

# Выбросы в данных

[Материалы](#) > [Анализ и обработка данных](#)

Отдельным вопросом работы с количественными данными являются **выбросы** (outliers), которые существенно влияют на многие статистические показатели, мешают масштабировать данные и ухудшают качество моделей машинного обучения.

## Про выбросы

### Понятие и причины выбросов

Выбросы — это данные, которые сильно отличаются от общего распределения. При этом можно выделить:

- ошибочно возникающие выбросы:
  - человеческий фактор, ошибка ввода данных;
  - погрешности измерения;
  - ошибка эксперимента (например, шум при записи голоса);
  - ошибка обработки;
  - ошибка получения выборки (sampling error);
- естественные выбросы:
  - например, высокий человек, разовая большая покупка;
  - аномально низкая цена на отдельный объект недвижимости.

Примечание. В последнем случае, хотя с точки зрения модели такой выброс будет считаться нежелательным, имеет смысл проверить, с чем связана такая цена.

### Влияние выбросов

**Статистические показатели.** На занятии по [статистическому выводу](#) мы провели тест Стьюдента для того, чтобы определить на основе выборки вероятность того, что средний рост составляет 182 см.

Откроем блокнот к этому занятию📖

```

1 np.random.seed(42)
2 height = list(np.round(np.random.normal(180, 10, 1000)))
3 print(height)

```

```

1 [185.0, 179.0, 186.0, 195.0, 178.0, 178.0, 196.0, 188.0, 175.0, 185.0, 175.0, 18

```

Напомню, что исходя из данных мы смогли отвергнуть нулевую гипотезу, которая утверждала, что рост действительно составляет 182 см.

```

1 import scipy.stats as st
2 t_statistic, p_value = st.ttest_1samp(height, 182)
3 p_value

```

```

1 9.035492171563733e-09

```

Добавим выброс и повторно проверим гипотезу.

```

1 height.append(1000)
2
3 t_statistic, p_value = st.ttest_1samp(height, 182)
4 p_value

```

```

1 0.26334958447468043

```

Как мы видим, одно сильно отличающееся наблюдение изменило результаты теста.

**Масштабирование данных.** Как мы только что видели, сильно отличающиеся от остальных значения мешают качественному масштабированию данных.

**Модели ML.** Выбросы влияют на качество модели линейной регрессии. Возьмем третий набор данных из квартета Энскомба (Anscombe's quartet).

```

1 anscombe = pd.read_json('/content/sample_data/anscombe.json')
2 anscombe = anscombe[anscombe.Series == 'III']
3 anscombe.head()

```

	Series	X	Y
22	III	10	7.46
23	III	8	6.77
24	III	13	12.74
25	III	9	7.11
26	III	11	7.81

```

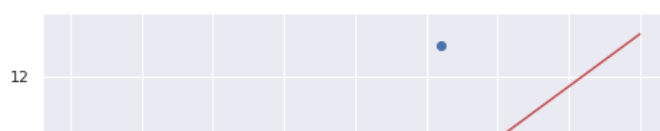
1 x, y = anscombe.X, anscombe.Y

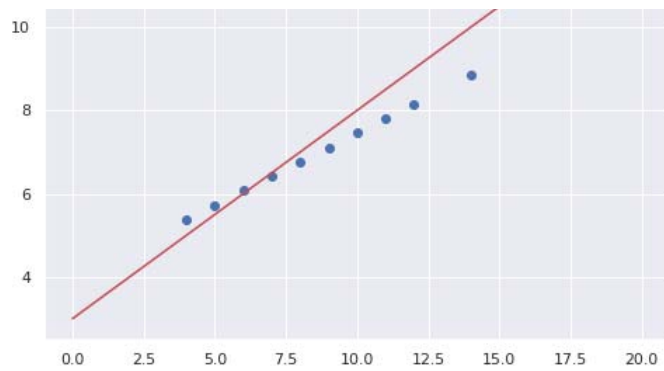
```

```

1 plt.scatter(x, y)
2
3 slope, intercept = np.polyfit(x, y, deg = 1)
4
5 x_vals = np.linspace(0, 20, num = 1000)
6 y_vals = intercept + slope * x_vals
7 plt.plot(x_vals, y_vals, 'r')
8
9 plt.show()

```





Посмотрим на коэффициент корреляции.

```
1 | np.corrcoef(x, y)[0][1]
```

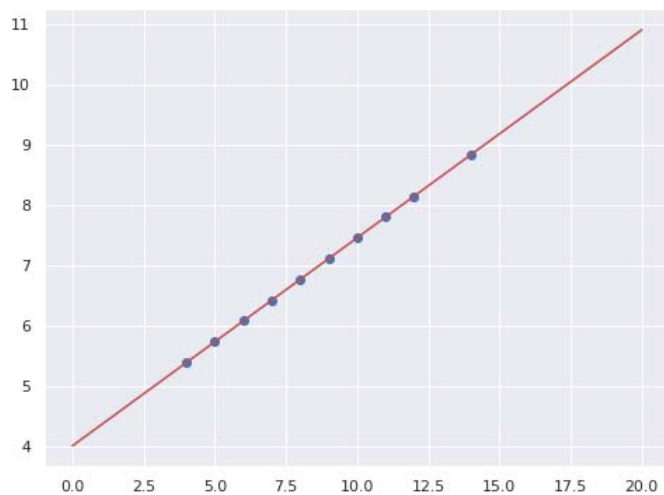
```
1 | 0.8162867394895984
```

Удалим выброс.

```
1 | # будем считать выбросом наблюдение с индексом 24
2 | x.drop(index = 24, inplace = True)
3 | y.drop(index = 24, inplace = True)
```

