Искусственный Интеллект

Лекция 5: Матричные вычисления. Предобработка данных.

Мартынюк Полина Антоновна

telegram: @PAMartynyuk

email: pa-martynyuk@yandex.ru

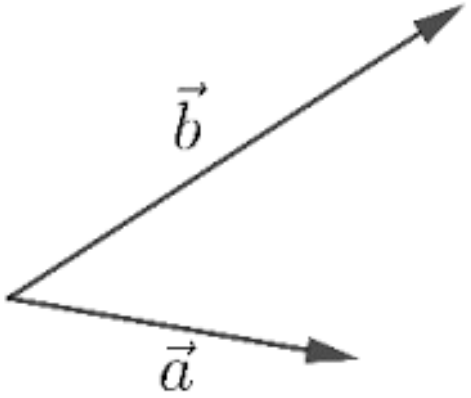
Матричные вычисления

- Основная формула линейной регрессии:

$$y = \sum_{i=1}^{p}(x_i w_i) + b$$

$$y = x_1 w_1 + x_2 w_2 + \ldots + x_p w_p + b$$

- На что похоже?
- **Формула скалярного произведения векторов**



$$\vec{a}\{x_1; y_1; z_1\} \qquad \vec{b}\{x_2; y_2; z_2\}$$

$$\vec{a} \cdot \vec{b} = x_1 x_2 + y_1 y_2 + z_1 z_2$$

Матричные вычисления

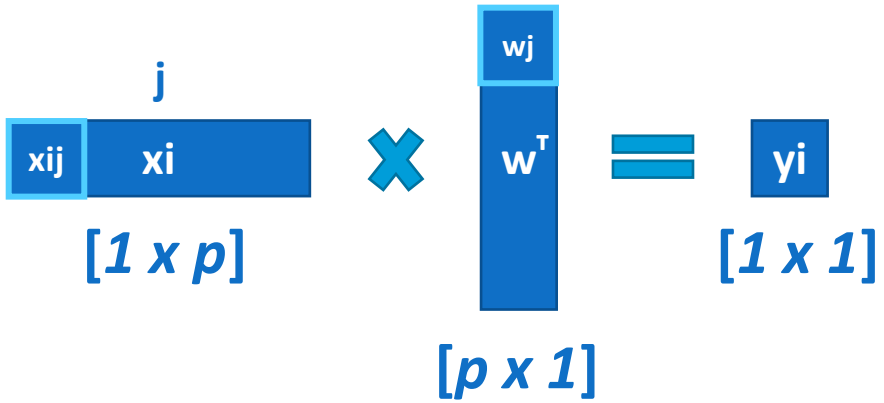
- Перепишем в векторном виде:

$$y = x_1 w_1 + x_2 w_2 + \ldots + x_p w_p + b$$

$$\underset{[1 \times 1]}{y} = \underset{[1 \times p]}{[x_1 \ldots x_p]} * \underset{[p \times 1]}{\begin{bmatrix} w_1 \\ \ldots \\ w_p \end{bmatrix}} + \underset{[1 \times 1]}{b}$$

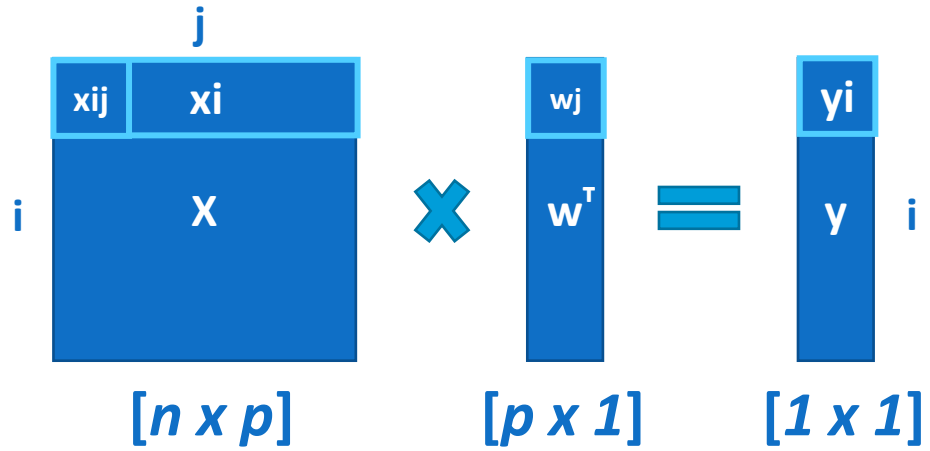
Матричные вычисления

- Преобразования размерностей



Размерность результата

[1 x p] * [p x 1] = [1 x 1]



[n x p] * [p x 1] = [n x 1]

Матричные вычисления

- Преобразования размерностей

$$y = [x_1 \dots x_p] * \begin{bmatrix} w_1 \\ \dots \\ w_p \end{bmatrix} + b$$

$$\begin{bmatrix} y^{(1)} \\ \dots \\ y^{(n)} \end{bmatrix} = \begin{bmatrix} x_1{}^{(1)} & \dots & x_p{}^{(1)} \\ \dots & \dots & \dots \\ x_1{}^{(n)} & \dots & x_p{}^{(n)} \end{bmatrix} * \begin{bmatrix} w_1 \\ \dots \\ w_p \end{bmatrix} + b$$

Матричные вычисления

- Преобразования размерностей

Было:

$$w_1, \ldots, w_p := 0$$
$$b := 0$$
$$for\ i\ in\ range(n\_iter):$$
$$\quad w_1 := w_1 - \alpha \frac{\partial L}{\partial w_1}$$
$$\quad \ldots$$
$$\quad w_p := w_p - \alpha \frac{\partial L}{\partial w_p}$$
$$\quad b := b - \alpha \frac{\partial L}{\partial b}$$

Стало:

$$W := [0, \ldots, 0]$$
$$b := 0$$
$$for\ i\ in\ range(n\_iter):$$
$$\quad W := W - \alpha \frac{\partial L}{\partial W}$$
$$\quad b := b - \alpha \frac{\partial L}{\partial b}$$

Матричные вычисления

- Проведём эксперимент, сравним время работы

```
[51] N = 10000

     weights = np.random.rand(N) # generate N weights
     X = np.random.randint(0, 100, N) # generate N features
```

```
[52] X
```

```
array([39, 63, 40, ..., 77, 70, 21])
```

```
[53] len(X)
```

```
10000
```

```
[54] weights
```

```
array([0.5836377 , 0.34974885, 0.21639855, ..., 0.78468476, 0.53201545,
       0.54283647])
```

```
[55] len(weights)
```

```
10000
```

Матричные вычисления

- Реализуем функцию линейной регрессии двумя способами:
  - Без матричных вычислений
  - С матричными вычислениями

```
[56] def unvectorized(X, weights):
         prediction = 0.0
         for x, weight in zip(X, weights):
             prediction += x * weight
         return prediction
```

```
[57] def vectorized(X, weights):
         prediction = np.dot(X, weights.T)
         return prediction
```

Матричные вычисления

- Сравним время выполнения функций на одних и тех же данных

  %time

  https://ipython.readthedocs.io/en/stable/interactive/magics.html

```
[58] %time unvectorized(X, weights)
```

```
CPU times: user 7.83 ms, sys: 0 ns, total: 7.83 ms
Wall time: 27.5 ms
246646.01705608808
```

```
[59] %time vectorized(X, weights)
```

```
CPU times: user 93 µs, sys: 0 ns, total: 93 µs
Wall time: 97.5 µs
246646.0170560878
```

Предобработка данных

- **Виды данных**
  - ➢Числовые
  - ➢Категориальные

Категориальные:

- ➢Бинарные

  Всего 2 категории (класса)
- ➢Небинарные

  Произвольное число категорий (классов)

