Московский государственный технический университет им. Н. Э. Баумана

Курс «Технологии машинного обучения»

Отчёт по лабораторной работе №2

Выполнил:
Каятский П. Е.
группа ИУ5-64Б

Проверил:
Гапанюк Ю.Е.

Дата: 07.04.25

Дата:

Подпись:

Подпись:

Москва, 2025 г.

Цель лабораторной работы: 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.

Мой датасет: https://www.kaggle.com/datasets/gauravkumar2525/shark-attacks?resource=download

Ход выполнения:

Add Code Cell | Add Markdown Cell

[1]
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```
Executed at 2025.03.06 12:26:33 in 25ms

Загрузка и первичный анализ данных

Используем данные из(https://www.kaggle.com/datasets/gauravkumar2525/shark-attacks?resource=download)

[10]
```
# Будем использовать только обучающую выборку
data = pd.read_csv('../datasets/global_shark_attacks.csv', sep=",")
```
Executed at 2025.03.06 12:29:08 in 50ms

[11]
```
# размер набора данных
data.shape
```
Executed at 2025.03.06 12:29:12 in 11ms

```
(6890, 13)
```

[12]
```
# типы колонок
data.dtypes
```
Executed at 2025.03.06 12:29:17 in 6ms

```
date          object
year         float64
type          object
country       object
area          object
location      object
activity      object
name          object
sex           object
age           object
fatal_y_n     object
```

[13]
```
# проверим есть ли пропущенные значения
data.isnull().sum()
```
Executed at 2025.03.06 12:29:39 in 16ms

```
date          303
year          132
type           19
country        51
area          481
location      565
activity      586
name          220
sex           572
age          2987
fatal_y_n       0
```

[14]
```
# Первые 5 строк датасета
data.head()
```
Executed at 2025.03.06 12:29:47 in 15ms

5 rows × 13 columns — Static Output

	date	year	type	country	area	location
0	2023-05-13	2023.0	Unprovoked	AUSTRALIA	South Australia	Elliston
1	2023-04-29	2023.0	Unprovoked	AUSTRALIA	Western Australia	Yallingup, Busselton
2	2022-10-07	2022.0	Unprovoked	AUSTRALIA	Western Australia	Port Hedland
3	2021-10-04	2021.0	Unprovoked	USA	Florida	Fort Pierce State Park, St. Lucie County
4	2021-10-03	2021.0	Unprovoked	USA	Florida	Jensen Beach, Martin County

[15]
```
total_count = data.shape[0]
print('Всего строк: {}'.format(total_count))
```
Executed at 2025.03.06 12:29:51 in 10ms

```
Всего строк: 6890
```

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.

[16]
```
# Удаление колонок, содержащих пустые значения
data_new_1 = data.dropna(axis=1, how='any')
(data.shape, data_new_1.shape)
```
Executed at 2025.03.06 12:30:09 in 16ms

```
((6890, 13), (6890, 1))
```

[17]
```
# Удаление строк, содержащих пустые значения
data_new_2 = data.dropna(axis=0, how='any')
(data.shape, data_new_2.shape)
```
Executed at 2025.03.06 12:30:28 in 11ms

```
((6890, 13), (1653, 13))
```

[18]
```
data.head()
```
Executed at 2025.03.06 12:30:43 in 16ms

5 rows | 5 rows × 13 columns | Static Output

	date	year	type	country	area	location
0	2023-05-13	2023.0	Unprovoked	AUSTRALIA	South Australia	Elliston
1	2023-04-29	2023.0	Unprovoked	AUSTRALIA	Western Australia	Yallingup, Busselton
2	2022-10-07	2022.0	Unprovoked	AUSTRALIA	Western Australia	Port Hedland
3	2021-10-04	2021.0	Unprovoked	USA	Florida	Fort Pierce State Park, St. Lucie County
4	2021-10-03	2021.0	Unprovoked	USA	Florida	Jensen Beach, Martin County

[19]
```
# 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()
```
Executed at 2025.03.06 12:31:14 in 26ms

5 rows | 5 rows × 13 columns | Static Output

	date	year	type	country	area	location
0	2023-05-13	2023.0	Unprovoked	AUSTRALIA	South Australia	Elliston
1	2023-04-29	2023.0	Unprovoked	AUSTRALIA	Western Australia	Yallingup, Busselton
2	2022-10-07	2022.0	Unprovoked	AUSTRALIA	Western Australia	Port Hedland
3	2021-10-04	2021.0	Unprovoked	USA	Florida	Fort Pierce State Park, St. Lucie County
4	2021-10-03	2021.0	Unprovoked	USA	Florida	Jensen Beach, Martin County