Вновь посмотрим на график и коэффициент корреляции.

```
1 | plt.scatter(x, y)
2 |
3 | slope, intercept = np.polyfit(x, y, deg = 1)
4 |
5 | x_vals = np.linspace(0, 20, num = 1000)
6 | y_vals = intercept + slope * x_vals
7 | plt.plot(x_vals, y_vals, 'r')
8 |
9 | plt.show()
```



```
1 | np.corrcoef(x, y)[0][1]
```

```
1 | 0.9999965537848283
```

Выбросы особенно сильны, когда мы располагаем небольшим количеством данных.

## Outlier vs. Novelty

Отличающееся наблюдение можно разделить на **выбросы** (outlier) и **новые отличающиеся наблюдения** (novelty). Выброс уже присутствует в данных. Другими словами, мы смотрим на данные и понимаем, что часть содержащихся в них наблюдений существенно отличаются от общей массы. Во втором случае, у нас уже есть набор данных, нас просят определить, является ли новое наблюдение выбросом или нет.

Так как для выявления уже присутствующих выбросов у нас нет никакой разметки, это *обучение без учителя*. Во втором случае, это *частичное обучение с учителем* (semi-supervised).

На практике это означает, что при использовании продвинутых алгоритмов для выявления выбросов, если речь идет о новых отличающихся наблюдениях (novelty), мы должны использовать `.fit()` на обучающей выборке, а `.predict()`, `.decision_function()` и `.score_samples()` на тестовой (про алгоритмы и эти методы поговорим дальше). Подробнее [здесь](#).

## Работа с выбросами

Начнем с более простых методов. Скачаем, подгрузим и импортируем данные.

boston.csv

Скачать

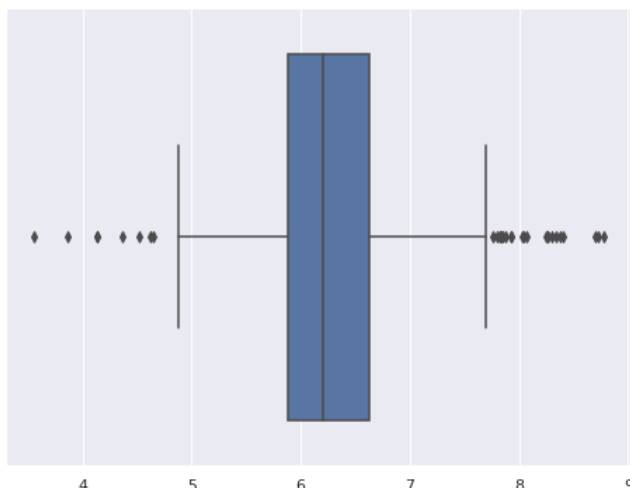
```
1 | boston = pd.read_csv('/content/boston.csv')
```

## Статистические методы

### boxplot

Выбросы можно увидеть на boxplot. По умолчанию, длина «усов» рассчитывается как  $1,5 \times IQR$ . Данные, которые выходят за их пределы — выбросы.

```
1 | sns.boxplot(x = boston.RM);
```

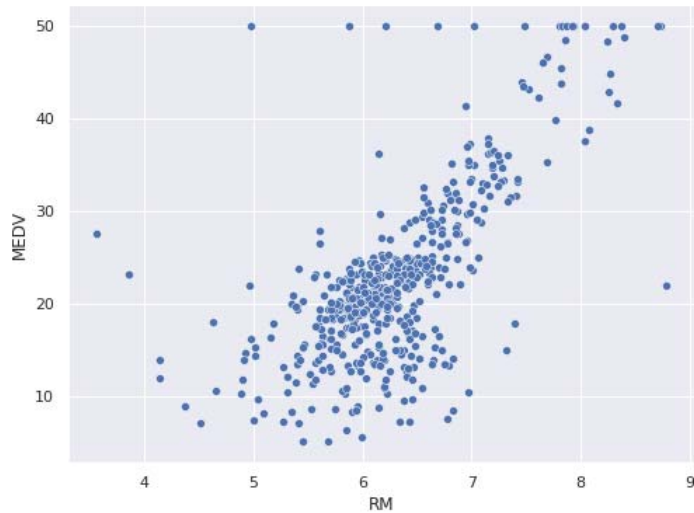


RM

## scatter plot

Кроме того, выбросы можно попытаться выявить на точечной диаграмме.

```
1 sns.scatterplot(x = boston.RM, y = boston.MEDV);
```



