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**

О Т Ч Е Т
по лабораторной работе № 2

Дисциплина: Технологии машинного обучения
Тема: «Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных»

Студент	ИУ5Ц-82Б		А.Н. Свинцов
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Ю.Е. Гапанюк
		(Подпись, дата)	(И.О. Фамилия)
Преподаватель			А.Н. Нардид
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2023

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.

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

# Размер набора данных
data.shape
```

```
(1460, 81)
```

```
# Типы данных
data.dtypes
```

```
Id                int64
MSSubClass        int64
MSZoning         object
LotFrontage     float64
LotArea           int64
                  ...
MoSold            int64
```

```
YrSold             int64
SaleType          object
SaleCondition     object
SalePrice          int64
Length: 81, dtype: object
```

```
# Проверка на пропущенные данные
data.isnull().sum()
```

```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      259
LotArea            0
                ...
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
SalePrice          0
Length: 81, dtype: int64
```

```
# Первые 5 строк
data.head()
```

```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape
\
0   1          60       RL         65.0     8450   Pave   NaN      Reg

1   2          20       RL         80.0     9600   Pave   NaN      Reg

2   3          60       RL         68.0    11250   Pave   NaN      IR1

3   4          70       RL         60.0     9550   Pave   NaN      IR1

4   5          60       RL         84.0    14260   Pave   NaN      IR1


  LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscVal
MoSold  \
0         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
1         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
5
2         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
9
3         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
4         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
12
```

```
  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]
```

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

```
Всего строк: 1460
```

Обработка пропусков данных

Простые стратегии удаления

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

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

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)
```

```
((1460, 81), (1460, 62))
```

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

```
((1460, 81), (0, 81))
```

```
data.head()
```

```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape
\
0   1          60       RL         65.0     8450   Pave   NaN      Reg

1   2          20       RL         80.0     9600   Pave   NaN      Reg

2   3          60       RL         68.0    11250   Pave   NaN      IR1
```

```
3   4          70       RL         60.0     9550   Pave   NaN      IR1

4   5          60       RL         84.0    14260   Pave   NaN      IR1


  LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscVal
MoSold  \
0         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
1         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
5
2         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
9
3         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
4         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]
```

```
# 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()
```

```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape
\
0   1          60       RL         65.0     8450   Pave     0      Reg

1   2          20       RL         80.0     9600   Pave     0      Reg

2   3          60       RL         68.0    11250   Pave     0      IR1

3   4          70       RL         60.0     9550   Pave     0      IR1

4   5          60       RL         84.0    14260   Pave     0      IR1


  LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscVal
MoSold  \
0         Lvl    AllPub  ...        0      0     0           0       0
```

```
2
1         Lvl    AllPub  ...        0      0     0           0       0
5
2         Lvl    AllPub  ...        0      0     0           0       0
9
3         Lvl    AllPub  ...        0      0     0           0       0
2
4         Lvl    AllPub  ...        0      0     0           0       0
12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]
```