"Внедрение значений" - импьютация (imputation)

1 2 531

Обработка пропусков в числовых данных

[20]
```
# 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))
```
Executed at 2025.03.06 12:31:28 in 24ms

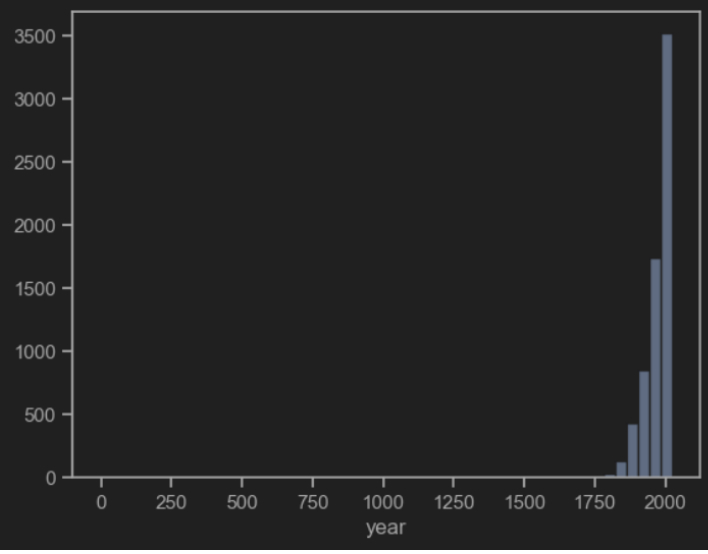
```
Колонка year. Тип данных float64. Количество пустых значений 132, 1.92%.
```

[21]
```
# Фильтр по колонкам с пропущенными значениями
data_num = data[num_cols]
data_num
```
Executed at 2025.03.06 12:31:34 in 17ms

11 rows · 6890 rows × 1 columns · Static Output

	year
0	2023.0
1	2023.0
2	2022.0
3	2021.0
4	2021.0
...	...
6885	NaN
6886	NaN
6887	NaN
6888	NaN
6889	NaN

[22]
```
# Гистограмма по признакам
for col in data_num:
    plt.hist(data[col], 50)
    plt.xlabel(col)
    plt.show()
```
Executed at 2025.03.06 12:32:03 in 345ms



Будем использовать встроенные средства импьютации библиотеки scikit-learn - https://scikit-learn.org/stable/modules/impute.html

[24]
```
data_num_MasVnrArea = data_num[['year']]
data_num_MasVnrArea.head()
```
Executed at 2025.03.06 12:32:28 in 15ms

5 rows · 5 rows × 1 columns · Static Output

	year
0	2023.0
1	2023.0
2	2022.0
3	2021.0
4	2021.0

[26]
```
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
```
Executed at 2025.03.06 12:39:08 in 1s 660ms

[27]
```
# Фильтр для проверки заполнения пустых значений
indicator = MissingIndicator()
mask_missing_values_only = indicator.fit_transform(data_num_MasVnrArea)
mask_missing_values_only
```
Executed at 2025.03.06 12:39:15 in 12ms

```
array([[False],
       [False],
       [False],
       ...,
       [ True],
       [ True],
       [ True]], shape=(6890, 1))
```

С помощью класса SimpleImputer можно проводить импьютацию различными показателями центра распределения

[28]
```
strategies=['mean', 'median', 'most_frequent']
```
Executed at 2025.03.06 12:39:37 in 6ms

[29]
```
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]
```
Executed at 2025.03.06 12:39:40 in 6ms

[30]
```
strategies[0], test_num_impute(strategies[0])
```
Executed at 2025.03.06 12:39:42 in 14ms

```
1970.93592779, 1970.93592779, 1970.93592779, 1970.93592779,
1970.93592779, 1970.93592779, 1970.93592779, 1970.93592779,
1970.93592779, 1970.93592779, 1970.93592779, 1970.93592779,
1970.93592779, 1970.93592779, 1970.93592779, 1970.93592779,
1970.93592779, 1970.93592779, 1970.93592779, 1970.93592779,
1970.93592779, 1970.93592779, 1970.93592779, 1970.93592779,
1970.93592779, 1970.93592779, 1970.93592779, 1970.93592779,
1970.93592779, 1970.93592779, 1970.93592779, 1970.93592779,
1970.93592779, 1970.93592779, 1970.93592779, 1970.93592779,
1970.93592779, 1970.93592779, 1970.93592779, 1970.93592779]))
```