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Предобработка данных

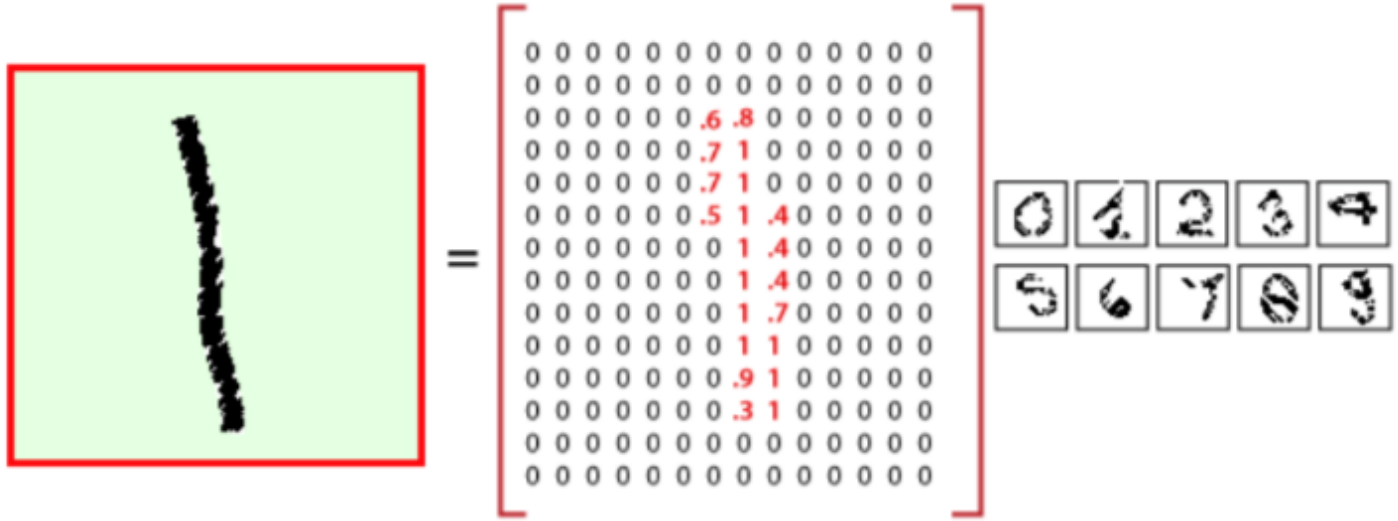
- **Также:**
  - Изображения

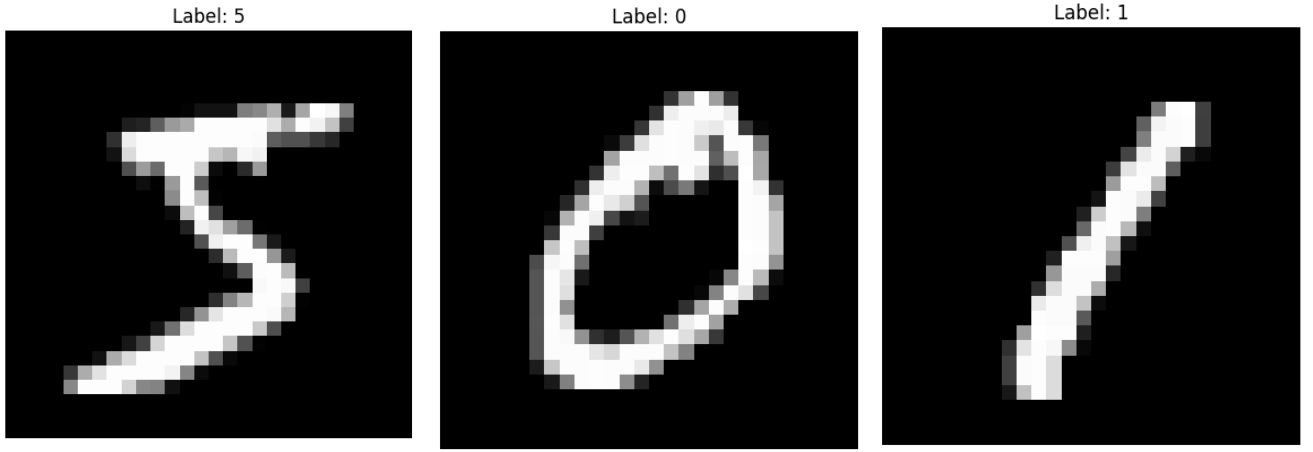
Изображения представляются в цифровой форме с использованием **пикселей**.

Каждый пиксель имеет определенный цвет, который может быть представлен в формате ЧБ (черно-белое изображение) или RGB (красный, зеленый, синий) или в других цветовых пространствах.

Значения цветовых каналов для каждого пикселя могут быть использованы как **признаки для обучения моделей**.

Размеры изображений могут варьироваться от небольших значений (например, 28x28 пикселей для MNIST) до высоких разрешений для фотографий.

