Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»

Кафедра «Автоматизированные системы обработки информации и управления»

«Методы машинного обучения»

Отчет по Лабораторной работе №3

Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных

Выполнила:
студентка группы ИУ5-22М
Петрова Ирина
Проверил: доцент, к.т.н.
Гапанюк Ю. Е.

Москва, 2020

Лабораторная работа №3. Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных.

Цель лабораторной работы: изучение способов предварительной обработки данных для дальнейшего формирования моделей.

Требования к отчету: отчет по лабораторной работе должен содержать:

- титульный лист; описание задания; текст программы;
- экранные формы с примерами выполнения
- программы.
-

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 (не менее 3 признаков);
   - кодирование категориальных признаков (не менее 3 признаков);
   - масштабирование данных (не менее 3 признаков).

In [1]:

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

In [53]:

```
# Будем использовать только обучающую выборку
data = pd.read_csv('D:/Загрузки/train.csv', sep=",")
data.head()
```

Out[53]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
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

5 rows × 81 columns

In [54]:

```
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 [55]:

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

Out[55]: ((1460, 81),

(1460, 62)) In [56]:

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

Out[56]:

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

In [57]:

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

Out[57]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
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

5 rows × 81 columns

1.2. "Внедрение значений" - импьютация (imputation)

1.2.1. Обработка пропусков в числовых данных

In [58]:

```
# 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(c
ol, dt, temp_null_count, temp_perc))
```

```
Колонка LotFrontage. Тип данных float64. Количество пустых значений 259, 1
7.74%.
Колонка MasVnrArea. Тип данных float64. Количество пустых значений 8, 0.5
5%.
Колонка GarageYrBlt. Тип данных float64. Количество пустых значений 81, 5.
55%.
```

In [59]:

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

Out[59]:

	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 [61]:

```
# Фильтр по пустым значениям поля pain
data[data['MasVnrArea'].isnull()]
```

Out[61]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCon
234	235	60	RL	NaN	7851	Pave	NaN	Reg	
529	530	20	RL	NaN	32668	Pave	NaN	IR1	
650	651	60	FV	65.0	8125	Pave	NaN	Reg	
936	937	20	RL	67.0	10083	Pave	NaN	Reg	
973	974	20	FV	95.0	11639	Pave	NaN	Reg	**977** 978 120 FV 35.0 4274 Pave Pave IR1
1243	1244	20	RL	107.0	13891	Pave	NaN	Reg	
1278	1279	60	RL	75.0	9473	Pave	NaN	Reg	

8 rows × 81 columns

In [62]:

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

Out[62]: Int64Index([234, 529, 650, 936, 973, 977, 1243, 1278],

dtype='int64') In [63]:

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

Out[63]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCon
234	235	60	RL	NaN	7851	Pave	NaN	Reg	
529	530	20	RL	NaN	32668	Pave	NaN	IR1	
650	651	60	FV	65.0	8125	Pave	NaN	Reg	
936	937	20	RL	67.0	10083	Pave	NaN	Reg	
973	974	20	FV	95.0	11639	Pave	NaN	Reg	**977** 978 120 FV 35.0 4274 Pave Pave IR1
1243	1244	20	RL	107.0	13891	Pave	NaN	Reg	
1278	1279	60	RL	75.0	9473	Pave	NaN	Reg	

8 rows × 81 columns

In [64]:

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

Out[64]:

```
234    NaN
529    NaN
650    NaN
936    NaN
973    NaN
977    NaN
1243   NaN
1278   NaN
Name: MasVnrArea, dtype: float64
```

In [66]:

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

Out[66]:

	MasVnrArea
0	196.0
1	0.0
2	162.0
3	0.0
4	350.0

In [67]:

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

In [68]:

```
# Фильтр для проверки заполнения пустых значений
indicator = MissingIndicator()
mask_missing_values_only = indicator.fit_transform(data_num_MasVnrArea)
mask_missing_values_only Out[68]:
```

```
array([[False],
[False],
       [False],
...,
       [False],
       [False],
[False]]) In [69]:
```

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

In [42]:

```
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]
```

In [70]:

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

Out[70]:

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

In [71]:

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

Out[71]: ('median', array([0., 0., 0., 0., 0., 0.,

0., 0.])) In [72]:

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

Out[72]: ('most_frequent', array([0., 0., 0., 0., 0., 0.,

0., 0.]))

In [73]:

```
# 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]
```

In [75]:

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

Out[75]:

	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

In [76]:

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

Out[76]: ('GarageYrBlt', 'mean', 81, 1978.5061638868744,

1978.5061638868744) In [77]:

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

Out[77]: ('GarageYrBlt', 'median', 81,

1980.0, 1980.0) In [79]:

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

Out[79]:

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

1.2.1. Обработка пропусков в категориальных данных

In [80]:

```
# 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(c
ol, 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, 0.5
5%.
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, 0.0
7%.
Колонка FireplaceQu. Тип данных object. Количество пустых значений 690, 4
7.26%.
Колонка GarageType. Тип данных object. Количество пустых значений 81, 5.5
5%.
Колонка GarageFinish. Тип данных object. Количество пустых значений 81, 5.
55%.
Колонка GarageQual. Тип данных object. Количество пустых значений 81, 5.5
5%.
Колонка GarageCond. Тип данных object. Количество пустых значений 81, 5.5
5%.
Колонка PoolQC. Тип данных object. Количество пустых значений 1453, 99.5
2%.
Колонка Fence. Тип данных object. Количество пустых значений 1179, 80.75%.
Колонка MiscFeature. Тип данных object. Количество пустых значений 1406, 9
6.3%.
```

In [81]:

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

Out[81]:

	MasVnrType
0	BrkFace
1	None
2	BrkFace
3	None
4	BrkFace

In [82]:

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

Out[82]:

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

In [83]:

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

Out[83]:

```
(8, 1)
```

In [84]:

```
# Импьютация наиболее частыми значениями
imp2 = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
data_imp2 = imp2.fit_transform(cat_temp_data) data_imp2 Out[84]:

array([['BrkFace'],
['None'],
```

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

In [85]:

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

Out[85]:

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

In [86]:

```
# Импьютация константой
imp3 = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value='!!!')
data_imp3 = imp3.fit_transform(cat_temp_data) data_imp3 Out[86]:
```

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

In [87]:

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

Out[87]:

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

In [88]:

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

Out[88]:

```
8
```

2. Преобразование категориальных признаков вчисловые

In [89]:

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

Out[89]:

	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 - labelencoding

In [90]:

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

In [91]:

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

In [92]:

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

Out[92]:

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

In [93]:

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

Out[93]:

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

In [94]:

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

Out[94]:

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

2.2. Кодирование категорий наборами бинарных значений - one-hotencoding

In [95]:

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

In [96]:

```
cat_enc.shape
```

Out[96]:

```
(1460, 1)
```

In [97]:

```
cat_enc_ohe.shape
```

Out[97]:

(1460, 4) In

[98]:

```
cat_enc_ohe
```

Out[98]:

```
<1460x4 sparse matrix of type '<class 'numpy.float64'>'      with
1460 stored elements in Compressed Sparse Row format> In [99]:
```

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

Out[99]:

```
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 [100]:
```

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

Out[100]:

	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 [101]:

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

Out[101]:

	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 [102]:

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

Out[102]:

	MasVnrType_BrkCmn	MasVnrType_BrkFace	MasVnrType_None	MasVnrType_Stone	MasVnr
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 [103]:

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

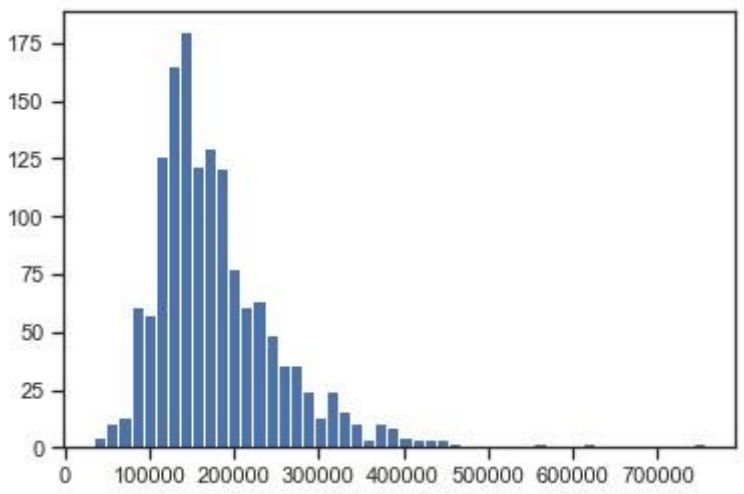
3.1. MinMax масштабирование

In [104]:

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

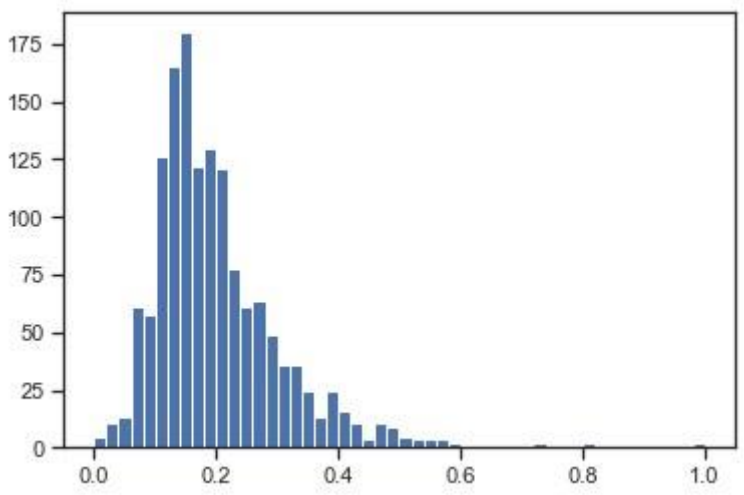
In [105]:

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



In [106]:

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



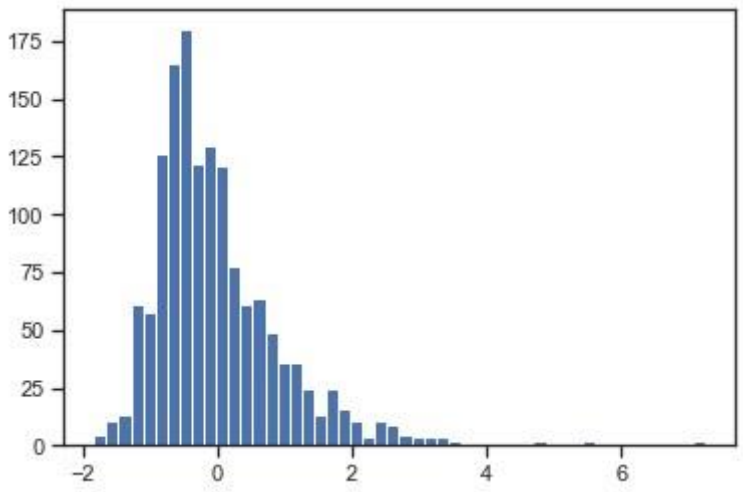
3.2. Масштабирование данных на основе Z-оценки - StandardScaler

In [107]:

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

In [108]:

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



3.3. Нормализация данных

In [109]:

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

In [110]:

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