[31]
```python
strategies[1], test_num_impute(strategies[1])
```
Executed at 2025.03.06 12:39:50 in 12ms

```
        1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986.,
        1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986.,
        1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986.,
        1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986.,
        1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986.,
        1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986.,
        1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986.,
        1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986.,
        1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986.,
        1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986., 1986.,
        1986., 1986., 1986., 1986., 1986., 1986.]))
```

[32]
```python
strategies[2], test_num_impute(strategies[2])
```
Executed at 2025.03.06 12:39:55 in 15ms

```
        2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015.,
        2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015.,
        2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015.,
        2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015.,
        2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015.,
        2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015.,
        2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015.,
        2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015.,
        2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015.,
        2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015., 2015.,
        2015., 2015., 2015., 2015., 2015., 2015.]))
```

[33]
```python
# 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]
```
Executed at 2025.03.06 12:40:01 in 10ms

[34]
```python
data[['year']].describe()
```
Executed at 2025.03.06 12:40:11 in 13ms

8 rows × 1 columns

	year
count	6758.000000
mean	1970.935928
std	56.227881
min	1.000000
25%	1950.000000
50%	1986.000000
75%	2009.000000
max	2023.000000

[36]
```python
test_num_impute_col(data, 'year', strategies[0])
```
Executed at 2025.03.06 12:40:26 in 17ms

```
('year',
 'mean',
 132,
 np.float64(1970.9359277892868),
 np.float64(1970.9359277892868))
```

```
[37] 1 test_num_impute_col(data, 'year', strategies[1])
Executed at 2025.03.06 12:40:32 in 15ms
```

```
('year', 'median', 132, np.float64(1986.0), np.float64(1986.0))
```

```
[38] 1 test_num_impute_col(data, 'year', strategies[2])
Executed at 2025.03.06 12:40:37 in 17ms
```

```
('year', 'most_frequent', 132, np.float64(2015.0), np.float64(2015.0))
```

Обработка пропусков в категориальных данных

1 2 531

```
[39] 1 # Выберем категориальные колонки с пропущенными значениями
     2 # Цикл по колонкам датасета
     3 cat_cols = []
     4 for col in data.columns:
     5     # Количество пустых значений
     6     temp_null_count = data[data[col].isnull()].shape[0]
     7     dt = str(data[col].dtype)
     8     if temp_null_count>0 and (dt=='object'):
     9         cat_cols.append(col)
    10         temp_perc = round((temp_null_count / total_count) * 100.0, 2)
    11         print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(col, dt,
                temp_null_count, temp_perc))
Executed at 2025.03.06 12:41:04 in 23ms
```

```
Колонка date. Тип данных object. Количество пустых значений 303, 4.4%.
Колонка type. Тип данных object. Количество пустых значений 19, 0.28%.
Колонка country. Тип данных object. Количество пустых значений 51, 0.74%.
Колонка area. Тип данных object. Количество пустых значений 481, 6.98%.
Колонка location. Тип данных object. Количество пустых значений 565, 8.2%.
Колонка activity. Тип данных object. Количество пустых значений 586, 8.51%.
Колонка name. Тип данных object. Количество пустых значений 220, 3.19%.
Колонка sex. Тип данных object. Количество пустых значений 572, 8.3%.
Колонка age. Тип данных object. Количество пустых значений 2987, 43.35%.
Колонка time. Тип данных object. Количество пустых значений 3518, 51.06%.
Колонка species. Тип данных object. Количество пустых значений 3118, 45.25%.
```

1 2 531

```
[40] 1 cat_temp_data = data[['country']]
     2 cat_temp_data.head()
Executed at 2025.03.06 12:42:47 in 16ms
```