"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))
```

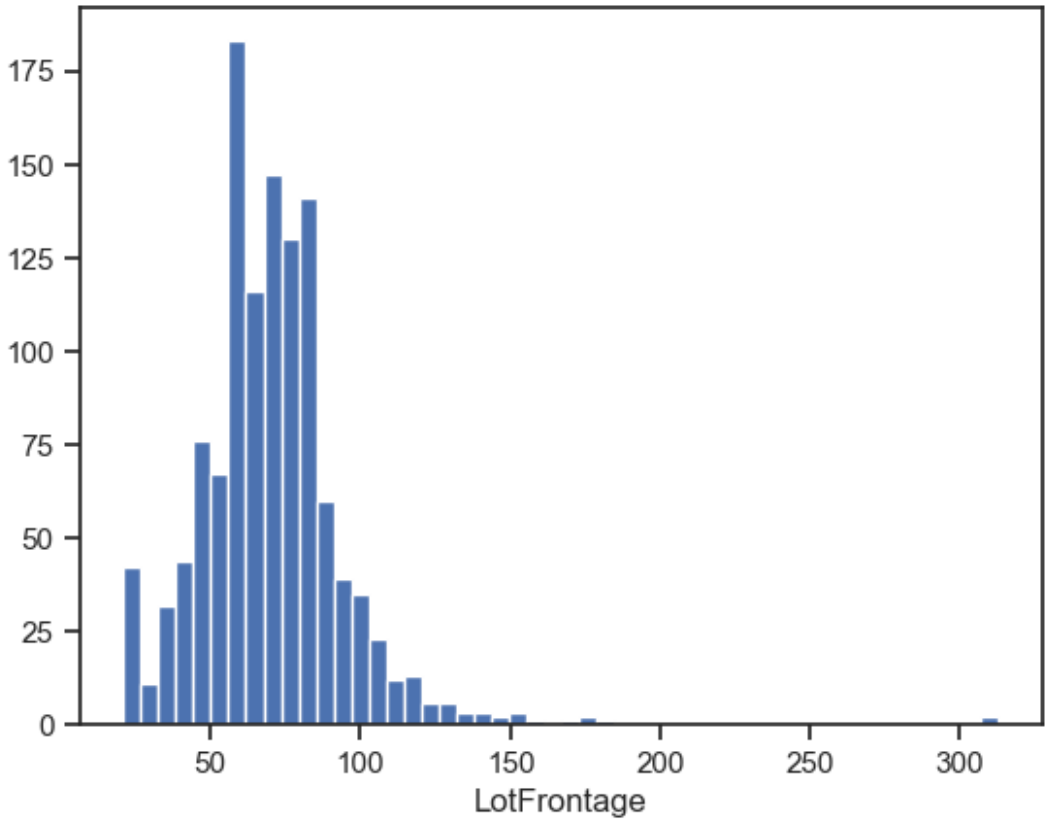
```
Колонка LotFrontage. Тип данных float64. Количество пустых значений
259, 17.74%.
Колонка MasVnrArea. Тип данных float64. Количество пустых значений 8,
0.55%.
Колонка GarageYrBlt. Тип данных float64. Количество пустых значений