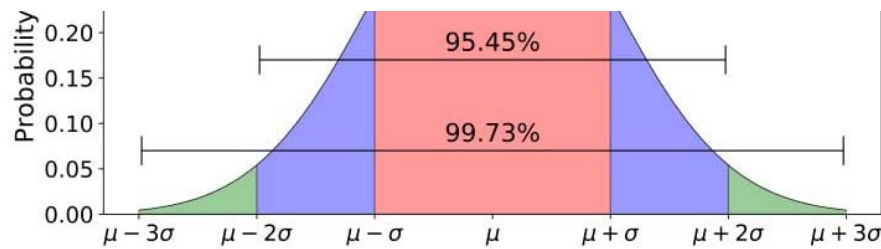
## z-score

Количественно выбросы можно найти через **стандартизированную оценку** (z-оценку, z-score). Эта оценка показывает на сколько средних квадратических отклонений значение отличается от среднего.

```
1 from scipy import stats
2
3 z = stats.zscore(boston)
4 z.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.982843
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.867883
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.867883
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.752922
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.752922

Так как мы знаем, что 99,7 процентов наблюдений лежат в пределах трех СКО от среднего, то можем предположить, что выбросами будут оставшиеся 0,3 процента.



Выведем эти значения.

```
1 # найдем те значения, которые отклоняются больше, чем на три СКО
2 # технически, метод .any() выводит True для тех строк (axis = 1),
3 # где хотя бы одно значение True (т.е. > 3)
4 boston[(np.abs(z) > 3).any(axis = 1)].head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
55	0.01311	90.0	1.22	0.0	0.403	7.249	21.9	8.6966	5.0	226.0	17.9	395.93	4.81	35.4
56	0.02055	85.0	0.74	0.0	0.410	6.383	35.7	9.1876	2.0	313.0	17.3	396.90	5.77	24.7
57	0.01432	100.0	1.32	0.0	0.411	6.816	40.5	8.3248	5.0	256.0	15.1	392.90	3.95	31.6
102	0.22876	0.0	8.56	0.0	0.520	6.405	85.4	2.7147	5.0	384.0	20.9	70.80	10.63	18.6
141	1.62864	0.0	21.89	0.0	0.624	5.019	100.0	1.4394	4.0	437.0	21.2	396.90	34.41	14.4

Посмотрим, как удалить выбросы в отдельном столбце.

```
1 # выведем True там, где в столбце RM значение меньше трех СКО
2 col_mask = np.where(np.abs(z.RM) < 3, True, False)
3
4 # применим маску к столбцу
5 boston.RM[col_mask].head()
```

```
1 0    6.575
2 1    6.421
3 2    7.185
4 3    6.998
5 4    7.147
6 Name: RM, dtype: float64
```

Теперь удалим выбросы во всем датафрейме.

```
1 # если в строке (axis = 1) есть хотя бы один False как следствие условия np.abs(z) < 3,
2 # метод .all() вернет логический массив, который можно использовать как фильтр
3 z_mask = (np.abs(z) < 3).all(axis = 1)
4
5 boston_z = boston[z_mask]
6 boston_z.shape
```

```
1 (415, 14)
```

Выведем корреляцию до и после удаления выбросов.

```
1 boston[['RM', 'MEDV']].corr()
```

	RM	MEDV
RM	1.00000	0.69536
MEDV	0.69536	1.00000

```
1 boston_z[['RM', 'MEDV']].corr()
```

	RM	MEDV
RM	1.000000	0.734041
MEDV	0.734041	1.000000

В данном случае корреляция увеличилась.

## Измененный z-score

Важно понимать, что z-score, который мы используем для идентификации выбросов сам по себе зависит от сильно отличающихся значений, поскольку для расчета используется среднее арифметическое и СКО.

$$z = \frac{x - \mu}{\sigma}$$

Вместо z-оценки можно использовать **измененную z-оценку** (modified z-score), которая использует метрики робастной статистики

$$z_{mod} = \frac{x - Q2}{MAD},$$

где MAD представляет собой *среднее абсолютное отклонение* (median absolute deviation) и рассчитывается по формуле  $MAD = \text{median}(|x - Q2|)$ . Заметим, что  $MAD = 0,6745\sigma$ .

Iglewicz и Hoaglin рекомендуют считать выбросами те значения, для которых  $|z_{mod}| > 3,5$ . Применим этот метод.

```

1 # рассчитаем MAD
2 median = boston.median()
3 dev_median = boston - (boston.median())
4 abs_dev_median = np.abs(dev_median)
5 MAD = abs_dev_median.median()
6
7 # рассчитаем измененный z-score
8 # добавим константу, чтобы избежать деления на ноль
9 zmod = (0.6745 * (boston - boston.median())) / (MAD + 1e-5)
10
11 # создадим фильтр
12 zmod_mask = (np.abs(zmod) < 3.5).all(axis = 1)
13
14 # выведем результат
15 boston_zmod = boston[zmod_mask]
16 boston_zmod.shape