5 rows | 5 rows × 1 columns | Static Output

	country
0	AUSTRALIA
1	AUSTRALIA
2	AUSTRALIA
3	USA
4	USA

```
[41] 1 cat_temp_data['country'].unique()
Executed at 2025.03.06 12:43:02 in 6ms
```

```
'ITALY / CROATIA', 'MONTENEGRO', 'YEMEN', 'VIETNAM',
'COOK ISLANDS', 'WEST INDIES', 'TURKS and CaICOS', 'Sierra Leone',
'TONGA', 'VANUATU', 'IRAN', 'HONG KONG', 'NEW BRITAIN',
'SOLOMON ISLANDS / VANUATU', 'PANAMA', 'GUYANA', 'ATLANTIC OCEAN',
'INDIAN OCEAN?', 'NORWAY', 'ANTIGUA', 'CANADA',
'MEDITERRANEAN SEA', 'BURMA', 'ST KITTS / NEVIS', 'MALDIVES',
'ECUADOR', 'VENEZUELA', 'PERSIAN GULF', 'THE BALKANS', 'NICARAGUA',
'BARBADOS', 'SAN DOMINGO', 'CROATIA', 'FRANCE', 'LEBANON',
'RED SEA?', 'IRAQ', 'NIGERIA', 'WESTERN SAMOA', 'SCOTLAND',
'CAPE VERDE', 'CEYLON', 'ROATAN', 'PERU', 'SUDAN', 'SAUDI ARABIA',
'SEYCHELLES', 'ARGENTINA', 'REUNION ISLAND', 'GRENADA',
```

[42]
```
cat_temp_data[cat_temp_data['country'].isnull()].shape
```
Executed at 2025.03.06 12:43:13 in 13ms

```
(51, 1)
```

[43]
```
# Импьютация наиболее частыми значениями
imp2 = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
data_imp2 = imp2.fit_transform(cat_temp_data)
data_imp2
```
Executed at 2025.03.06 12:43:17 in 13ms

```
array([['AUSTRALIA'],
       ['AUSTRALIA'],
       ['AUSTRALIA'],
       ...,
       ['IRAQ'],
       ['SOUTH AFRICA'],
       ['AUSTRALIA']], shape=(6890, 1), dtype=object)
```

[44]
```
# Пустые значения отсутствуют
np.unique(data_imp2)
```
Executed at 2025.03.06 12:43:22 in 8ms

```
       'INDIAN OCEAN?', 'INDONESIA', 'IRAN', 'IRAN / IRAQ', 'IRAQ',
       'IRELAND', 'ISRAEL', 'ITALY', 'ITALY / CROATIA', 'JAMAICA',
       'JAPAN', 'JAVA', 'JOHNSTON ISLAND', 'JORDAN', 'KENYA', 'KIRIBATI',
       'KOREA', 'KUWAIT', 'LEBANON', 'LIBERIA', 'LIBYA', 'MADAGASCAR',
       'MALAYSIA', 'MALDIVE ISLANDS', 'MALDIVES', 'MALTA',
       'MARSHALL ISLANDS', 'MARTINIQUE', 'MAURITIUS', 'MAYOTTE',
       'MEDITERRANEAN SEA', 'MEXICO', 'MICRONESIA', 'MID ATLANTIC OCEAN',
       'MID-PACIFC OCEAN', 'MONACO', 'MONTENEGRO', 'MOZAMBIQUE',
       'Maldives', 'MeXICO', 'NAMIBIA', 'NETHERLANDS ANTILLES', 'NEVIS',
       'NEW BRITAIN', 'NEW CALEDONIA', 'NEW GUINEA', 'NEW ZEALAND',
       'NICARAGUA', 'NIGERIA', 'NORTH ATLANTIC OCEAN',
```

[45]
```
# Импьютация константой
imp3 = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value='NA')
data_imp3 = imp3.fit_transform(cat_temp_data)
data_imp3
```
Executed at 2025.03.06 12:43:32 in 10ms

```
array([['AUSTRALIA'],
       ['AUSTRALIA'],
       ['AUSTRALIA'],
       ...,
       ['IRAQ'],
       ['SOUTH AFRICA'],
       ['AUSTRALIA']], shape=(6890, 1), dtype=object)
```

Преобразование категориальных признаков в числовые

```
[48] 1 cat_enc = pd.DataFrame({'c1':data_imp2.T[0]})
     2 cat_enc
```

Executed at 2025.03.06 12:43:44 in 8ms

11 rows | 6890 rows × 1 columns | Static Output

	c1
0	AUSTRALIA
1	AUSTRALIA
2	AUSTRALIA
3	USA
4	USA
...	...
6885	GREECE
6886	INDONESIA
6887	IRAQ
6888	SOUTH AFRICA
6889	AUSTRALIA