81, 5.55%.
```

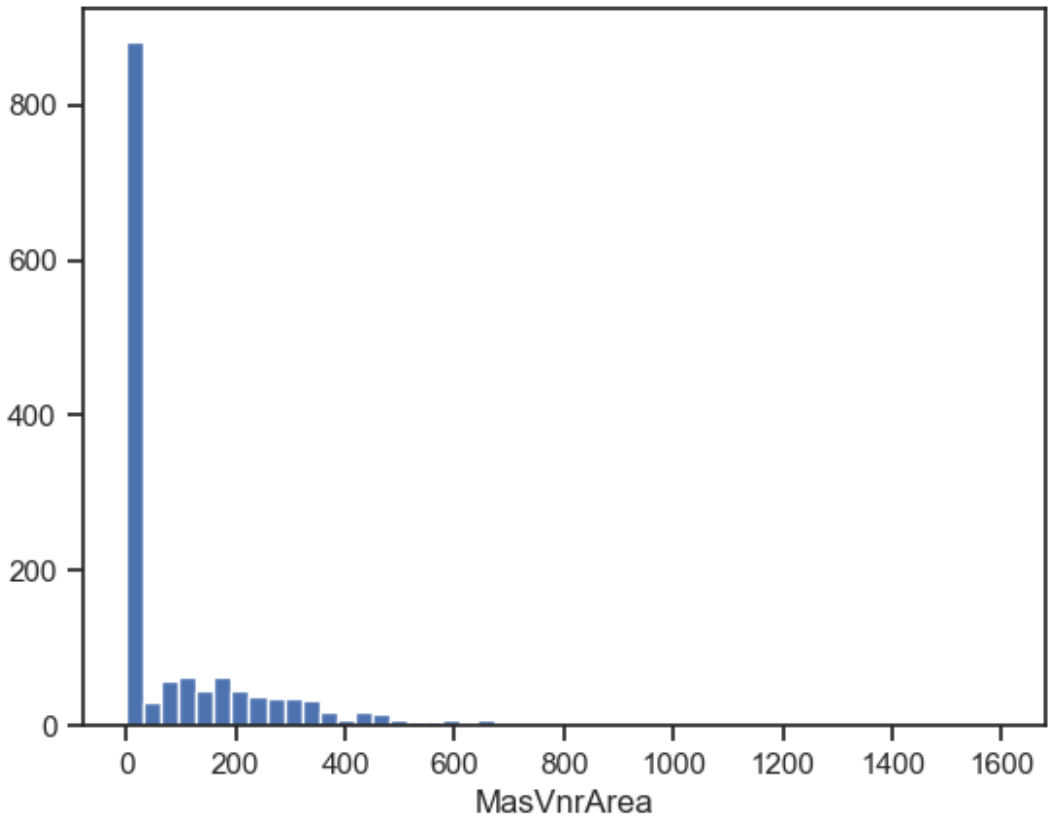
```
# Фильтр по колонкам с пропущенными значениями
data_num = data[num_cols]
data_num
```

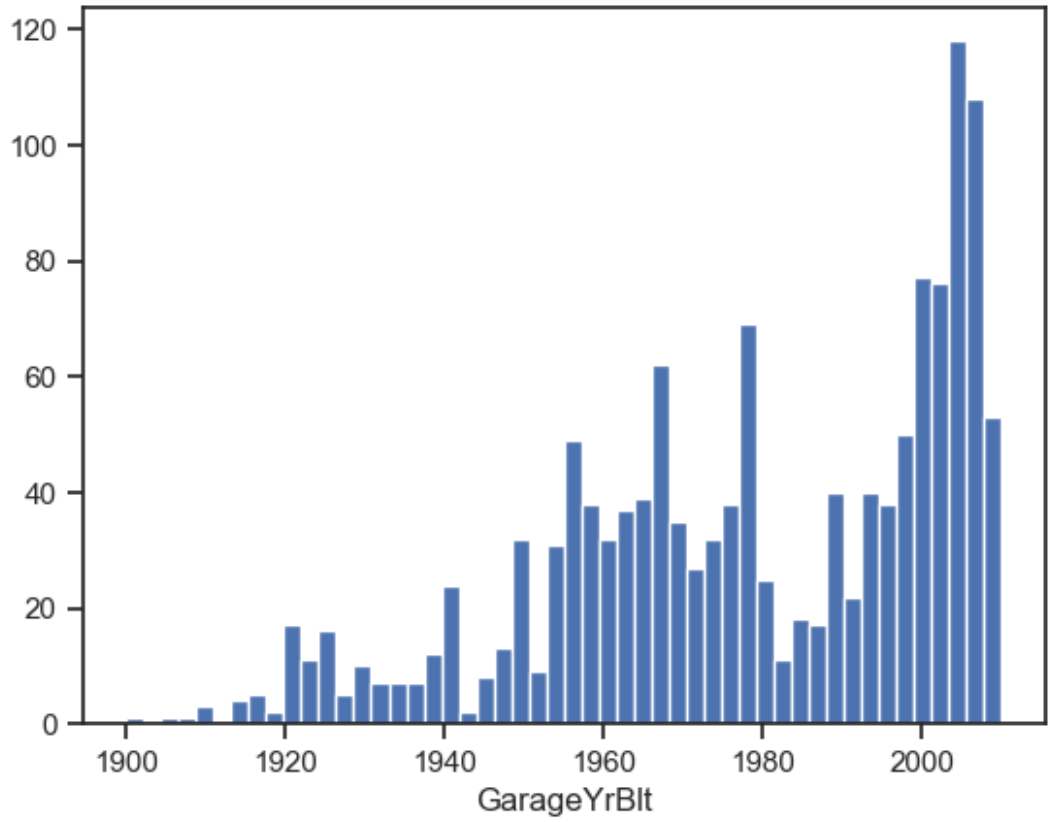
```
      LotFrontage  MasVnrArea  GarageYrBlt
0            65.0       196.0       2003.0
1            80.0         0.0       1976.0
```