```

```
1 (168, 14)
```

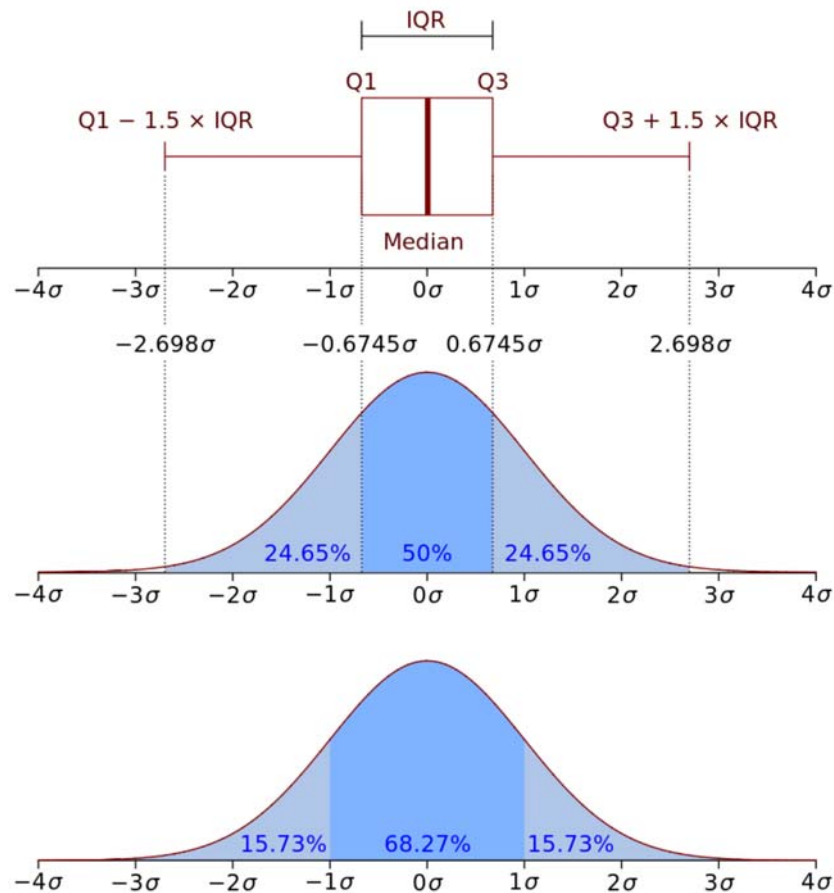
Обратите внимание, что в данном случае мы очень агрессивно удаляли значения. Посмотрим на корреляцию.

```
1 boston_zmod[['RM', 'MEDV']].corr().iloc[0, 1].round(3)
```

```
1 0.719
```

## 1.5 IQR

Еще одним распространенным способом выявления и удаления выбросов является правило  $1,5 \times IQR$ . Рассмотрим, почему именно 1,5? В стандартном нормальном распределении Q1 и Q3 соответствуют  $-0.6745\sigma$  и  $0.6745\sigma$ .



Зная эти показатели, мы можем рассчитать верхнюю и нижнюю границу.

```
1 q1 = -0.6745
2 q3 = 0.6745
3
4 iqr = q3 - q1
5
6 lower_bound = q1 - (1.5 * iqr)
7 upper_bound = q3 + (1.5 * iqr)
8
9 # тогда lower_bound и upper_bound почти равны трем СК0 от среднего
10 # (было бы точнее, если использовать 1.75)
11 print(lower_bound, upper_bound)
```

```
1 -2.698 2.698
```

Замечу, что для того чтобы этот метод нахождения выбросов был аналогичен z-score, было бы точнее использовать показатель  $1,75 \times IQR$ . При таком решении выбросами будет считаться большее количество наблюдений.

Найдем и удалим выбросы в столбце.

```
1 # найдем границы 1.5 * IQR
2 q1 = boston.RM.quantile(0.25)
3 q3 = boston.RM.quantile(0.75)
4
5 iqr = q3 - q1
6
```



```

7 | lower_bound = q1 - (1.5 * iqr)
8 | upper_bound = q3 + (1.5 * iqr)
9 |
10 | print(lower_bound, upper_bound)

```

```

1 | 4.778499999999999 7.730500000000001

```

```

1 | # применим эти границы, чтобы найти выбросы в столбце RM
2 | boston[(boston.RM < lower_bound) | (boston.RM > upper_bound)].head()

```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
97	0.12083	0.0	2.89	0.0	0.445	8.069	76.0	3.4952	2.0	276.0	18.0	396.90	4.21	38.7
98	0.08187	0.0	2.89	0.0	0.445	7.820	36.9	3.4952	2.0	276.0	18.0	393.53	3.57	43.8
162	1.83377	0.0	19.58	1.0	0.605	7.802	98.2	2.0407	5.0	403.0	14.7	389.61	1.92	50.0
163	1.51902	0.0	19.58	1.0	0.605	8.375	93.9	2.1620	5.0	403.0	14.7	388.45	3.32	50.0
166	2.01019	0.0	19.58	0.0	0.605	7.929	96.2	2.0459	5.0	403.0	14.7	369.30	3.70	50.0

```

1 | # найдем значения без выбросов (переворачиваем маску)
2 | boston[~(boston.RM < lower_bound) | (boston.RM > upper_bound)].head()

```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Найдем и удалим выбросы во всем датафрейме.

```

1 | # найдем границы 1.5 * IQR по каждому столбцу
2 | Q1 = boston.quantile(0.25)
3 | Q3 = boston.quantile(0.75)
4 | IQR = Q3 - Q1
5 | lower, upper = Q1 - 1.5 * IQR, Q3 + 1.5 * IQR
6 |
7 | # создадим маску для выбросов
8 | # если хотя бы один выброс в строке (True), метод .any() сделает всю строку True
9 | mask_out = ((boston < lower) | (boston > upper)).any(axis = 1)

```

```

1 | # найдем выбросы во всем датафрейме
2 | boston[mask_out].shape

```

```

1 | (238, 14)

