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

In [1]:

```
import numpy as np
import pandas as pd
pd.set_option('display.max.columns', 100)
# to draw pictures in jupyter notebook
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
# we don't like warnings
# you can comment the following 2 lines if you'd like to
import warnings
warnings.filterwarnings('ignore')
```

Загрузка и первичный анализ данных

In [172]:

```
data = pd.read_csv("C:/Users/VTsapiy/Desktop/лаба3/train.csv", sep=',')
data.head()
```

Out[172]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Ut
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	/
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	/
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	/
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	/
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	/

In [169]:

```
data.shape
```

Out[169]:

(401, 25)

In [164]:

```
data.dtypes
```

Out[164]:

```
age       object
bp        object
sg        object
al        object
su        object
rbc       object
pc        object
pcc       object
ba        object
bgr       object
bu        object
sc        object
sod       object
pot       object
hemo      object
pcv       object
wc        object
rc        object
htn       object
dm        object
cad       object
appet     object
pe        object
ane       object
class     object
dtype: object
```

In [97]:

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

Out[97]:

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

In [98]:

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

```
Всего строк: 1460
```

1. Обработка пропусков в данных

1.1. Простые стратегии - удаление или заполнение нулями

In [99]:

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

Out[99]:

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

In [100]:

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

Out[100]:

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

In [101]:

```
data.head()
```

Out[101]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Ut
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	/
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	/
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	/
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	/
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	/

In [102]:

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

Out[102]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Ut
0	1	60	RL	65.0	8450	Pave	0	Reg	Lvl	/
1	2	20	RL	80.0	9600	Pave	0	Reg	Lvl	/
2	3	60	RL	68.0	11250	Pave	0	IR1	Lvl	/
3	4	70	RL	60.0	9550	Pave	0	IR1	Lvl	/
4	5	60	RL	84.0	14260	Pave	0	IR1	Lvl	/

1.2. "Внедрение значений" - импьютация (imputation)

1.2.1. Обработка пропусков в числовых данных

In [103]:

```
# 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=='int64' or dt=='float64'):
        num_cols.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(col,
```

```
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, 5.5
5%.
```

In [104]:

```
data_num = data[num_cols]
data_num
```

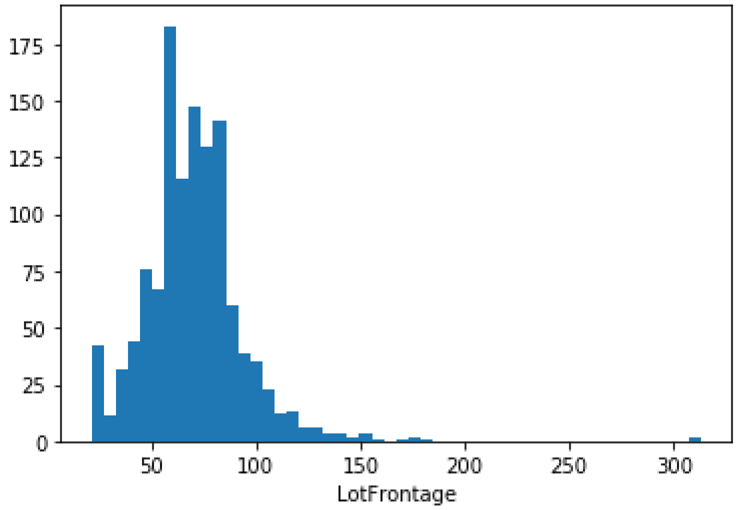
Out[104]:

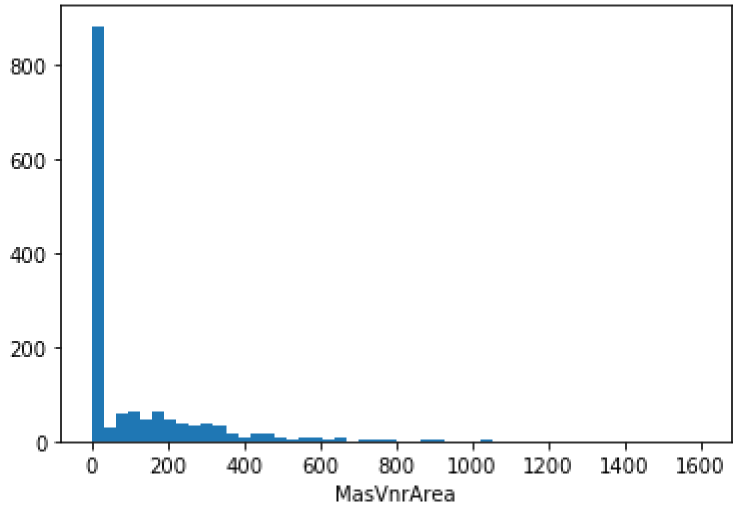
	LotFrontage	MasVnrArea	GarageYrBlt
0	65.0	196.0	2003.0
1	80.0	0.0	1976.0
2	68.0	162.0	2001.0
3	60.0	0.0	1998.0
4	84.0	350.0	2000.0
...	...	...	...
1455	62.0	0.0	1999.0
1456	85.0	119.0	1978.0
1457	66.0	0.0	1941.0
1458	68.0	0.0	1950.0
1459	75.0	0.0	1965.0