```
2            68.0       162.0       2001.0
3            60.0         0.0       1998.0
4            84.0       350.0       2000.0
...           ...         ...          ...
1455         62.0         0.0       1999.0
1456         85.0       119.0       1978.0
1457         66.0         0.0       1941.0
1458         68.0         0.0       1950.0
1459         75.0         0.0       1965.0

[1460 rows x 3 columns]
```

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




800
600
400
200
0
0
200
400
600
800
1000
1200
1400
1600
MasVnrArea


120
100
80
60
40
20
0
1900
1920
1940
1960
1980
2000
GarageYrBlt

```
data_num_MasVnrArea = data_num[['MasVnrArea']]
data_num_MasVnrArea.head()
```

```
   MasVnrArea
0       196.0
1         0.0
2       162.0
3         0.0
4       350.0
```

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

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

```
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

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

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

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

```
('mean',
 array([103.68526171, 103.68526171, 103.68526171, 103.68526171,
        103.68526171, 103.68526171, 103.68526171, 103.68526171]))
```

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

('median', array([0., 0., 0., 0., 0., 0., 0., 0.]))

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

('most_frequent', array([0., 0., 0., 0., 0., 0., 0., 0.]))

# 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[['GarageYrBlt']].describe()
```

```
       GarageYrBlt
count  1379.000000
mean   1978.506164
std      24.689725
min    1900.000000
25%    1961.000000
50%    1980.000000
75%    2002.000000
max    2010.000000
```

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

```
('GarageYrBlt', 'mean', 81, 1978.5061638868744, 1978.5061638868744)
```

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

```
('GarageYrBlt', 'median', 81, 1980.0, 1980.0)
```

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

```
('GarageYrBlt', 'most_frequent', 81, 2005.0, 2005.0)
```

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 Alley. Тип данных object. Количество пустых значений 1369,
93.77%.
Колонка MasVnrType. Тип данных object. Количество пустых значений 8,
```

```
0.55%.
Колонка BsmtQual. Тип данных object. Количество пустых значений 37,
2.53%.
Колонка BsmtCond. Тип данных object. Количество пустых значений 37,
2.53%.
Колонка BsmtExposure. Тип данных object. Количество пустых значений
38, 2.6%.
Колонка BsmtFinType1. Тип данных object. Количество пустых значений
37, 2.53%.
Колонка BsmtFinType2. Тип данных object. Количество пустых значений
38, 2.6%.
Колонка Electrical. Тип данных object. Количество пустых значений 1,
0.07%.
Колонка FireplaceQu. Тип данных object. Количество пустых значений
690, 47.26%.
Колонка GarageType. Тип данных object. Количество пустых значений 81,
5.55%.
Колонка GarageFinish. Тип данных object. Количество пустых значений
81, 5.55%.
Колонка GarageQual. Тип данных object. Количество пустых значений 81,
5.55%.
Колонка GarageCond. Тип данных object. Количество пустых значений 81,
5.55%.
Колонка PoolQC. Тип данных object. Количество пустых значений 1453,
99.52%.
Колонка Fence. Тип данных object. Количество пустых значений 1179,
80.75%.
Колонка MiscFeature. Тип данных object. Количество пустых значений
1406, 96.3%.
```

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

```
  MasVnrType
0    BrkFace
1       None
2    BrkFace
3       None
4    BrkFace
```

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

```
array(['BrkFace', 'None', 'Stone', 'BrkCmn', nan], dtype=object)
```

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

```
(8, 1)
```

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

```
array([['BrkFace'],
       ['None'],
       ['BrkFace'],
       ...,
       ['None'],
       ['None'],
       ['None']], dtype=object)
```