```

```

1 | # возьмем датафрейм без выбросов
2 | boston[~mask_out].shape

```

```

1 | (268, 14)

```

```

1 | # обратное условие, если все значения по всем строкам внутри границ
2 | # метод .all() выдаст True
3 | mask_no_out = ((boston >= lower) & (boston <= upper)).all(axis = 1)

```

```

1 | # выведем датафрейм без выбросов
2 | boston[mask_no_out].shape

```

```

1 | (268, 14)

```

```

1 | # выведем выбросы
2 | boston[~mask_no_out].shape

```

```
1 | (238, 14)
```

```
1 | # сохраним результат
2 | boston_iqr = boston[mask_no_out]
```

```
1 | boston_iqr[['RM', 'MEDV']].corr()
```

	RM	MEDV
RM	1.000000	0.644819
MEDV	0.644819	1.000000

Как мы видим, несмотря на более активное удаление значений, коэффициент корреляции стал даже меньше, чем в изначальных данных.

## Методы, основанные на модели

Теперь посмотрим на методы выявления выбросов, основанные на модели (model-based approaches)

### Isolation Forest

**Изолирующий лес** (Isolation Forest или iForest) использует принципы решающего дерева (Decision Tree) и построенного на его основе ансамблевого метода случайного леса (Random Forest).

#### Принцип изолирующего леса

Принцип построения **изолирующего дерева** (Isolation Tree или iTree) довольно прост. Алгоритм случайным образом выбирает признак, затем в пределах диапазона этого признака случайно выбирает разделяющую границу (split). Та часть наблюдений, которая меньше либо равна этой границе, оказывается в левом потомке (left child node), та, которая больше — в правом (right child node). Затем процесс рекурсивно повторяется.

Что интересно, в получившейся древовидной структуре путь (то есть количество сплитов) от корневого узла (root node) до выброса будет существенно короче, чем до обычного наблюдения. Это логично поскольку, например, в задаче классификации с помощью решающего дерева в первую очередь удастся отделить тот класс, который наиболее «удален» от остальных.

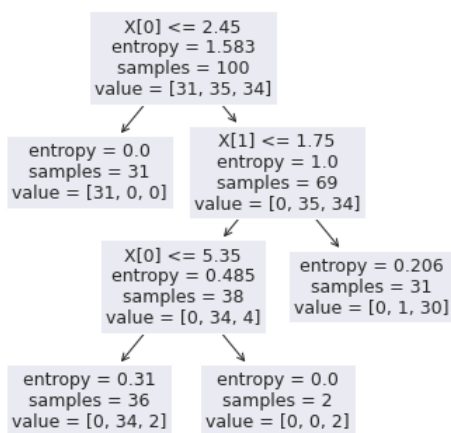
Приведем классификацию с помощью решающего дерева на примере датасета цветов ириса.

```
1 | from sklearn.tree import DecisionTreeClassifier
2 | from sklearn import tree
3 |
4 | from sklearn.datasets import load_iris
5 | iris = load_iris()
6 |
7 | df = pd.DataFrame(iris.data[:, [2, 3]], columns = ['petal_l', 'petal_w'])
8 | df['target'] = iris.target
9 |
10 | X = df[['petal_l', 'petal_w']]
```

```

11 y = df.target
12
13 from sklearn.model_selection import train_test_split
14
15 X_train, X_test, y_train, y_test = train_test_split(X, y,
16                                                    test_size = 1/3,
17                                                    random_state = 42)
18
19
20 clf = DecisionTreeClassifier(criterion = 'entropy',
21                             max_leaf_nodes = 4,
22                             random_state = 42)
23
24 clf.fit(X_train, y_train)
25
26 plt.figure(figsize = (6,6))
27 tree.plot_tree(clf)
28 plt.show()

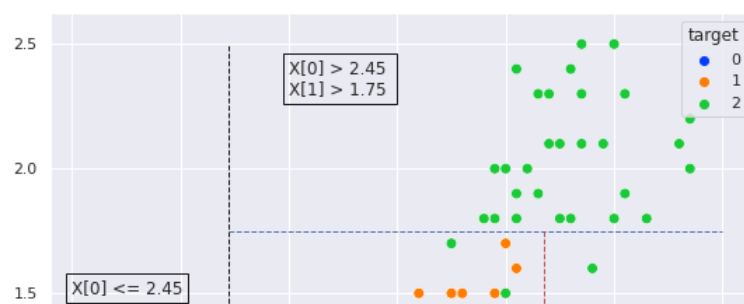
```