Использование LabelEncoder

```
[49] 1 from sklearn.preprocessing import LabelEncoder
```

Executed at 2025.03.06 12:43:58 in 8ms

```
[50] 1 cat_enc['c1'].unique()
```

Executed at 2025.03.06 12:44:00 in 11ms

```
'MONACO', 'GHANA', 'EGYPT / ISRAEL', 'IRAN / IRAQ', 'JAVA',
'ST HELENA, British overseas territory', 'ST. MAARTIN',
'NORTH SEA', 'BRITISH NEW GUINEA', 'SWEDEN',
'British Overseas Territory', 'EL SALVADOR',
'RED SEA / INDIAN OCEAN', 'ANGOLA', 'CENTRAL PACIFIC', 'Australia',
'UNITED ARAB EMIRATES', 'GRAND CAYMAN', 'GUINEA', 'New Zealand',
'NEVIS', 'CURACAO', 'PARAGUAY', 'DIEGO GARCIA', 'SOUTH CHINA SEA',
'BAY OF BENGAL', 'COMOROS', 'ARUBA', 'BRITISH VIRGIN ISLANDS',
'NORTHERN ARABIAN SEA', 'GABON', 'FALKLAND ISLANDS', 'BAHREIN',
'Maldives', 'ST. MARTIN', 'MID-PACIFC OCEAN', 'TASMAN SEA',
'KOREA'], dtype=object)
```

```
[51] 1 le = LabelEncoder()
     2 cat_enc_le = le.fit_transform(cat_enc['c1'])
```

Executed at 2025.03.06 12:44:11 in 15ms

```
[52] 1  # Наименования категорий в соответствии с порядковыми номерами
     2
     3  # Свойство называется classes, потому что предполагается что мы решаем
     4  # задачу классификации и каждое значение категории соответствует
     5  # какому-либо классу целевого признака
     6
     7  le.classes_
```

Executed at 2025.03.06 12:44:14 in 12ms

```
array(['SOUTH CHINA SEA', 'SOUTH KOREA', 'SOUTH PACIFIC OCEAN',
       'SOUTHWEST PACIFIC OCEAN', 'SPAIN', 'SRI LANKA',
       'ST HELENA, British overseas territory', 'ST KITTS / NEVIS',
       'ST MARTIN', 'ST. MAARTIN', 'ST. MARTIN', 'SUDAN', 'SUDAN?',
       'SWEDEN', 'SYRIA', 'Seychelles', 'Sierra Leone', 'South Africa',
       'TAIWAN', 'TANZANIA', 'TASMAN SEA', 'THAILAND', 'THE BALKANS',
       'TOBAGO', 'TONGA', 'TRINIDAD & TOBAGO', 'TUNISIA', 'TURKEY',
       'TURKS & CAICOS', 'TURKS and CaICOS', 'TUVALU',
       'UNITED ARAB EMIRATES', 'UNITED ARAB EMIRATES (UAE)',
       'UNITED KINGDOM', 'URUGUAY', 'USA', 'VANUATU', 'VENEZUELA',
       'VIETNAM', 'WEST INDIES', 'WESTERN SAMOA', 'YEMEN'], dtype=object)
```

```
[53] 1  cat_enc_le
```

Executed at 2025.03.06 12:44:17 in 8ms

```
array([ 12,  12,  12, ...,  87, 171,  12], shape=(6890,))
```

Add Code Cell | Add Markdown Cell

```
[53] 1  cat_enc_le
```

Executed at 2025.03.06 12:44:17 in 8ms

```
array([ 12,  12,  12, ...,  87, 171,  12], shape=(6890,))
```

```
[54] 1  np.unique(cat_enc_le)
```

Executed at 2025.03.06 12:44:20 in 16ms

```
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,
```

```
[55] 1  # В этом примере видно, что перед кодированием
     2  # уникальные значения признака сортируются в лексикографическом порядке
     3  le.inverse_transform([0, 1, 2, 3])
```

Executed at 2025.03.06 12:44:24 in 8ms

```
array(['ADMIRALTY ISLANDS', 'AFRICA', 'ALGERIA', 'AMERICAN SAMOA'],
      dtype=object)
```

Использование OrdinalEncoder

[56]
```
from sklearn.preprocessing import OrdinalEncoder
```
Executed at 2025.03.06 12:44:30 in 4ms