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

```
array(['BrkCmn', 'BrkFace', 'None', 'Stone'], 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([['BrkFace'],
       ['None'],
       ['BrkFace'],
       ...,
       ['None'],
       ['None'],
       ['None']], dtype=object)
```

```
np.unique(data_imp3)
```

```
array(['BrkCmn', 'BrkFace', 'NA', 'None', 'Stone'], dtype=object)
```

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

```
8
```

Преобразование категориальных признаков в числовые

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

```
           c1
0     BrkFace
1        None
2     BrkFace
3        None
4     BrkFace
...       ...
1455     None
1456    Stone
1457     None
1458     None
1459     None
```

```
[1460 rows x 1 columns]
```

Кодирование категорий целочисленными значениями (label encoding)

LabelEncoder

```
from sklearn.preprocessing import LabelEncoder
```

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

```
array(['BrkFace', 'None', 'Stone', 'BrkCmn'], 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(['BrkCmn', 'BrkFace', 'None', 'Stone'], dtype=object)
```

```
cat_enc_le
```

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

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

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

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

```
array(['BrkCmn', 'BrkFace', 'None', 'Stone'], dtype=object)
```

OrdinalEncoder

```
from sklearn.preprocessing import OrdinalEncoder
```

```
data_oe = data[['MasVnrType', 'BsmtQual', 'BsmtCond']]
data_oe.head()
```

```
  MasVnrType BsmtQual BsmtCond
0    BrkFace       Gd       TA
1       None       Gd       TA
```

```
2    BrkFace       Gd       TA
3       None       TA       Gd
4    BrkFace       Gd       TA
```

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

```
array([['BrkFace', 'Gd', 'TA'],
       ['None', 'Gd', 'TA'],
       ['BrkFace', 'Gd', 'TA'],
       ...,
       ['None', 'TA', 'Gd'],
       ['None', 'TA', 'TA'],
       ['None', 'TA', 'TA']], dtype=object)
```

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

```
array([[1., 2., 4.],
       [3., 2., 4.],
       [1., 2., 4.],
       ...,
       [3., 4., 1.],
       [3., 4., 4.],
       [3., 4., 4.]])
```

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

```
array([0., 1., 2., 3., 4.])
```

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

```
array([0., 1., 2., 3., 4.])
```

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

```
array([0., 1., 2., 3., 4.])
```

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

```
[array(['BrkCmn', 'BrkFace', 'NA', 'None', 'Stone'], dtype=object),
 array(['Ex', 'Fa', 'Gd', 'NA', 'TA'], dtype=object),
 array(['Fa', 'Gd', 'NA', 'Po', 'TA'], dtype=object)]
```

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

```
array([['BrkFace', 'Gd', 'TA'],
       ['None', 'Gd', 'TA'],
       ['BrkFace', 'Gd', 'TA'],
       ...,
       ['None', 'TA', 'Gd'],
       ['None', 'TA', 'TA'],
       ['None', 'TA', 'TA']], dtype=object)
```

