:'small', 2:'medium', 3:'large'})
pd_sizes
```

	sizes	sizes_codes	sizes_decoded
0	small	1	small
1	medium	2	medium
2	large	3	large
3	small	1	small
4	medium	2	medium
5	large	3	large
6	small	1	small
7	medium	2	medium
8	large	3	large

Кодирование категорий наборами бинарных значений - one-hot encoding

В этом случае каждое уникальное значение признака становится новым отдельным признаком.

```
[ ] from sklearn.preprocessing import OneHotEncoder
```

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

```
[ ] cat_enc.shape
```

```
(458983, 1)
```

```
[ ] cat_enc_ohe.shape
```

```
(458983, 96)
```

```
[ ] cat_enc_ohe
```

```
<458983x96 sparse matrix of type '<class 'numpy.float64'>'
        with 458983 stored elements in Compressed Sparse Row format>
```

```
cat_enc_ohe.todense()[0:10]
```

```
matrix([[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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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., 0., 0., 0., 0., 0., 0., 0., 0.,
```

```
[ ] cat_enc.head(10)
```

	c1
0	Kia
1	Kia
2	BMW
3	Volvo
4	BMW
5	Nissan
6	BMW
7	Chevrolet
8	Audi
9	Chevrolet

Pandas get_dummies - быстрый вариант one-hot кодирования

```
pd.get_dummies(cat_enc).head()
```

	c1_Acura	c1_Aston Martin	c1_Audi	c1_BMW	c1_Bentley	c1_Buick	c1_Cadillac	c1_Chevrolet	c1_Chrysler	c1_Daewoo	...	c1_oldsmobile	c1_plymouth	c1_pontiac	c1_porsche	c1_smart	c1_subaru	c1_suzuki	c1_toyota	c1_volkswagen	c1_vw
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
2	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 96 columns

```
[ ] pd.get_dummies(cat_temp_data, dummy_na=True).head()
```

	make_Acura	make_Aston Martin	make_Audi	make_BMW	make_Bentley	make_Buick	make_Cadillac	make_Chevrolet	make_Chrysler	make_Daewoo	...	make_plymouth	make_pontiac	make_porsche	make_smart	make_subaru	make_suzuki	make_toyota	make_volkswagen	make_vw	make_nan
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
2	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 97 columns

```
one_hot_cat_data = pd.get_dummies(clean_n_data, columns = ['make', 'model', 'trim', 'body', 'transmission', 'color', 'interior'], dummy_na=True)
one_hot_cat_data.head()
```

	year	vin	state	condition	odometer	seller	mmr	sellingprice	saledate	make_Acura	...	interior_off-white	interior_orange	interior_purple	interior_red	interior_silver	interior_tan	interior_white	interior_yellow	interior_	interior_nan
0	2015	5xyktca69fg566472	ca	5.0	16639.0	kia motors america inc	20500.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)	0	...	0	0	0	0	0	0	0	0	0	0
1	2015	5xyktca69fg561319	ca	5.0	9393.0	kia motors america inc	20800.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)	0	...	0	0	0	0	0	0	0	0	0	0
2	2014	wba3c1c51ek116351	ca	45.0	1331.0	financial services remarketing (lease)	31900.0	30000.0	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)	0	...	0	0	0	0	0	0	0	0	0	0
3	2015	yv1612tb4f1310987	ca	41.0	14282.0	volvo na rep/world omni	27500.0	27750.0	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)	0	...	0	0	0	0	0	0	0	0	0	0
4	2014	wba6b2c57ed129731	ca	43.0	2641.0	financial services remarketing (lease)	66000.0	67000.0	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 3095 columns

+ Код + Текст