[57]
```
data_oe = data[['activity', 'country', 'sex']]
data_oe.head()
```
Executed at 2025.03.06 12:45:36 in 16ms

5 rows 5 rows × 3 columns Static Output

	activity	country	sex
0	Surfing	AUSTRALIA	M
1	Swimming	AUSTRALIA	M
2	Spearfishing	AUSTRALIA	M
3	Surfing	USA	M
4	Swimming	USA	M

[58]
```
imp4 = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value='NA')
data_oe_filled = imp4.fit_transform(data_oe)
data_oe_filled
```
Executed at 2025.03.06 12:45:42 in 9ms

```
array([['Surfing', 'AUSTRALIA', 'M'],
       ['Swimming', 'AUSTRALIA', 'M'],
       ['Spearfishing', 'AUSTRALIA', 'M'],
       ...,
       ['Swimming', 'IRAQ', 'M'],
       ['Crew swimming alongside their anchored ship', 'SOUTH AFRICA',
        'M'],
       ['Pearl diving', 'AUSTRALIA', 'M']], shape=(6890, 3), dtype=object)
```

[59]
```
oe = OrdinalEncoder()
cat_enc_oe = oe.fit_transform(data_oe_filled)
cat_enc_oe
```
Executed at 2025.03.06 12:45:49 in 19ms

```
array([[1166.,   12.,    2.],
       [1202.,   12.,    2.],
       [1055.,   12.,    2.],
       ...,
       [1202.,   87.,    2.],
       [ 244.,  172.,    2.],
       [ 855.,   12.,    2.]], shape=(6890, 3))
```

[60]
```
# Уникальные значения 1 признака
np.unique(cat_enc_oe[:, 0])
```
Executed at 2025.03.06 12:45:52 in 8ms

```
array([0.000e+00, 1.000e+00, 2.000e+00, ..., 1.551e+03, 1.552e+03,
       1.553e+03], shape=(1554,))
```

[61]
```
# Уникальные значения 2 признака
np.unique(cat_enc_oe[:, 1])
```
Executed at 2025.03.06 12:45:54 in 11ms

```
        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.,
```

[62]
```
# Уникальные значения 3 признака
np.unique(cat_enc_oe[:, 2])
```
Executed at 2025.03.06 12:45:56 in 15ms

```
array([0., 1., 2., 3., 4., 5., 6.])
```

[63]
```
# Наименования категорий в соответствии с порядковыми номерами
oe.categories_
```
Executed at 2025.03.06 12:45:59 in 11ms

```
       'SLOVENIA', 'SOLOMON ISLANDS', 'SOLOMON ISLANDS / VANUATU',
       'SOMALIA', 'SOUTH AFRICA', 'SOUTH ATLANTIC OCEAN',
       'SOUTH CHINA SEA', 'SOUTH KOREA', 'SOUTH PACIFIC OCEAN',
       'SOUTHWEST PACIFIC OCEAN', 'SPAIN', 'SRI LANKA',
       'ST HELENA, British overseas territory', 'ST KITTS / NEVIS',
       'ST MARTIN', 'ST. MAARTIN', 'ST. MARTIN', 'SUDAN', 'SUDAN?',
       'SWEDEN', 'SYRIA', 'Seychelles', 'Sierra Leone', 'South Africa',
       'TAIWAN', 'TANZANIA', 'TASMAN SEA', 'THAILAND', 'THE BALKANS',
       'TOBAGO', 'TONGA', 'TRINIDAD & TOBAGO', 'TUNISIA', 'TURKEY',
       'TURKS & CAICOS', 'TURKS and CaICOS', 'TUVALU',
       'UNITED ARAB EMIRATES', 'UNITED ARAB EMIRATES (UAE)',
```