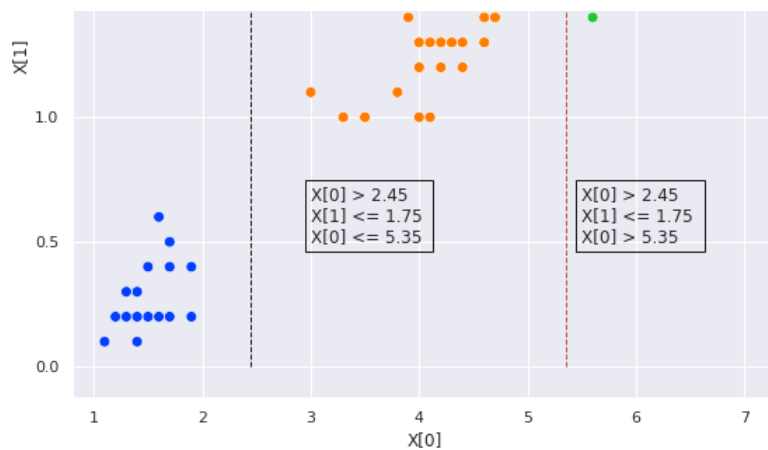


```

1 plt.figure(figsize = (9,9))
2 ax = plt.axes()
3
4 sns.scatterplot(x = X_train.petal_l, y = X_train.petal_w, hue = df.target, palette = 'magma')
5
6 ax.vlines(x = 2.45, ymin = 0, ymax = 2.5, linewidth = 1, color = 'k', linestyle = '--')
7 ax.text(1, 1.5, 'X[0] <= 2.45', fontsize = 12, bbox = dict(facecolor = 'none', edgecolor = 'k'))
8
9 ax.hlines(y = 1.75, xmin = 2.45, xmax = 7, linewidth = 1, color = 'b', linestyle = '--')
10 ax.text(3, 2.3, 'X[0] > 2.45 \nX[1] > 1.75', fontsize = 12, bbox = dict(facecolor = 'none', edgecolor = 'b'))
11
12 ax.vlines(x = 5.35, ymin = 0, ymax = 1.75, linewidth = 1, color = 'r', linestyle = '--')
13 ax.text(3, 0.5, 'X[0] > 2.45 \nX[1] <= 1.75 \nX[0] <= 5.35', fontsize = 12, bbox = dict(facecolor = 'none', edgecolor = 'r'))
14 ax.text(5.5, 0.5, 'X[0] > 2.45 \nX[1] <= 1.75 \nX[0] > 5.35', fontsize = 12, bbox = dict(facecolor = 'none', edgecolor = 'r'))
15
16 plt.xlabel('X[0]')
17 plt.ylabel('X[1]')
18
19 plt.show()

```





Как мы видим, нулевой, наиболее удаленный класс (синие точки) удалось отделить уже на первом шаге. По этому же принципу отделяются выбросы.

Лес изолирующих деревьев соответственно дает усредненное расстояние до каждой из точек.

### Показатель аномальности

Рассмотрим, как количественно выразить среднее расстояние до каждой из точек или **показатель аномальности** (anomaly score). Приведем формулу.

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}},$$

где

- $x$  — это конкретное наблюдение из общего числа  $n$  наблюдений; при этом
- $c(n)$  — это средний путь до листа в аналогичном по структуре двоичном дереве поиска (binary search tree, BST) с  $n$  наблюдений; а
- $E(h(x))$  — усредненное по всем деревьям расстояние до конкретного наблюдения  $x$ .

Нормализуя показатель  $E(h(x))$  по  $c(n)$  мы получаем, что:

- когда  $E(h(x)) \rightarrow c(n)$ ,  $s \rightarrow 0,5$ ;
- когда  $E(h(x)) \rightarrow 0$ ,  $s \rightarrow 1$ ;
- когда  $E(h(x)) \rightarrow n - 1$ ,  $s \rightarrow 0$ .

При этом расстояние  $h(x)$  находится в диапазоне  $(0, n - 1]$ , а  $s$  в диапазоне  $(0, 1]$ . Таким образом,

- если наблюдение имеет  $s$ , близкое к 1, это выброс;
- если наблюдение имеет  $s$ , близкое к 0, это не выброс;
- если  $s$  всех наблюдений близко к 0,5, тогда вся выборка не имеет выбросов.

Более подробно про этот алгоритм можно прочитать в публикации его создателей [Liu, Fei Tony, Ting, Kai Ming and Zhou, Zhi-Hua. "Isolation forest." Data Mining, 2008. ICDM'08. Eighth IEEE International Conference](#).