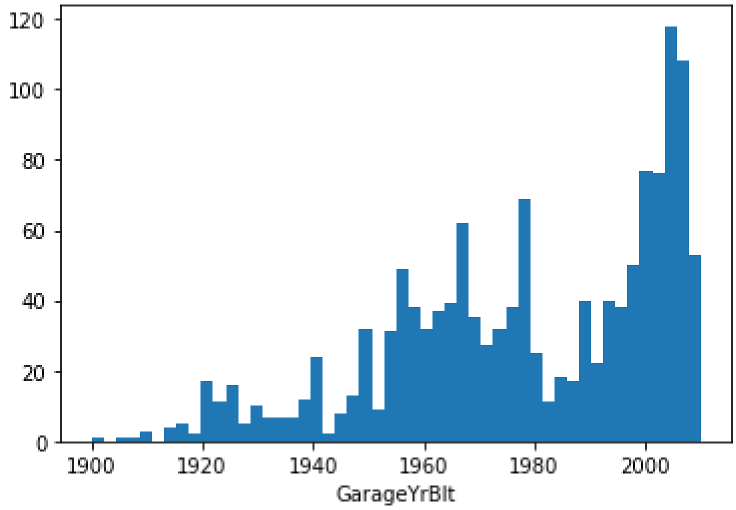
1460 rows × 3 columns

In [105]:

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







In [107]:

```
data[data['LotFrontage'].isnull()]
```

Out[107]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandConto
7	8	60	RL	NaN	10382	Pave	NaN	IR1	l
12	13	20	RL	NaN	12968	Pave	NaN	IR2	l
14	15	20	RL	NaN	10920	Pave	NaN	IR1	l
16	17	20	RL	NaN	11241	Pave	NaN	IR1	l
24	25	20	RL	NaN	8246	Pave	NaN	IR1	l
...	...	...	...	...	...	...	...	...	
1429	1430	20	RL	NaN	12546	Pave	NaN	IR1	l
1431	1432	120	RL	NaN	4928	Pave	NaN	IR1	l
1441	1442	120	RM	NaN	4426	Pave	NaN	Reg	l
1443	1444	30	RL	NaN	8854	Pave	NaN	Reg	l
1446	1447	20	RL	NaN	26142	Pave	NaN	IR1	l

259 rows × 81 columns

In [108]:

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

Out[108]:

```
Int64Index([   7,   12,   14,   16,   24,   31,   42,   43,   50,   64,
            ...
            1407, 1417, 1419, 1423, 1424, 1429, 1431, 1441, 1443, 1446],
           dtype='int64', length=259)
```

In [109]:

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

Out[109]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandConto
7	8	60	RL	NaN	10382	Pave	NaN	IR1	L
12	13	20	RL	NaN	12968	Pave	NaN	IR2	L
14	15	20	RL	NaN	10920	Pave	NaN	IR1	L
16	17	20	RL	NaN	11241	Pave	NaN	IR1	L
24	25	20	RL	NaN	8246	Pave	NaN	IR1	L
...	...	...	...	...	...	...	...	...	
1429	1430	20	RL	NaN	12546	Pave	NaN	IR1	L
1431	1432	120	RL	NaN	4928	Pave	NaN	IR1	L
1441	1442	120	RM	NaN	4426	Pave	NaN	Reg	L
1443	1444	30	RL	NaN	8854	Pave	NaN	Reg	L
1446	1447	20	RL	NaN	26142	Pave	NaN	IR1	L

259 rows × 81 columns

In [110]:

```
# фильтр по колонке
data_num[data_num.index.isin(flt_index)]['LotFrontage']
```