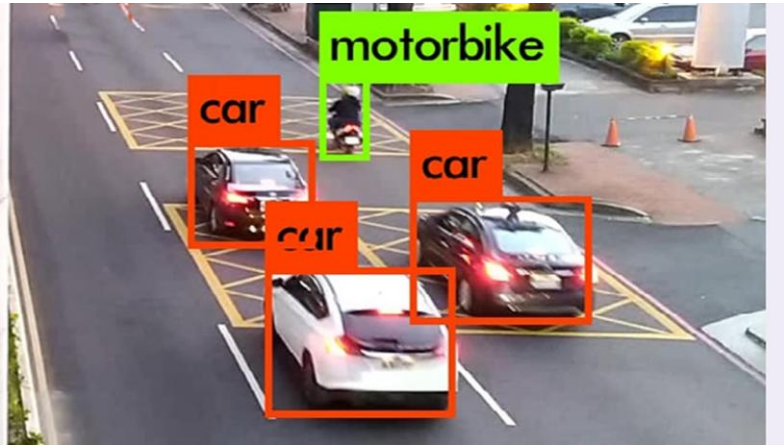


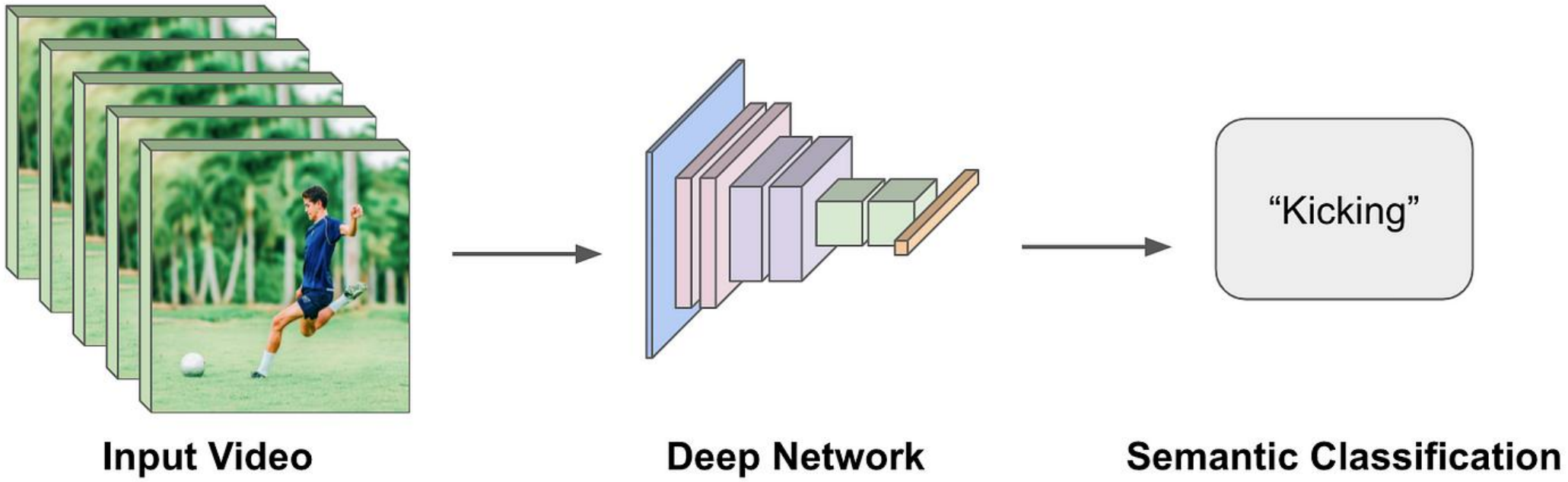


Предобработка данных

- **Также:**
  - Видео – последовательность изображений

Bounding boxes

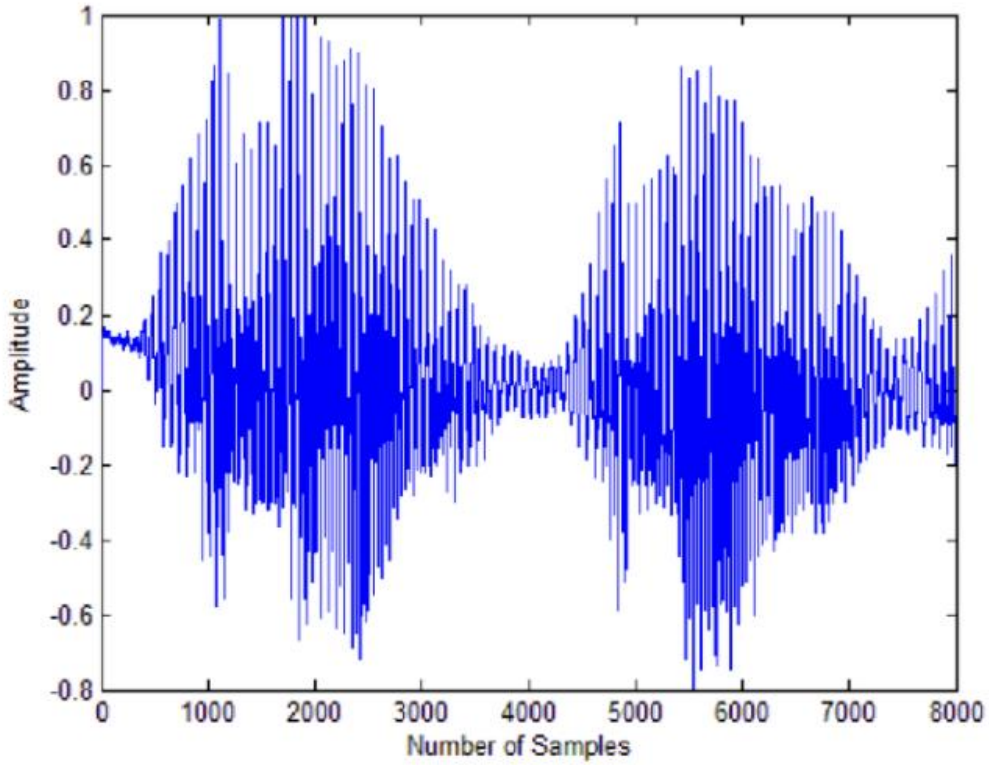


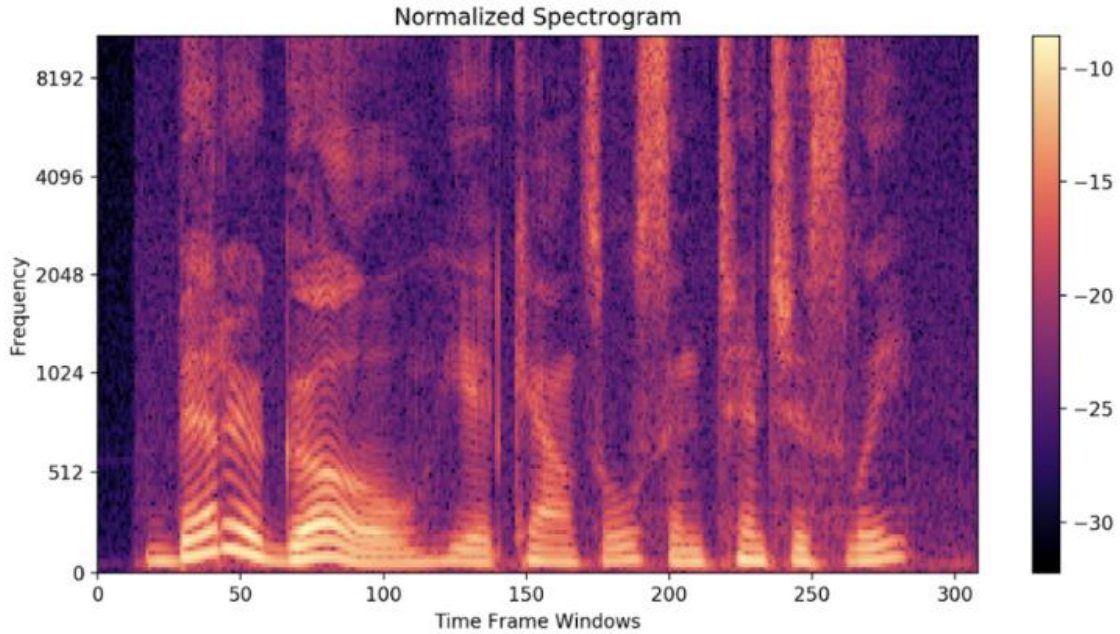


Предобработка данных

- **Также:**
  - Звук

Преобразование в изображение (преобразование Фурье)





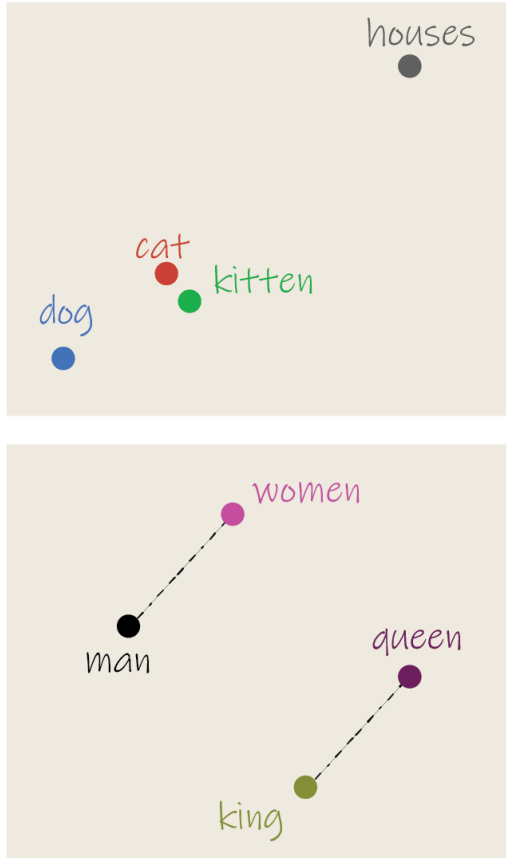
Предобработка данных

- **Также:**
  - Тексты

	living being	feline	human	gender	royalty	verb	plural
cat -->	0.6	0.9	0.1	0.4	-0.7	-0.3	-0.2
kitten -->	0.5	0.8	-0.1	0.2	-0.6	-0.5	-0.1
dog -->	0.7	-0.1	0.4	0.3	-0.4	-0.1	-0.3
houses -->	0.8	-0.4	0.5	0.1	-0.9	0.3	0.8
man -->	0.6	-0.2	0.8	0.9	-0.1	-0.9	-0.7
women -->	0.7	0.3	0.9	-0.7	0.1	-0.5	-0.4
king -->	0.5	-0.4	0.7	0.8	-0.9	-0.7	-0.6
queen -->	0.8	-0.1	0.8	-0.9	0.8	-0.5	-0.9

Dimensionality reduction of word embeddings from 7D to2D

Dimensionality reduction of word embeddings from 7D to 2D



Word

Word Embedding

Dimentionality Reduction

Visualization Of Word Embeddings In 2D

Предобработка данных

- **Рассмотрим предобработку для классических типов данных:**
  - Числовые
  - Категориальные

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Проверка пропущенных значений:NaN

- **Пропущенные значения – NaN (np.nan)**
- **NaN превращает любую операцию с собой в NaN**

- Могут быть и у числовых, и у категориальных данных:

	Name	Age	Gender	Seat Class	Ticket Price
0	John Doe	32.0	Male	Business	1000
1	Jane Smith	45.0	Female	Economy	500
2	Bob Johnson	NaN	Male	Economy	450
3	Susan Williams	28.0	Female	NaN	600

Проверка пропущенных значений:NaN

df.isna()

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
887	False	False	False	False	False	False	False
888	False	False	False	False	False	False	False
889	False	False	False	False	True	False	False
890	False	False	False	False	False	False	False
891	False	False	False	False	False	False	False

df.isna().sum(axis=0)

	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0

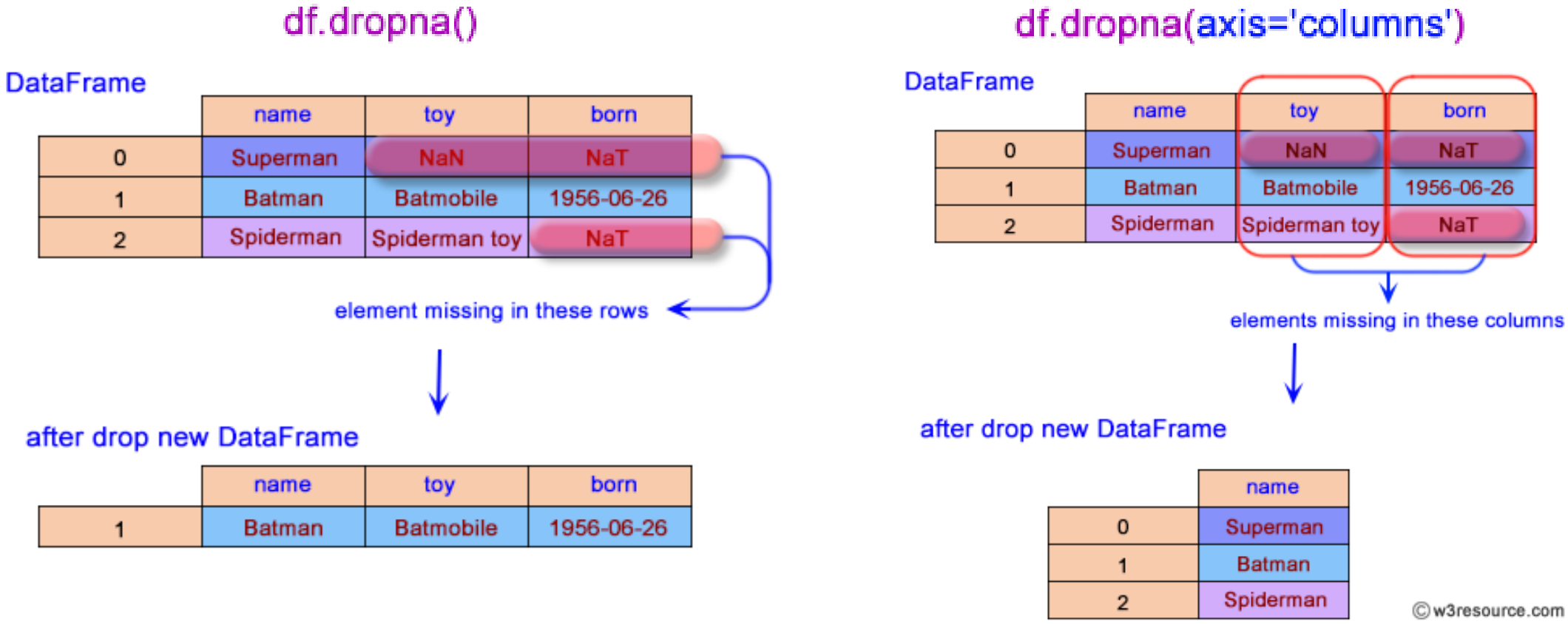
df.isna().sum(axis=1)

PassengerId	0
1	1
2	0
3	1
4	0
5	1
...	...
887	1
888	0
889	2
890	0
891	1

Проверка пропущенных значений:NaN

- **Пути решения:**

1. **Удаление строк или столбцов:** В Pandas, для удаления строк с отсутствующими значениями, можно использовать df.dropna(), а для удаления столбцов - df.dropna(axis='columns') или df.dropna(axis=1).