[64]
```
# Обратное преобразование
oe.inverse_transform(cat_enc_oe)
```
Executed at 2025.03.06 12:46:13 in 6ms

```
array([['Surfing', 'AUSTRALIA', 'M'],
       ['Swimming', 'AUSTRALIA', 'M'],
       ['Spearfishing', 'AUSTRALIA', 'M'],
       ...,
       ['Swimming', 'IRAQ', 'M'],
       ['Crew swimming alongside their anchored ship', 'SOUTH AFRICA',
        'M'],
       ['Pearl diving', 'AUSTRALIA', 'M']], shape=(6890, 3), 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.

[65]
```
# пример шкалы порядка 'small' < 'medium' < 'large'
sizes = ['small', 'medium', 'large', 'small', 'medium', 'large', 'small', 'medium', 'large']
```
Executed at 2025.03.06 12:46:19 in 6ms

[66]
```
pd_sizes = pd.DataFrame(data={'sizes':sizes})
pd_sizes
```
Executed at 2025.03.06 12:46:21 in 15ms

⚠ 1 ⚠ 2 ✓ 531

[67]
```
pd_sizes['sizes_codes'] = pd_sizes['sizes'].map({'small':1, 'medium':2, 'large':3})
pd_sizes
```
Executed at 2025.03.06 12:46:25 in 11ms

9 rows | 9 rows × 2 columns | Static Output

	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

Кодирование категорий наборами бинарных значений - one-hot encoding

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

[69]
```
from sklearn.preprocessing import OneHotEncoder
```
Executed at 2025.03.06 12:46:32 in 9ms

[70]
```
ohe = OneHotEncoder()
cat_enc_ohe = ohe.fit_transform(cat_enc[['c1']])
```
Executed at 2025.03.06 12:46:35 in 9ms

[71]
```
cat_enc.shape
```
Executed at 2025.03.06 12:46:35 in 5ms

```
(6890, 1)
```

[72]
```
cat_enc_ohe.shape
```
Executed at 2025.03.06 12:46:36 in 11ms

```
(6890, 215)
```

[73]
```
cat_enc_ohe
```
Executed at 2025.03.06 12:46:37 in 5ms

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
    with 6890 stored elements and shape (6890, 215)>
```

[74]
```
cat_enc_ohe.todense()[0:10]
```
Executed at 2025.03.06 12:46:38 in 9ms

```
matrix([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]], shape=(10, 215))
```

[75]
```
cat_enc.head(10)
```
Executed at 2025.03.06 12:46:39 in 16ms

10 rows | 10 rows × 1 columns | Static Output

	c1
0	AUSTRALIA
1	AUSTRALIA
2	AUSTRALIA
3	USA
4	USA
5	USA
6	USA
7	JAMAICA
8	USA
9	AUSTRALIA

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:

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

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

- Масштабирование данных на основе Z-оценки:

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

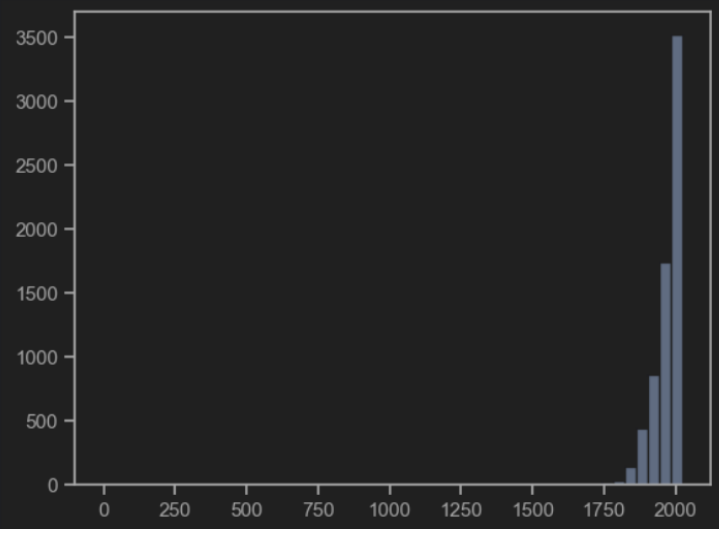
В этом случае большинство значений попадает в диапазон от -3 до 3.

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

MinMax масштабирование

```
[85] 1 sc1 = MinMaxScaler()
     2 sc1_data = sc1.fit_transform(data[['year']])
```
Executed at 2025.03.06 12:58:10 in 9ms

```
[86] 1 plt.hist(data['year'], 50)
     2 plt.show()
```
Executed at 2025.03.06 12:58:15 in 141ms



Масштабирование данных на основе Z-оценки - StandardScal

1 2 531

```
[88] 1 sc2 = StandardScaler()
     2 sc2_data = sc2.fit_transform(data[['year']])
```
Executed at 2025.03.06 12:58:36 in 13ms

```
[89] 1 plt.hist(sc2_data, 50)
     2 plt.show()
```
Executed at 2025.03.06 12:58:39 in 144ms