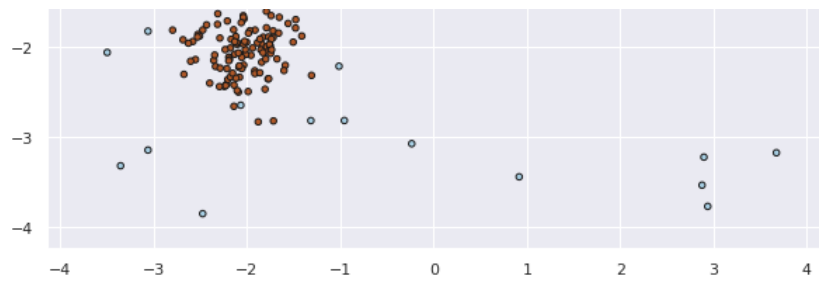
## IsolationForest в sklearn

### Пример из sklearn

Разберем [пример](#), приведенный на сайте sklearn. Вначале создадим синтетические данные с выбросами.

```
1  from sklearn.model_selection import train_test_split
2
3  # зададим количество обычных наблюдений и выбросов
4  n_samples, n_outliers = 120, 40
5  rng = np.random.RandomState(0)
6
7
8  # создадим вытянутое (за счет умножения на covariance)
9  covariance = np.array([[0.5, -0.1], [0.7, 0.4]])
10 # и сдвинутое вверх вправо
11 shift = np.array([2, 2])
12 # облако объектов
13 cluster_1 = 0.4 * rng.randn(n_samples, 2) @ covariance + shift
14
15 # создадим сферическое и сдвинутое вниз влево облако объектов
16 cluster_2 = 0.3 * rng.randn(n_samples, 2) + np.array([-2, -2])
17
18 # создадим выбросы
19 outliers = rng.uniform(low = -4, high = 4, size = (n_outliers, 2))
20
21 # создадим пространство из двух признаков
22 X = np.concatenate([cluster_1, cluster_2, outliers])
23
24 # а также целевую переменную (1 для обычных наблюдений, -1 для выбросов)
25 y = np.concatenate(
26     [np.ones((2 * n_samples), dtype = int),
27      -np.ones((n_outliers), dtype=int)]
28 )
29
30 scatter = plt.scatter(X[:, 0], X[:, 1],
31                       c = y,
32                       cmap = 'Paired',
33                       s = 20,
34                       edgecolor = 'k')
35
36 plt.title('Обычные наблюдения распределены нормально, \nвыбросы - равномерно')
37
38 plt.show()
```





Разделим выборку.

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y,
2                                                     stratify = y,
3                                                     random_state = 42)
4
5 # параметр stratify сделает так, что и в тестовой, и в обучающей выборке
6 # будет одинаковая доля выбросов
7 _, y_train_counts = np.unique(y_train, return_counts = True)
8 _, y_test_counts = np.unique(y_test, return_counts = True)
9
10 np.round(y_train_counts/len(y_train), 2), np.round(y_test_counts/len(y_test), 2)
```

Обучим класс изолирующего леса. Так как наблюдений мало, для каждого дерева будем брать все объекты обучающей выборки.

```
1 from sklearn.ensemble import IsolationForest
2
3 # обучим алгоритм
4 isof = IsolationForest(max_samples = len(X_train), random_state = 0)
5 isof.fit(X_train)
```

IsolationForest  
IsolationForest(max\_samples=210, random\_state=0)

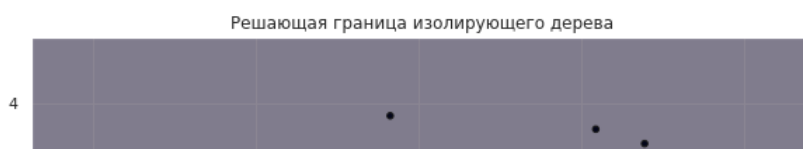
Сделаем прогноз на тесте и посмотрим на результат.

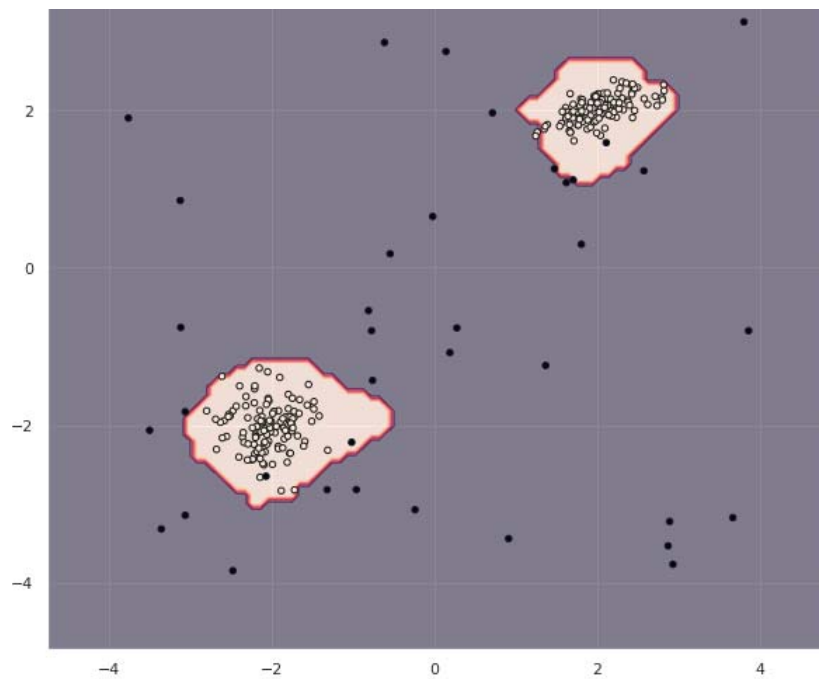
```
1 y_pred = isof.predict(X_test)
2
3 from sklearn.metrics import accuracy_score
4 accuracy_score(y_test, y_pred)
```

```
1 0.9428571428571428
```

Выведем решающую границу.

```
1 from sklearn.inspection import DecisionBoundaryDisplay
2
3 disp = DecisionBoundaryDisplay.from_estimator(
4     isof,
5     X,
6     response_method = 'predict',
7     alpha = 0.5,
8 )
9 disp.ax_.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
10 disp.ax_.set_title('Решающая граница изолирующего дерева')
11 plt.show()
```





### Настройка гиперпараметров

Разберем гиперпараметры модели:

- **n\_estimators** — количество деревьев в изолирующем лесу;
- **max\_samples** — количество наблюдений для обучения каждого изолирующего дерева;
- **max\_features** — количество признаков для обучения каждого дерева;
- **bootstrap** — получение выборки (sampling) с возвращением (True) или без (False);
- **contamination** — доля выбросов в датасете:
  - значение auto определяет выбросы так, как это было описано в публикации авторов, но с измененным порогом: anomaly score выброса отрицателен и близок к  $-1$ , anomaly\_score обычного наблюдения положителен и близок к  $0$ ;
  - кроме того, можно задать конкретную ожидаемую долю выбросов, например,  $0,1$  сделает выбросами 10 процентов наблюдений с самым высоким anomaly score.

Продemonстрируем особенности параметра contamination на простом примере. Возьмем небольшой набор данных и поочередно применим параметры contamination = 'auto', 0.1, 0.2. Несколько пояснений:

- метод **.predict()** помечает обычные наблюдения значением 1, выбросы значением  $-1$ ;
- метод **.decision\_function()** выдает anomaly\_score.