Проверка пропущенных значений:NaN

2. **Заполнение значениями:**

Чаще всего для этого используются статистики элементов в столбце, в котором мы заполняем пропуски: например, среднее, медиана или мода:

- **Мода** - значение изменяемого признака, которое встречается максимально часто.
- **Медиана** - значение признака, которое делит упорядоченное множество данных пополам.
- **Среднее** - среднее арифметическое, т.е. сумма всех значений признака, деленная на количество значений признака.

Если распределение симметрично, унимодально (имеет только одну моду) и не имеет заметных выбросов, то все три меры примерно дадут одинаковое значение. Если же ассиметрично или имеет заметные выбросы, лучше ориентироваться на моду или медиану.

Для **числовых признаков** часто используется **медиана**, для **категориальных — мода**.

Проверка пропущенных значений:NaN

- Реализация в коде

```
mean_Age = np.mean(df['Age'].dropna().values)
median_Age = np.quantile(df['Age'].dropna().values,q=0.5)
print(f"Среднее = {mean_Age}")
print(f"Медиана = {median_Age}")
```

```
Среднее = 29.69911764705882
Медиана = 28.0
```

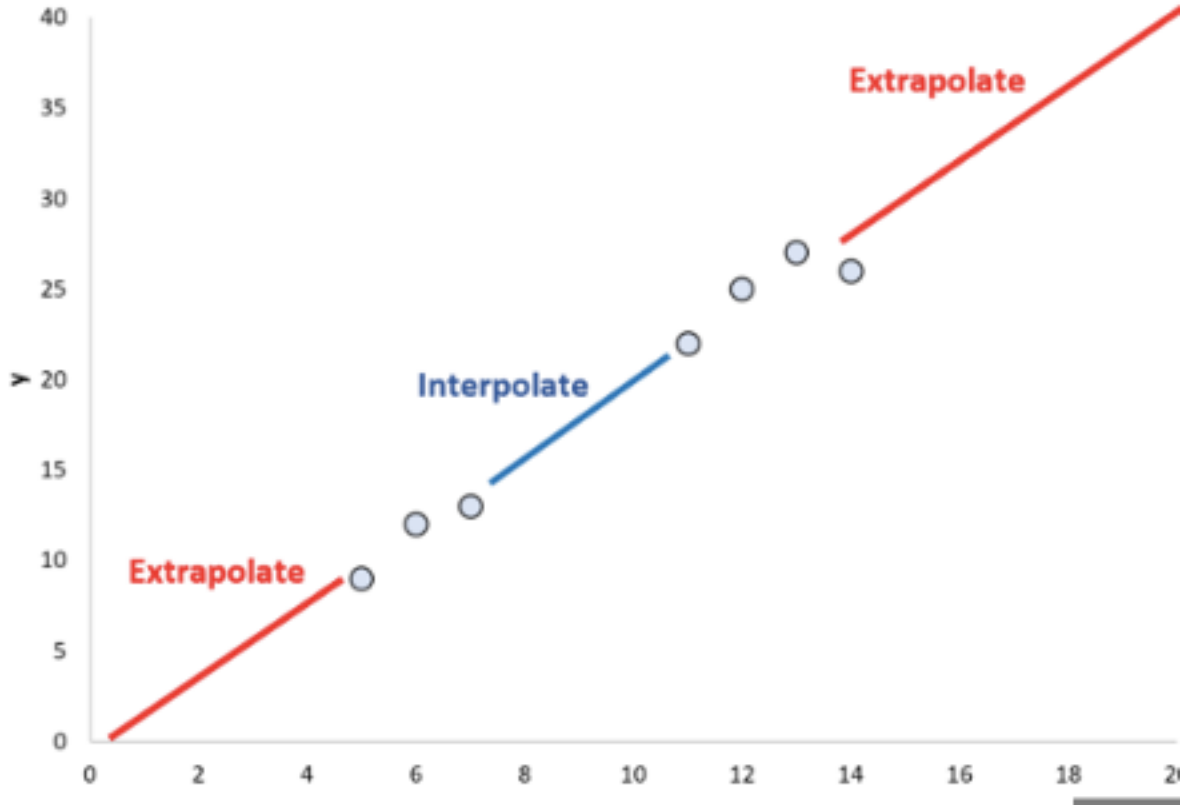
```
filled_df = df['Age'].fillna(mean_Age)
```

```
filled_df.isna().sum()
```

```
0
```

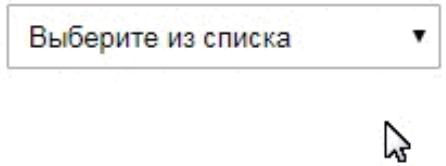
Проверка пропущенных значений:NaN

3. **Интерполяция:** Для временных рядов и числовых данных вы можете использовать метод интерполяции для заполнения отсутствующих значений на основе соседних значений.



Проверка пропущенных значений:NaN

4. В случае самостоятельного сбора данных – Валидация.



Числовые признаки

Обработка числовых признаков

Для многих моделей машинного обучения важно, чтобы количественные данные имели **одинаковый масштаб** (same scale).

Это справедливо для:

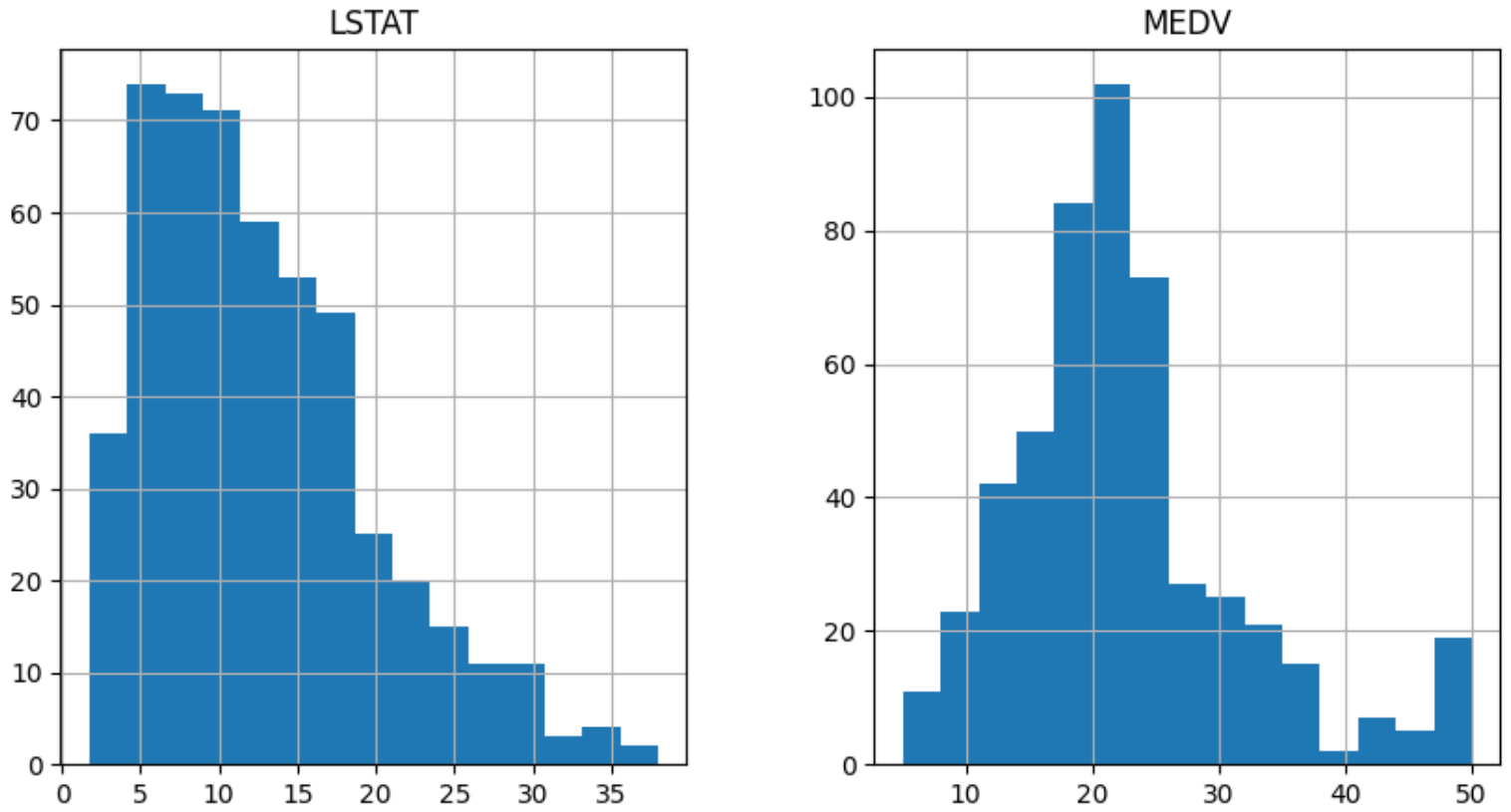
- алгоритмов, рассчитывающих *расстояние* (например, алгоритма k-ближайших соседей или метода k-средних); а также
- моделей, оптимизирующих веса *методом градиентного спуска* и использующих *регуляризацию* (в частности, это линейная или логистическая регрессия).