Шкала порядка

```
# 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
```

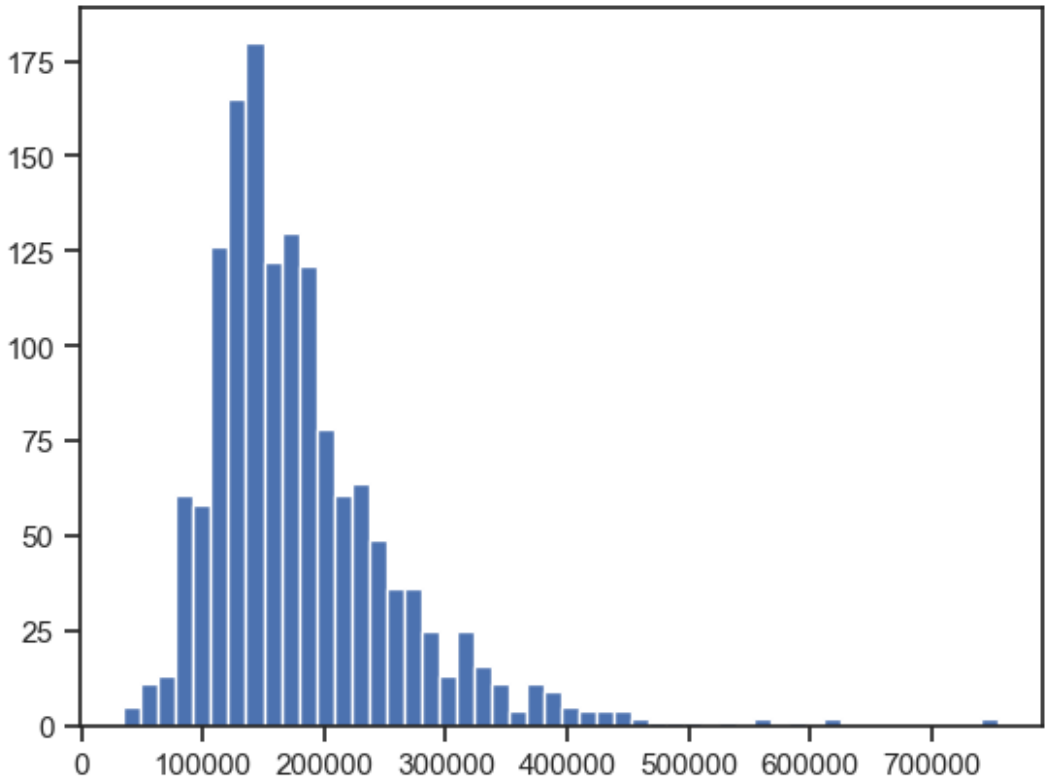
```
4  medium            2        medium
5   large            3         large
6   small            1         small
7  medium            2        medium
8   large            3         large
```

Кодирование категорий наборами бинарных значений

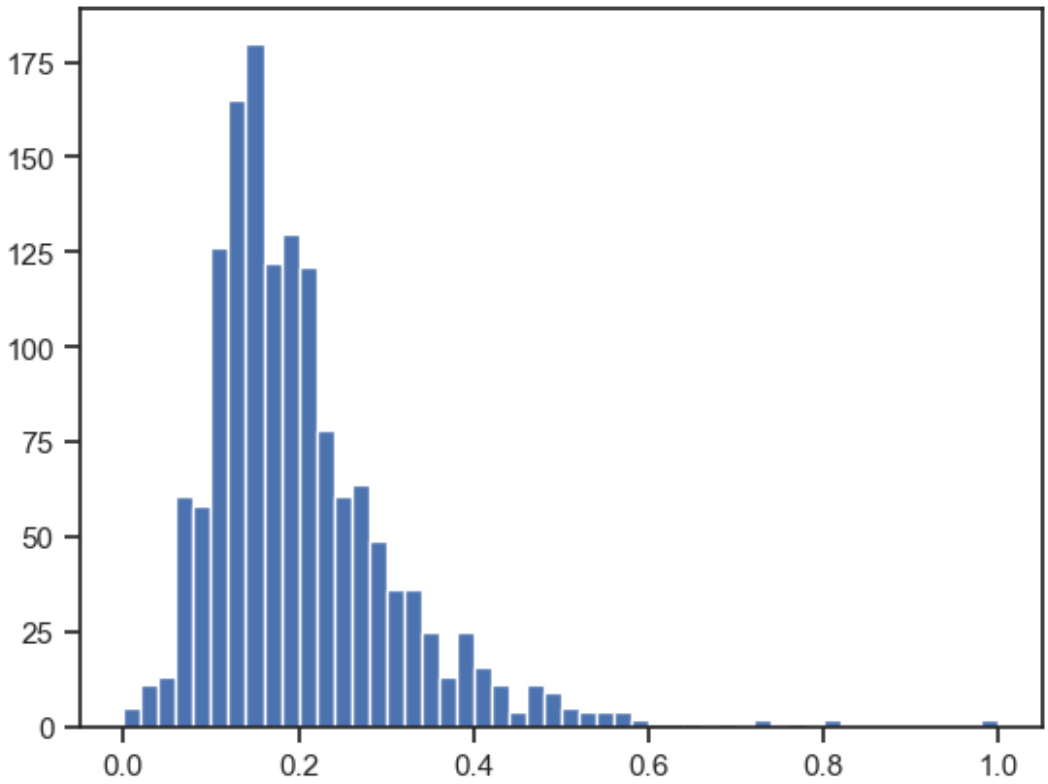
```
from sklearn.preprocessing import OneHotEncoder

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

cat_enc.shape

(1460, 1)

cat_enc_ohe.shape

(1460, 4)

cat_enc_ohe

<1460x4 sparse matrix of type '<class 'numpy.float64'>'
     with 1460 stored elements in Compressed Sparse Row format>

cat_enc_ohe.todense()[0:10]

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

cat_enc.head(10)

        c1
0  BrkFace
1     None
2  BrkFace
3     None
4  BrkFace
5     None
6    Stone
7    Stone
8     None
9     None
```