```
[ ] one_hot_cat_data.shape
```

```
(458983, 3095)
```

Масштабирование данных

Термины "масштабирование" и "нормализация" часто используются как синонимы, но это неверно. Масштабирование предполагает изменение диапазона измерения величины, а нормализация - изменение распределения этой величины. В этом разделе рассматривается только масштабирование.

Если признаки лежат в различных диапазонах, то необходимо их нормализовать. Как правило, применяют два подхода:

- MinMax масштабирование:

$$x_{новый} = \frac{x_{старый} - min(X)}{max(X) - min(X)}$$

В этом случае значения лежат в диапазоне от 0 до 1.

- Масштабирование данных на основе Z-оценки:

$$x_{новый} = \frac{x_{старый} - AVG(X)}{\sigma(X)}$$

В этом случае большинство значений попадает в диапазон от -3 до 3.

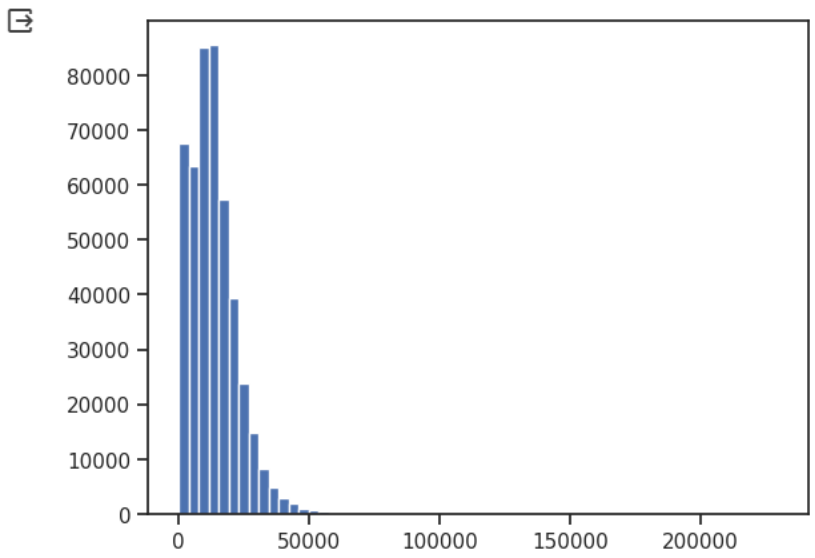
где X - матрица объект-признак, $AVG(X)$ - среднее значение, σ - среднеквадратичное отклонение.

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

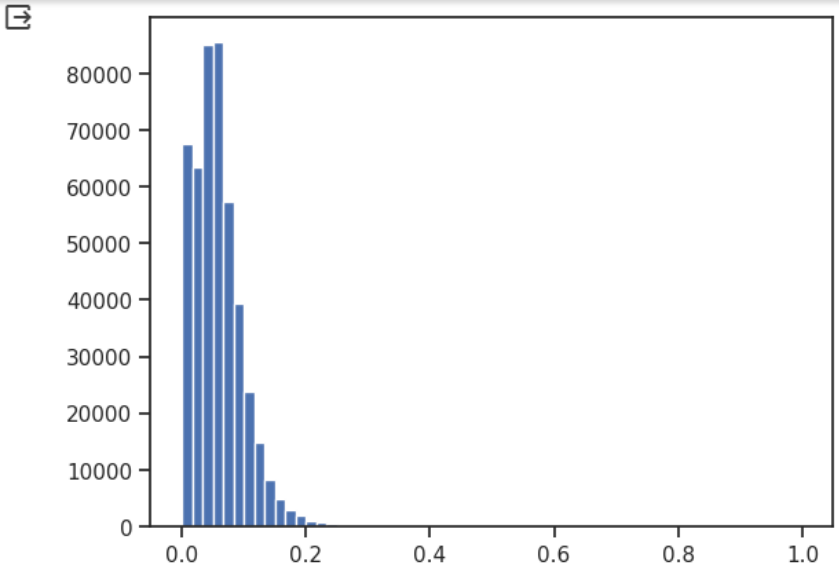
MinMax масштабирование

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

```
     plt.hist(data['sellingprice'], 60)
     plt.show()
```



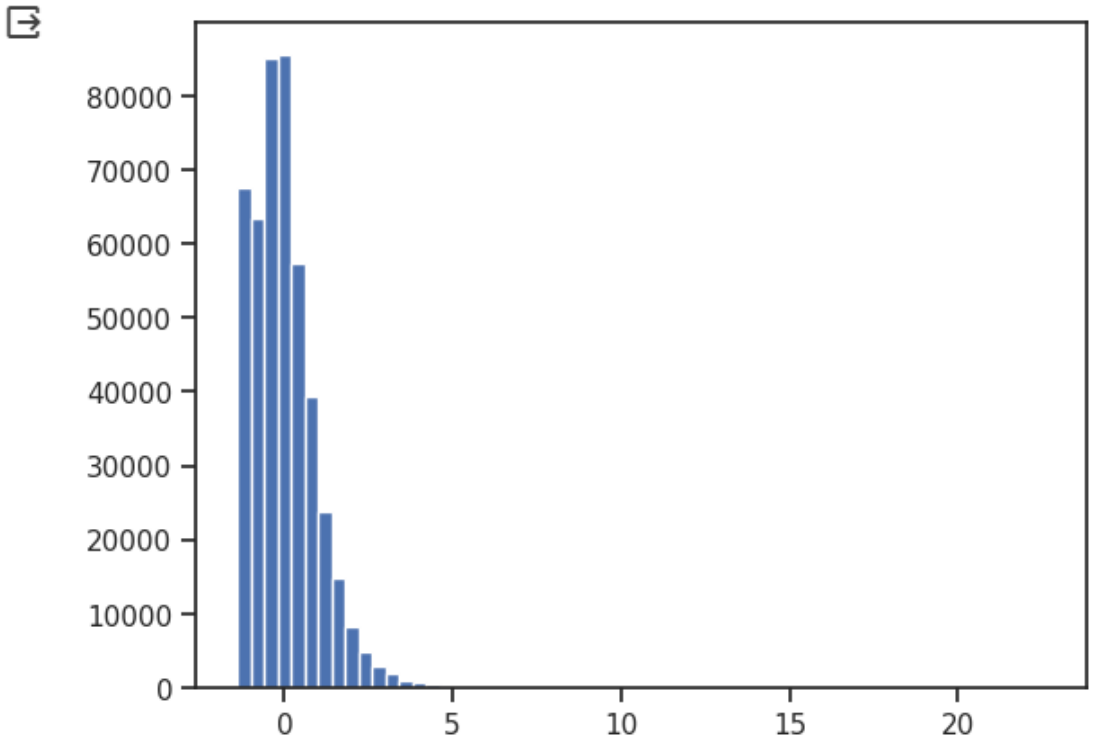

```
[ ]  plt.hist(sc1_data, 60)
     plt.show()
```



Масштабирование данных на основе Z-оценки - StandardScaler

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

```
[ ]  plt.hist(sc2_data, 60)
     plt.show()
```


80000
70000
60000
50000
40000
30000
20000
10000
0
0
5
10
15
20