Обработка числовых признаков

```
# возьмем признак LSTAT (процент населения с низким социальным статусом)
# и целевую переменную MEDV (медианная стоимость жилья)
boston = df_housing[['LSTAT', 'MEDV']]
boston.shape
```

```
(506, 2)
```

```
boston.hist(bins = 15, figsize = (10, 5));
```




```
boston.describe()
```

	LSTAT	MEDV
count	506.000000	506.000000
mean	12.653063	22.532806
std	7.141062	9.197104
min	1.730000	5.000000
25%	6.950000	17.025000
50%	11.360000	21.200000
75%	16.955000	25.000000
max	37.970000	50.000000

Обработка числовых признаков

```
# создадим два отличающихся наблюдения
outliers = pd.DataFrame({
    'LSTAT': [45, 50],
    'MEDV': [70, 72]
    })

# добавим их в исходный датафрейм
boston_outlier = pd.concat([boston, outliers], ignore_index = True)

# посмотрим на размерность нового датафрейма
boston_outlier.shape
```

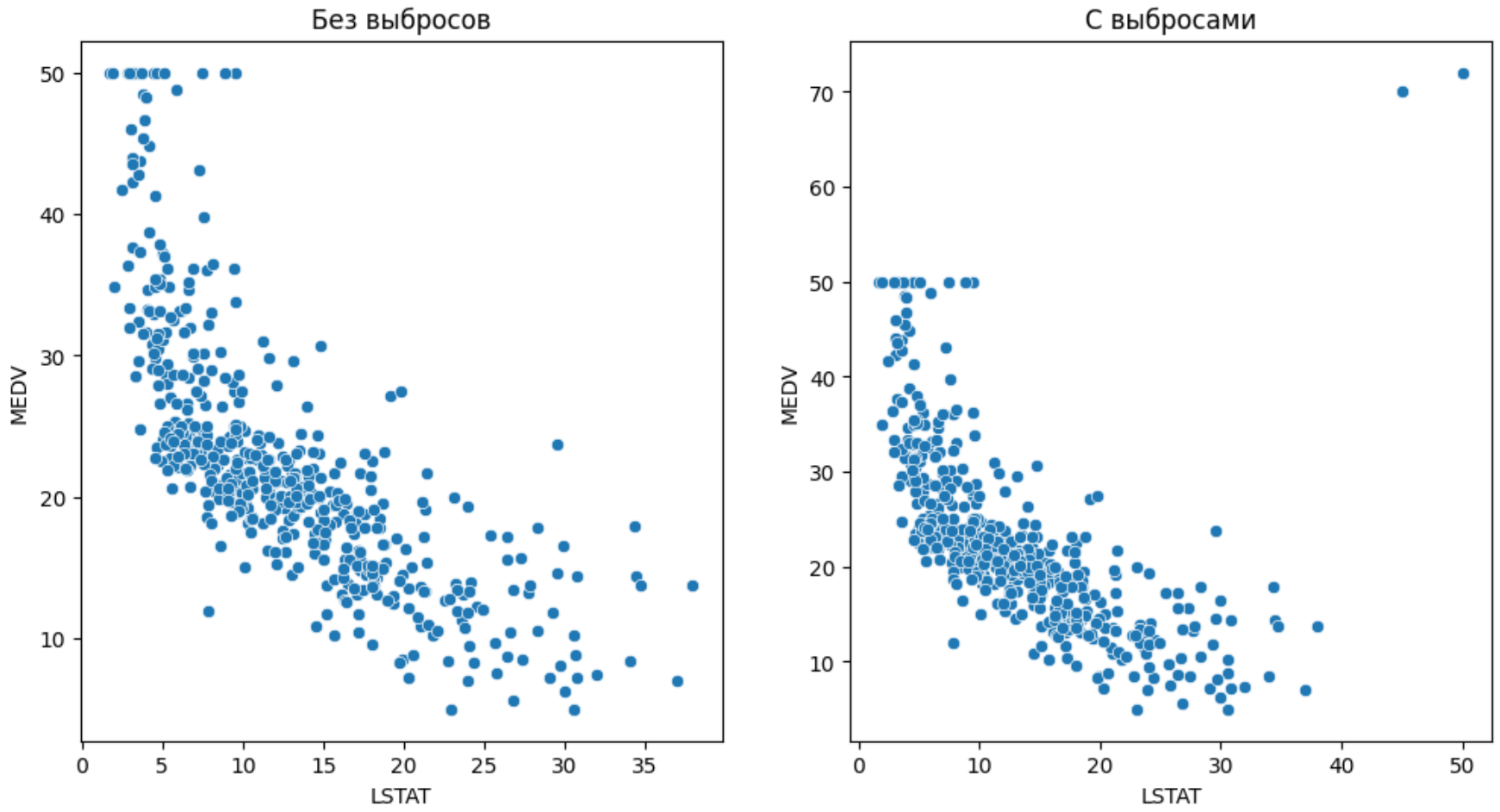
```
(508, 2)
```

```
# убедимся, что наблюдения добавились
boston_outlier.tail()
```

	LSTAT	MEDV
503	5.64	23.9
504	6.48	22.0
505	7.88	11.9
506	45.00	70.0
507	50.00	72.0

Обработка числовых признаков

```
fig, ax = plt.subplots(1, 2, figsize = (12,6))

sns.scatterplot(data = boston, x = 'LSTAT', y = 'MEDV', ax = ax[0]).set(title = 'Без выбросов')
sns.scatterplot(data = boston_outlier, x = 'LSTAT', y = 'MEDV', ax = ax[1]).set(title = 'С выбросами');
```



Обработка числовых признаков

- **Стандартизация**
- Если данные следуют нормальному или близкому к нормальному распределению (что желательно для многих моделей ML), имеет смысл прибегнуть к **стандартизации** (standartazation): то есть *приведению к нулевому среднему значению и единичному СКО* (так называемое стандартное нормальное распределение).

$$X' = \frac{X - \mu}{\sigma}$$

```
((boston - boston.mean()) / boston.std()).head(3)
```

	LSTAT	MEDV
0	-1.074499	0.159528
1	-0.491953	-0.101424
2	-1.207532	1.322937

Обработка числовых признаков

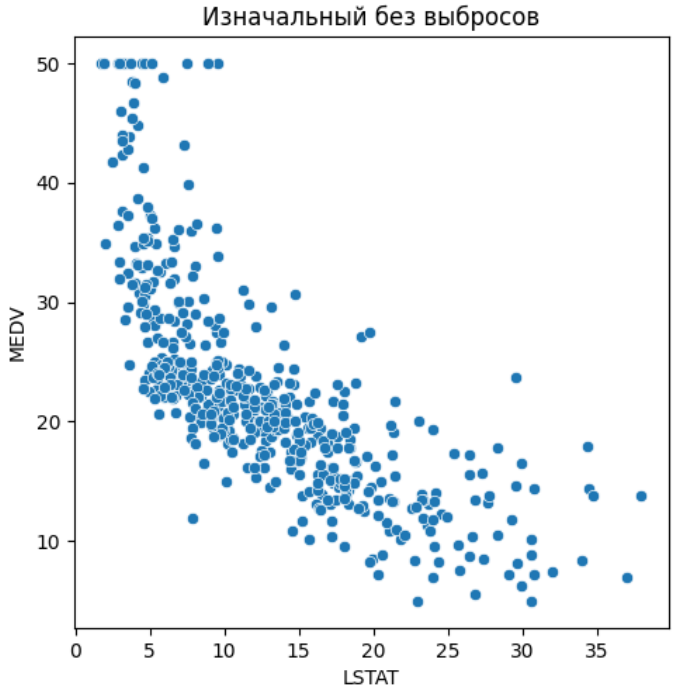
```
# из модуля preprocessing импортируем класс StandardScaler
from sklearn.preprocessing import StandardScaler

# создадим объект класса StandardScaler и применим метод .fit()
st_scaler = StandardScaler().fit(boston)
st_scaler
```