Out[110]:

```
7       NaN
12      NaN
14      NaN
16      NaN
24      NaN
         ..
1429    NaN
1431    NaN
1441    NaN
1443    NaN
1446    NaN
Name: LotFrontage, Length: 259, dtype: float64
```

In [115]:

```
data_num_LotFrontage = data_num[['LotFrontage']]
data_num_LotFrontage.head()
```

Out[115]:

	LotFrontage
0	65.0
1	80.0
2	68.0
3	60.0
4	84.0

In [116]:

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

In [117]:

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

Out[117]:

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

In [118]:

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

In [119]:

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

In [120]:

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

Out[120]:

```
('mean',
 array([70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837, 70.04995837,
        70.04995837, 70.04995837, 70.04995837, 70.04995837]))
```

In [121]:

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

Out[121]:

```
('median',
 array([69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.,
        69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69., 69.]))
```

In [122]:

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

Out[122]:

```
('most_frequent',
 array([60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.,
        60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60., 60.]))
```

In [123]:

```
# 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_dat
```

In [124]:

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

Out[124]:

	MasVnrArea
count	1452.000000
mean	103.685262
std	181.066207
min	0.000000
25%	0.000000
50%	0.000000
75%	166.000000
max	1600.000000

In [125]:

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

Out[125]:

```
('MasVnrArea', 'mean', 8, 103.68526170798899, 103.68526170798899)
```

In [126]:

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

Out[126]:

```
('MasVnrArea', 'median', 8, 0.0, 0.0)
```

In [127]:

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

Out[127]:

```
('MasVnrArea', 'most_frequent', 8, 0.0, 0.0)
```

1.2.2. Обработка пропусков в категориальных данных

In [128]:

```
# 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,
```

```
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, 2.5
3%.
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, 47.2
6%.
Колонка GarageType. Тип данных object. Количество пустых значений 81, 5.55%.
Колонка GarageFinish. Тип данных object. Количество пустых значений 81, 5.5
5%.
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%.
```

In [129]:

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

Out[129]:

	MasVnrType
0	BrkFace
1	None
2	BrkFace
3	None
4	BrkFace

In [130]:

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

Out[130]:

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

In [131]:

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

Out[131]:

```
(8, 1)
```

In [132]:

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

Out[132]:

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

In [133]:

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

Out[133]:

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

In [134]:

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

Out[134]:

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

In [135]:

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

Out[135]:

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

In [136]:

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

Out[136]:

```
8
```

2. Преобразование категориальных признаков в числовые

In [138]:

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

Out[138]:

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

1460 rows × 1 columns

2.1. Кодирование категорий целочисленными значениями - label encoding

In [139]:

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

In [140]:

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

In [141]:

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

Out[141]:

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

In [142]:

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

Out[142]:

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

In [143]:

```
le.inverse_transform([0, 1, 2, 3])
```

Out[143]:

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

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

In [144]:

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

In [145]:

```
cat_enc.shape
```

Out[145]:

```
(1460, 1)
```

In [146]:

```
cat_enc_ohe.shape
```

Out[146]:

```
(1460, 4)
```

In [147]:

```
cat_enc_ohe
```

Out[147]:

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

In [148]:

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

Out[148]:

```
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.]])
```

In [149]:

```
cat_enc.head(10)
```

Out[149]:

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

2.3. Pandas get_dummies - быстрый вариант one-hot кодирования

In [150]:

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

Out[150]:

	c1_BrkCmn	c1_BrkFace	c1_None	c1_Stone
0	0	1	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	1	0	0

In [151]:

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

Out[151]:

	MasVnrType_BrkCmn	MasVnrType_BrkFace	MasVnrType_None	MasVnrType_Stone	MasVnrTy
0	0	1	0	0	
1	0	0	1	0	
2	0	1	0	0	
3	0	0	1	0	
4	0	1	0	0	

3. Масштабирование данных

In [152]:

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

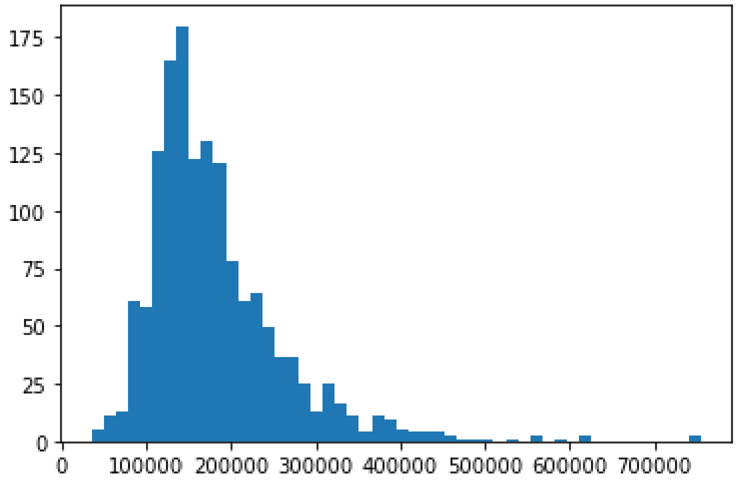
3.1. MinMax масштабирование

In [153]:

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

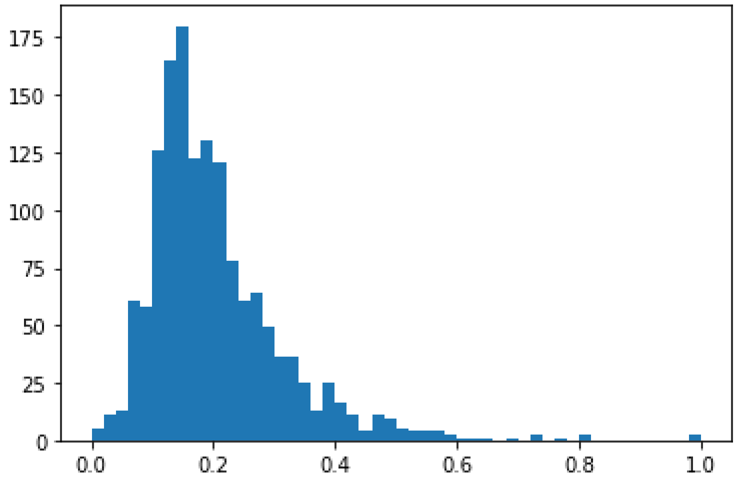
In [154]:

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



In [155]:

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



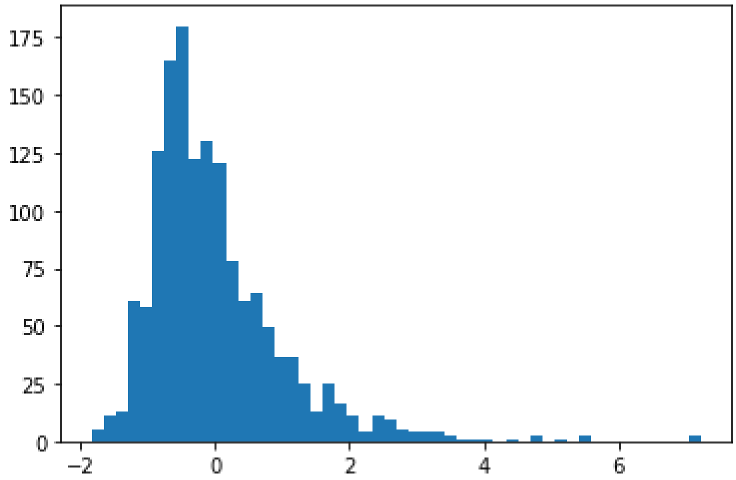
3.2. Масштабирование данных на основе Z-оценки - StandardScaler

In [156]:

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

In [157]:

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



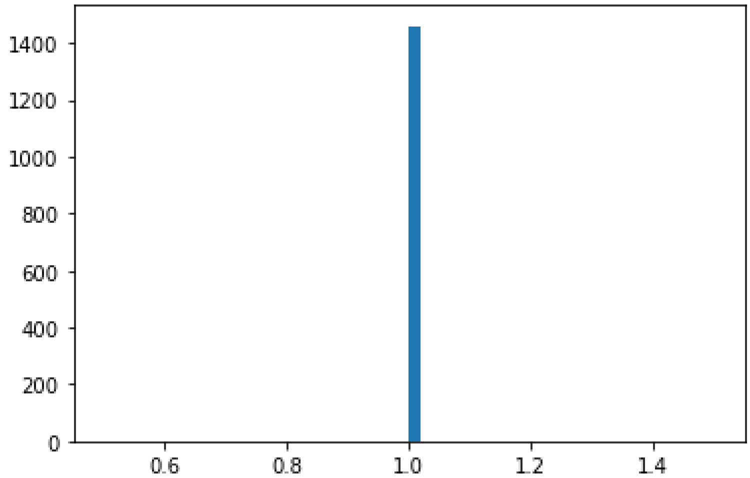
3.3. Нормализация данных

In [158]:

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

In [159]:

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



In [ ]:

```

```