Масштабирование данных

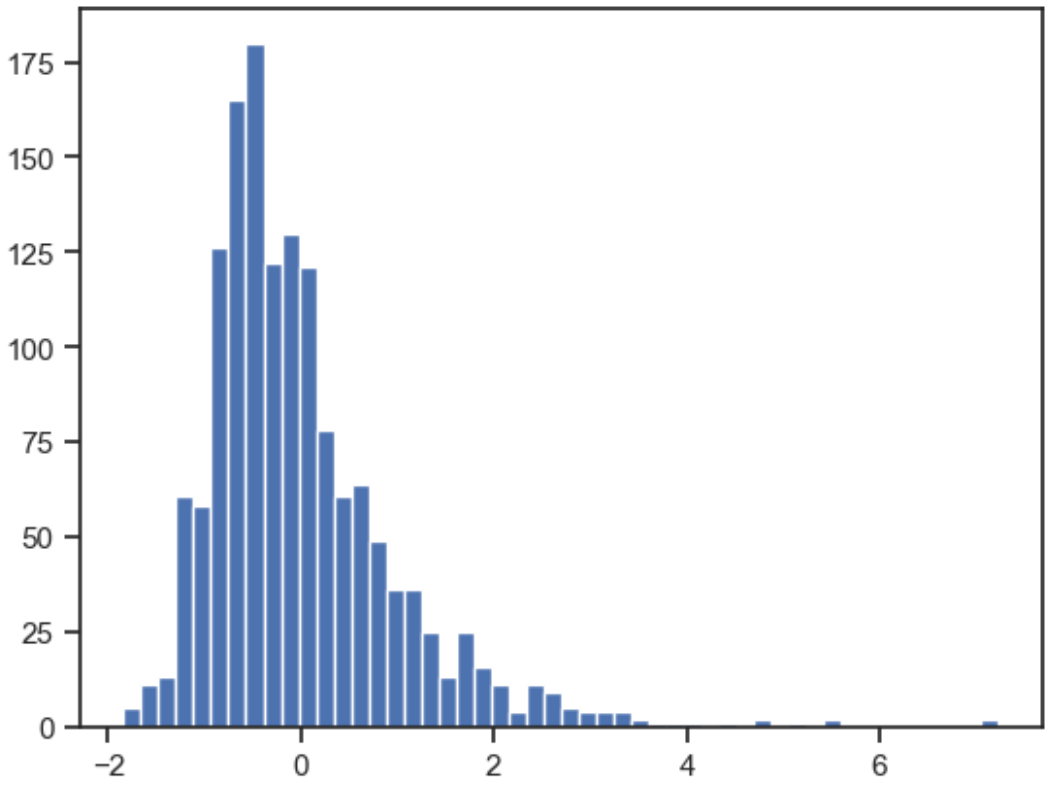
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
Normalizer

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

plt.hist(data['SalePrice'], 50)
plt.show()
```




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




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

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


175
150
125
100
75
50
25
0
−2
0
2
4
6