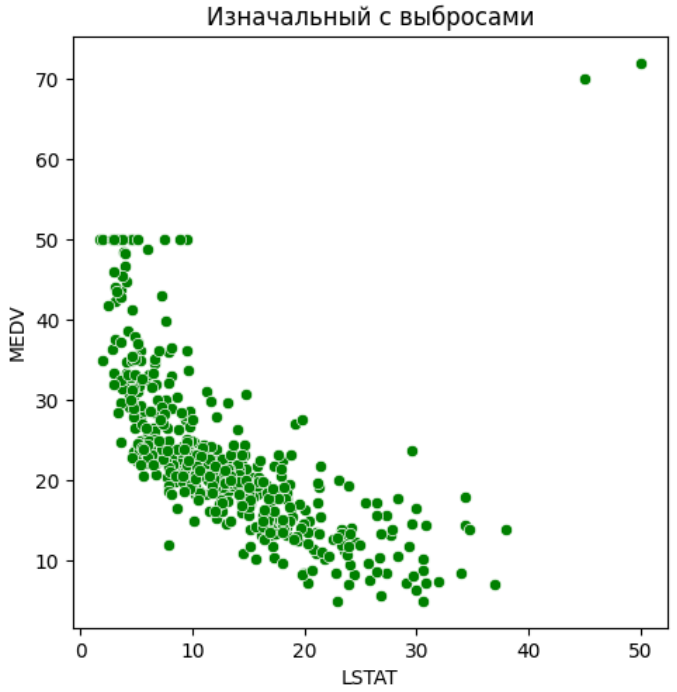
StandardScaler

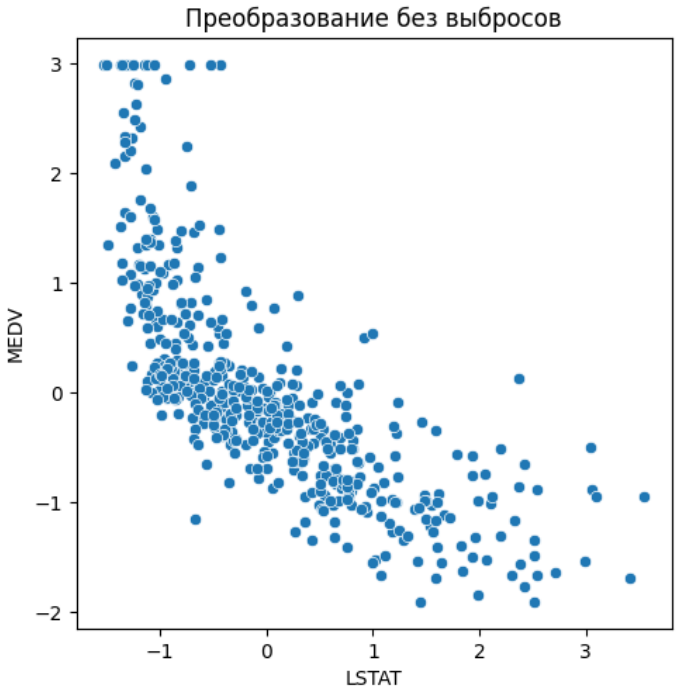
StandardScaler()

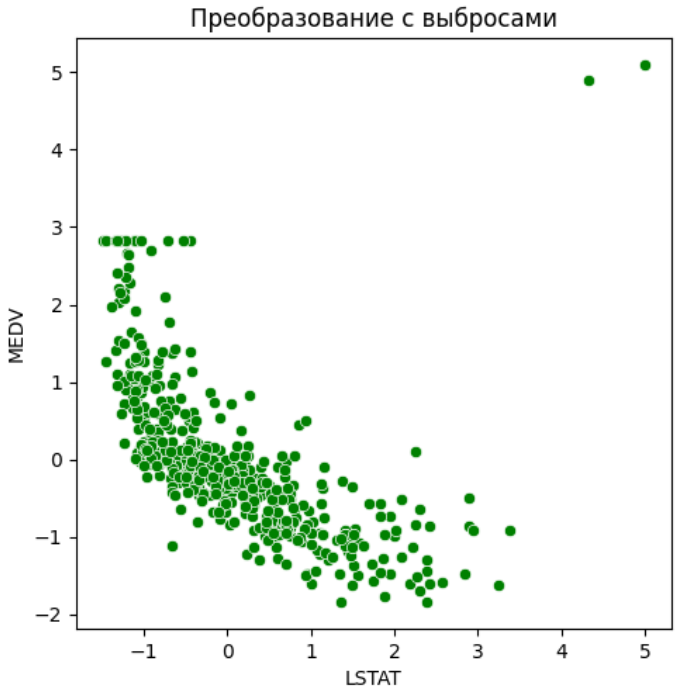
```
# метод .transform() возвращает массив Numpy с преобразованными значениями
boston_scaled = st_scaler.transform(boston)

# превратим массив в датафрейм с помощью функции pd.DataFrame()
pd.DataFrame(boston_scaled, columns = boston.columns).head(3)
```

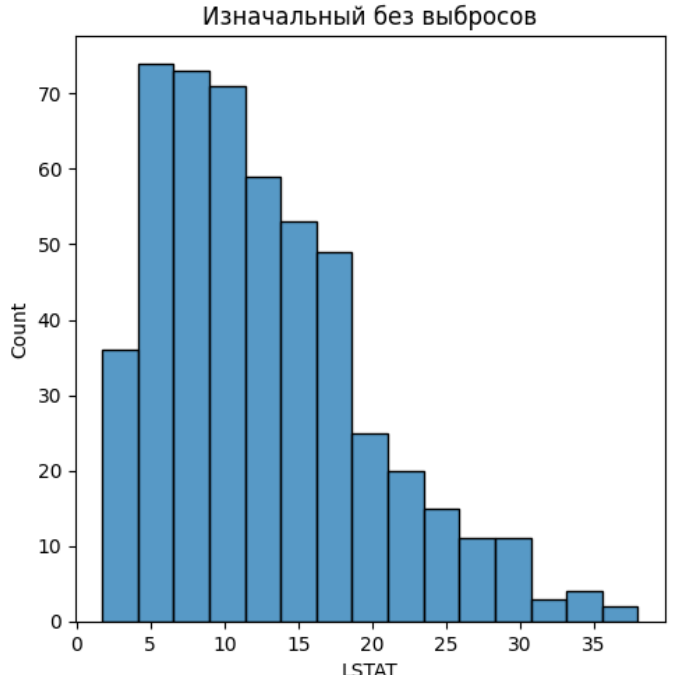
	LSTAT	MEDV
0	-1.075562	0.159686
1	-0.492439	-0.101524
2	-1.208727	1.324247

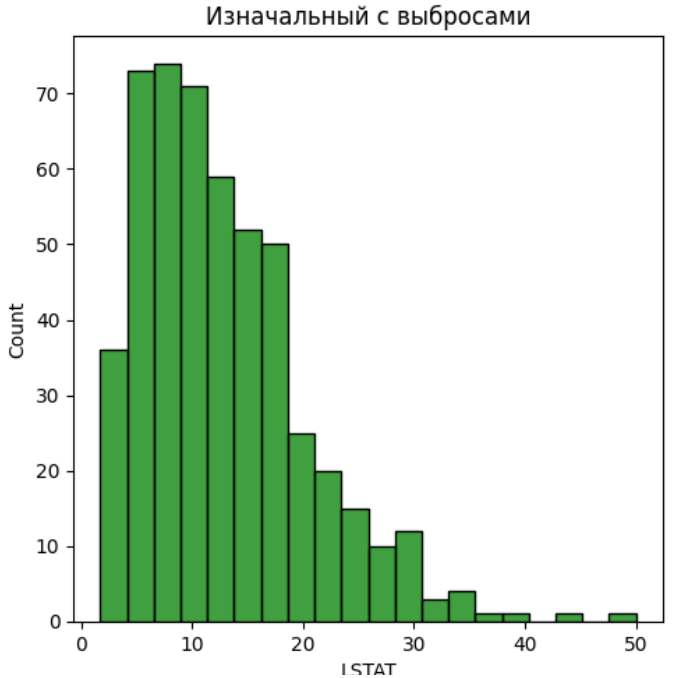

Изначальный без выбросов
MEDV
LSTAT
50
40
30
20
10
0
5
10
15
20
25
30
35

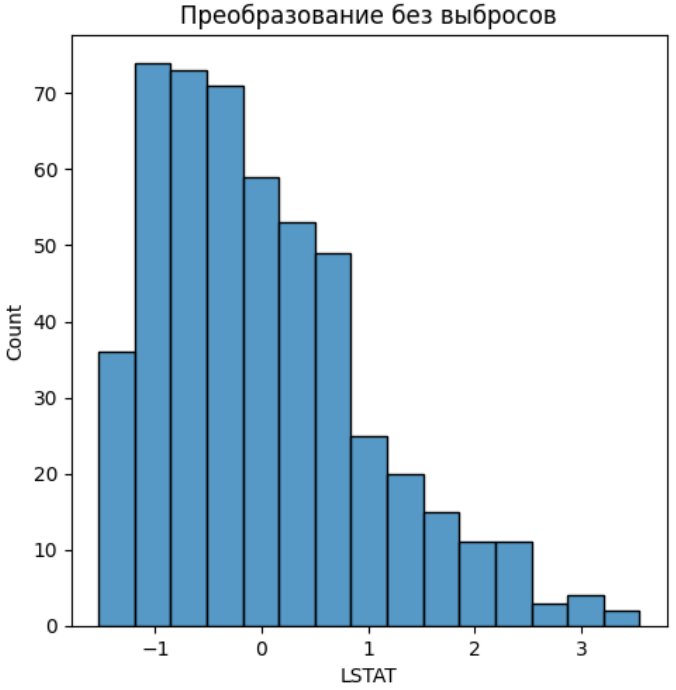

Изначальный с выбросами
MEDV
LSTAT
70
60
50
40
30
20
10
0
10
20
30
40
50

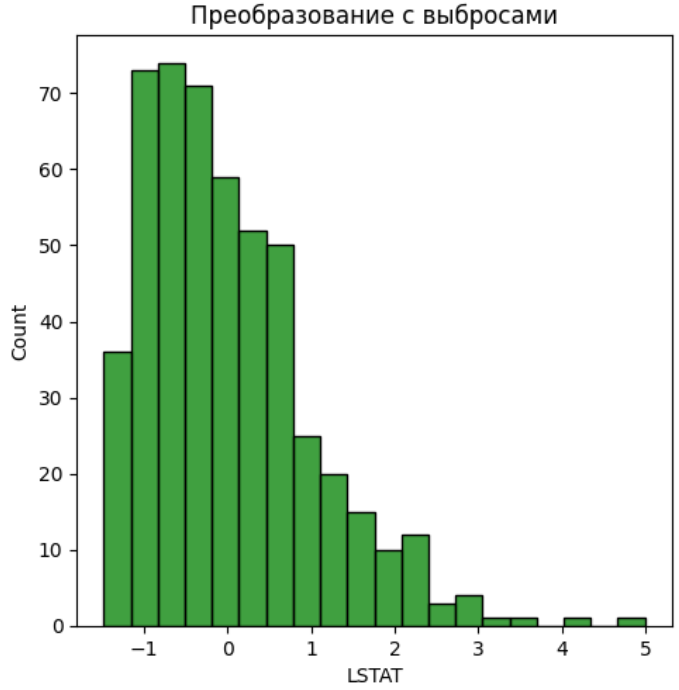

Преобразование без выбросов
MEDV
LSTAT
3
2
1
0
−1
−2
−1
0
1
2
3


Преобразование с выбросами
MEDV
LSTAT
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5


Изначальный без выбросов
Count
LSTAT


Изначальный с выбросами
Count
LSTAT


Преобразование без выбросов
Count
LSTAT


Преобразование с выбросами
Count
LSTAT

Обработка числовых признаков

- Обратите внимание, что стандартизация не ограничивает данные определенным диапазоном и допускает отрицательные значения.
- Метод чувствителен к выбросам в том смысле, что влияет на расчет СКО, и диапазон двух признаков с выбросами после стандартизации все равно будет различаться.
- Как следствие, **при наличии выбросов стандартизация не гарантирует одинаковый масштаб признаков.**

Обработка числовых признаков

- **Приведение к диапазону**
- **Приведение признаков к заданному диапазону** (scaling features to a range) является альтернативой стандартизации в тех случаях, когда нормальное распределение не является условием для обучения алгоритма.

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
# импортируем класс MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

# создаем объект этого класса,
# в параметре feature_range оставим диапазон по умолчанию
minmax = MinMaxScaler(feature_range = (0, 1))
minmax
```

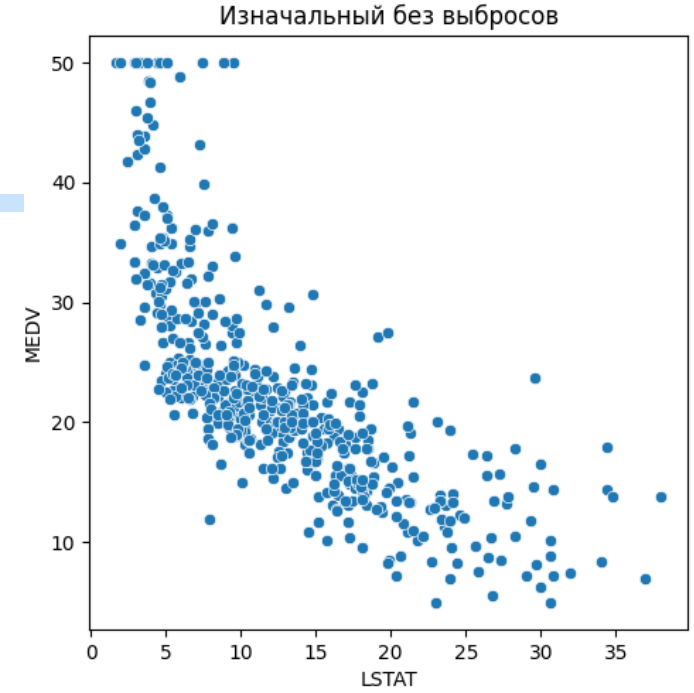
```
MinMaxScaler
MinMaxScaler()
```

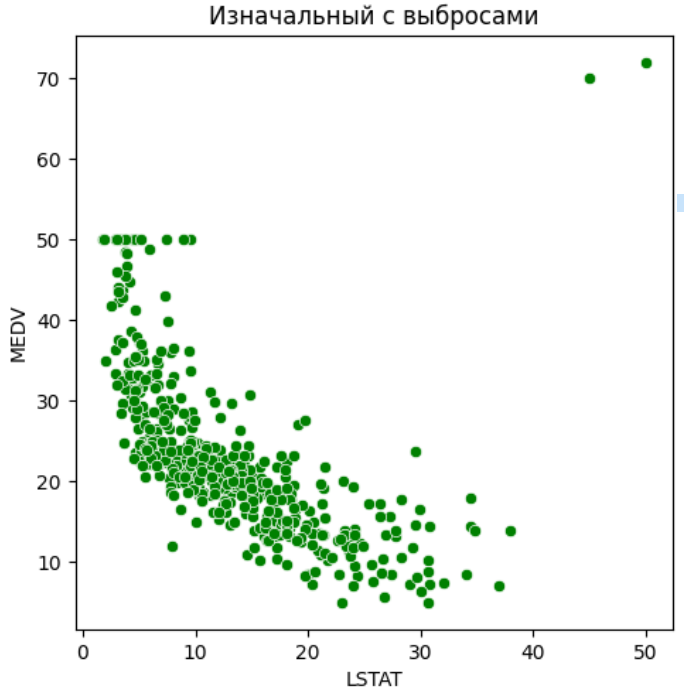
Обработка числовых признаков

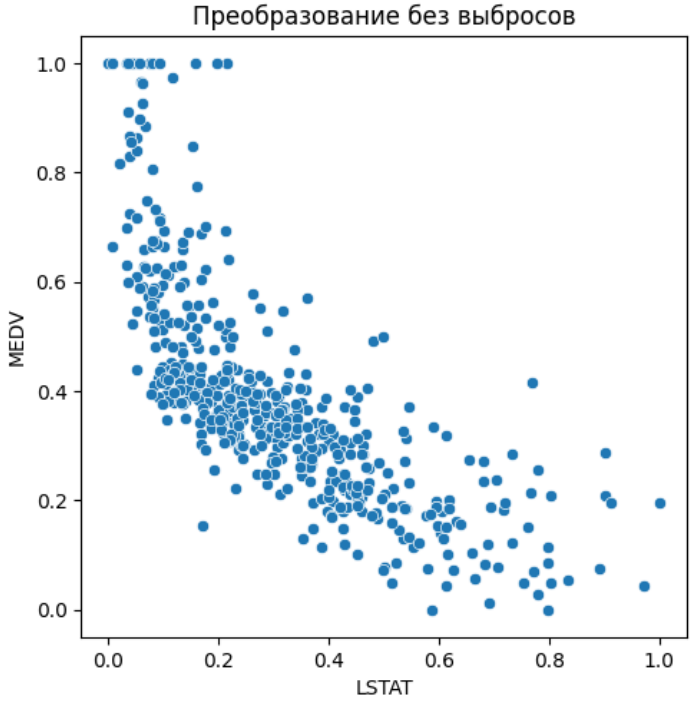
```
# применим метод .fit() и
minmax.fit(boston)

# найдем минимальные и максимальные значения
minmax.data_min_, minmax.data_max_
```

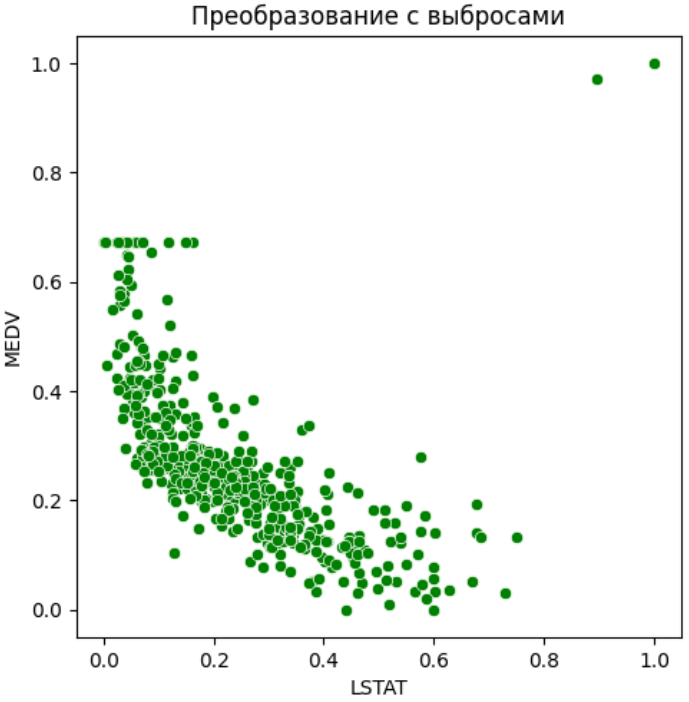
```
(array([1.73, 5.  ]), array([37.97, 50.  ]))
```

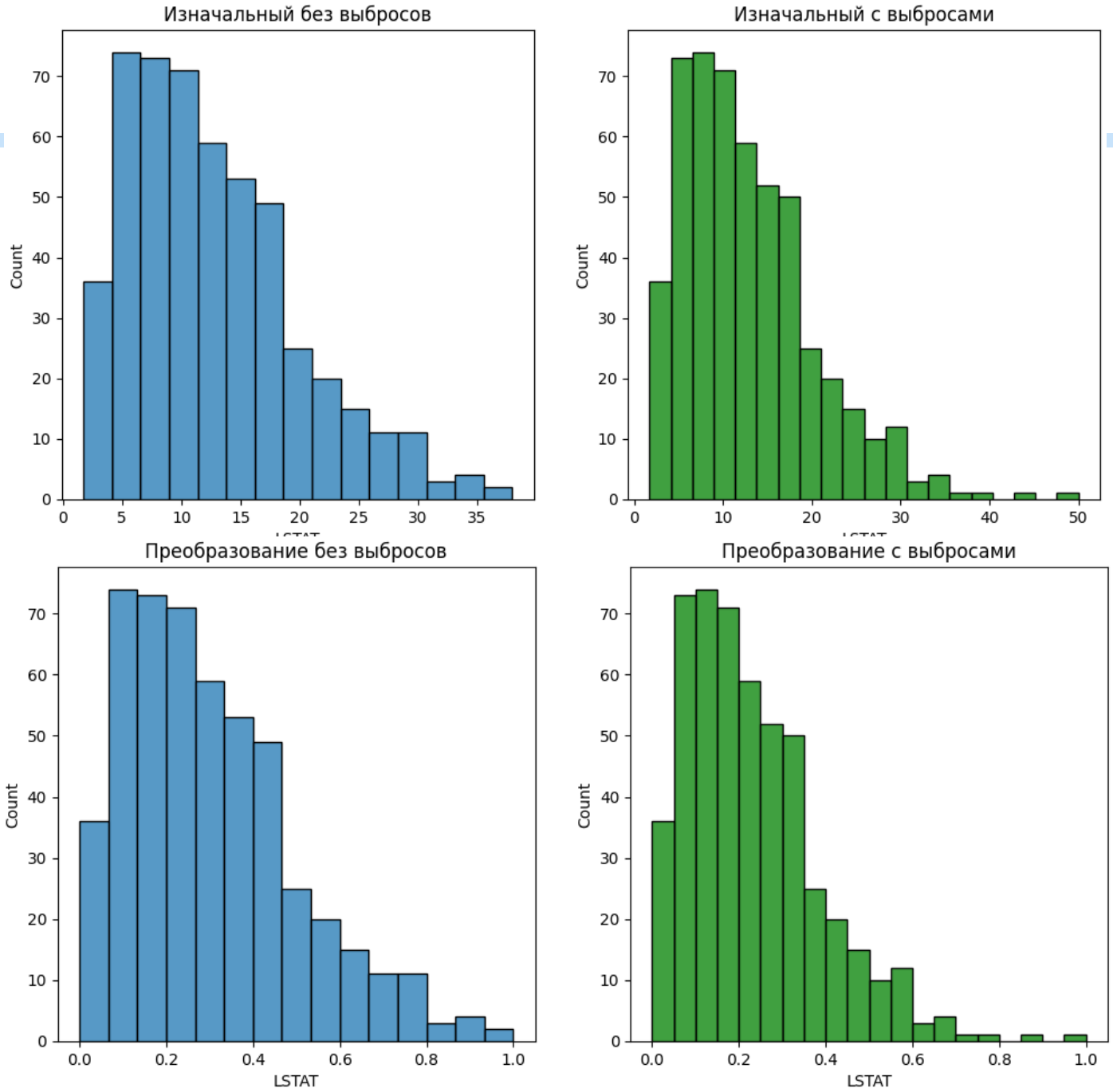
```
# приведем данные без выбросов (достаточно метода .transform())
boston_scaled = minmax.transform(boston)
# и с выбросами к заданному диапазону
boston_outlier_scaled = minmax.fit_transform(boston_outlier)

# преобразуем результаты в датафрейм
boston_scaled = pd.DataFrame(boston_scaled, columns = boston.columns)
boston_outlier_scaled = pd.DataFrame(boston_outlier_scaled, columns = boston.columns)
```


Изначальный без выбросов
MEDV
LSTAT
0
5
10
15
20
25
30
35
10
20
30
40
50


Изначальный с выбросами
MEDV
LSTAT
0
10
20
30
40
50
10
20
30
40
50
60
70


Преобразование без выбросов
MEDV
LSTAT
0.0
0.2
0.4
0.6
0.8
1.0


Преобразование с выбросами
MEDV
LSTAT
0.0
0.2
0.4
0.6
0.8
1.0


Изначальный без выбросов
Изначальный с выбросами
Преобразование без выбросов
Преобразование с выбросами
Count
LSTAT

Обработка категориальных признаков

Категориальные признаки

Обработка категориальных признаков

- **Определить порядок:** отсортировать значения

XXS < XS < S < M < L < XL < XXL

0 < 1 < 2 < 3 < 4 < 5 < 6

Обработка категориальных признаков

```
iris = sns.load_dataset("iris")
```

```
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

ее: Посмотреть рекомендованные графики | New interactive sheet

```
iris['species'].unique()
```

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
iris['species']= label_encoder.fit_transform(iris['species'])

iris['species'].unique()
```

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

```
iris.sample(5)
```

	sepal_length	sepal_width	petal_length	petal_width	species
1	4.9	3.0	1.4	0.2	0
43	5.0	3.5	1.6	0.6	0
97	6.2	2.9	4.3	1.3	1
94	5.6	2.7	4.2	1.3	1
104	6.5	3.0	5.8	2.2	2

Обработка категориальных признаков

- **Как быть с номинальными значениями, для которых нельзя определить отношения порядка?**

	Район
0	Манхэттен
1	Бруклин
2	Квинс
3	Бронкс
4	Ста́тен-Айленд

Обработка категориальных признаков

- **One-hot encoding:** превращает классы в вектор с одной единицей (эффективен при сравнительно малом числе классов)

	Район
0	Манхэттен
1	Бруклин
2	Квинс
3	Бронкс
4	Ста́тен-Айленд

	Район_Бронкс	Район_Бруклин	Район_Квинс	Район_Манхэттен	Район_Ста́тен-Айленд
0	0	0	0	1	0
1	0	1	0	0	0
2	0	0	1	0	0
3	1	0	0	0	0
4	0	0	0	0	1

Обработка категориальных признаков

```
unprocessed_cat_features
```

```
['Sex', 'Embarked', 'Title']
```

```
df[unprocessed_cat_features]
```

PassengerId	Sex	Embarked	Title
2	female	C	Mrs
4	female	S	Mrs
7	male	S	Mr
11	female	S	Miss
12	female	S	Miss
...	...	...	...
872	female	S	Mrs
873	male	S	Mr
880	female	C	Mrs
888	female	S	Miss
890	male	C	Mr

```
from sklearn.preprocessing import OneHotEncoder

#Initialize OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)

# Apply one-hot encoding to the categorical columns
one_hot_encoded = encoder.fit_transform(df[unprocessed_cat_features])

#Create a DataFrame with the one-hot encoded columns
#We use get_feature_names_out() to get the column names for the encoded data
one_hot_df = pd.DataFrame(one_hot_encoded, columns=encoder.get_feature_names_out(unprocessed_cat_features), index=df.index
                          )
```

```
one_hot_df
```

PassengerId	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	Title_Master	Title_Miss	Title_Mr	Title_Mrs	Title_Rare
2	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
4	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
7	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
11	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
12	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0

Высокоуровневая работа с данными

- **Агрегация**
- **Деагрегация**
- **Обогащение**

  Классический пример - добавление географических данных:

  Если у вас есть данные о клиентах и их почтовых адресах, вы можете обогатить эти данные, добавив информацию о географических координатах, чтобы знать, где находятся ваши клиенты.

Высокоуровневая работа с данными

```
df['Name']
```

PassengerId	Name
2	Cumings, Mrs. John Bradley (Florence Briggs Th...
4	Futrelle, Mrs. Jacques Heath (Lily May Peel)
7	McCarthy, Mr. Timothy J
11	Sandstrom, Miss. Marguerite Rut
12	Bonnell, Miss. Elizabeth
...	...
872	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)
873	Carlsson, Mr. Frans Olof
880	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)
888	Graham, Miss. Margaret Edith
890	Behr, Mr. Karl Howell

```
df['Title'] = df.Name.str.extract(' ([A-Za-z]+)\.', expand=False)

pd.crosstab(df['Title'], df['Sex'])
```

Sex	female	male
Title		
Capt	0	1
Col	0	1
Countess	1	0
Dr	1	2
Lady	1	0
Major	0	2
Master	0	7
Miss	44	0
Mlle	2	0
Mme	1	0
Mr	0	81
Mrs	38	0
Sir	0	1

Спасибо за внимание!