```
1 X_ = [[-1], [2], [3], [5], [7], [10], [12], [20], [30], [100]]
```

```
1 clf = IsolationForest(contamination = 'auto', random_state = 42).fit(X_)
2 print(clf.predict(X_))
3 print(clf.decision_function(X_))
```

```
1 [-1  1  1  1  1  1  1  1 -1 -1]
2 [-0.00403873  0.10617494  0.11864618  0.11188085  0.11479849  0.09281731
3    0.0780247  0.00948311 -0.08497048 -0.27336568]
```

```

1 clf = IsolationForest(contamination = 0.1, random_state = 42).fit(X_)
2 print(clf.predict(X_))
3 print(clf.decision_function(X_))

```

```

1 [ 1  1  1  1  1  1  1  1  1 -1]
2 [ 0.09977127  0.20998494  0.22245618  0.21569085  0.21860849  0.19662731
3  0.1818347  0.11329311  0.01883952 -0.16955568]

```

```

1 clf = IsolationForest(contamination = 0.2, random_state = 42).fit(X_)
2 print(clf.predict(X_))
3 print(clf.decision_function(X_))

```

```

1 [ 1  1  1  1  1  1  1  1 -1 -1]
2 [ 0.01618635  0.12640002  0.13887126  0.13210593  0.13502358  0.11304239
3  0.09824979  0.02970819 -0.0647454 -0.25314059]

```

## Датасет boston

Применим алгоритм изолирующего леса к датасету boston.

```

1 X_boston = boston.drop(columns = 'MEDV')
2 y_boston = boston.MEDV
3
4 clf = IsolationForest(max_samples = 100, random_state = 0)
5 clf.fit(X_boston)
6
7 # создадим столбец с anomaly_score
8 boston['scores'] = clf.decision_function(X_boston)
9 # и результатом (выброс (-1) или нет (1))
10 boston['anomaly'] = clf.predict(X_boston)
11
12 # посмотрим на количество выбросов
13 boston[boston.anomaly == -1].shape[0]

```

```

1 106

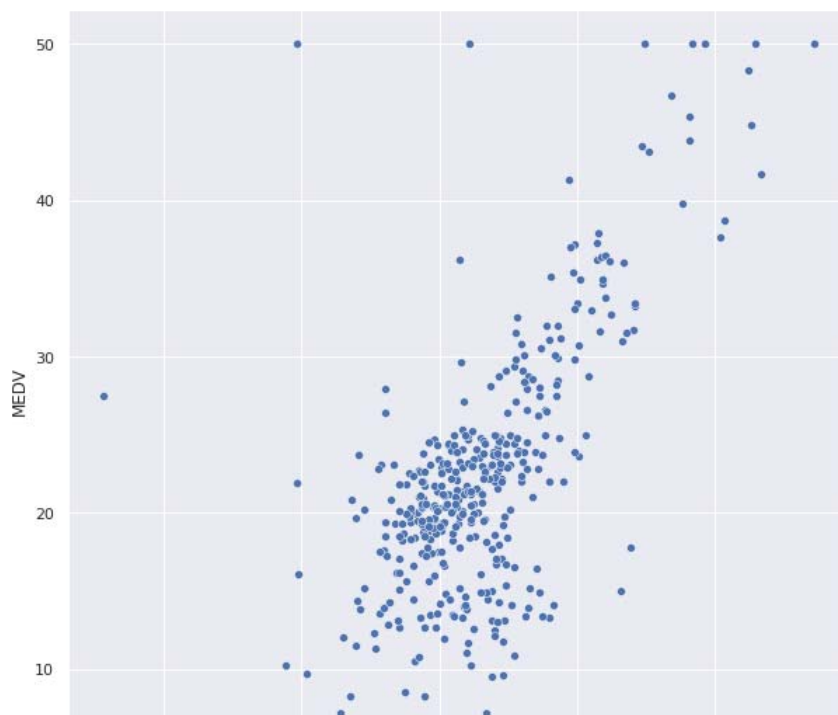
```

Посмотрим на взаимосвязь признака RM с целевой переменной после удаления выбросов.

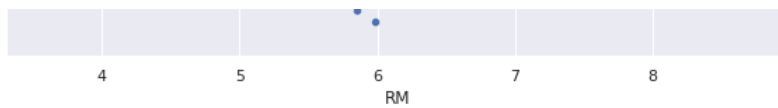
```

1 boston_ifor = boston[boston.anomaly == 1]
2 sns.scatterplot(x = boston_ifor.RM, y = boston_ifor.MEDV);

```



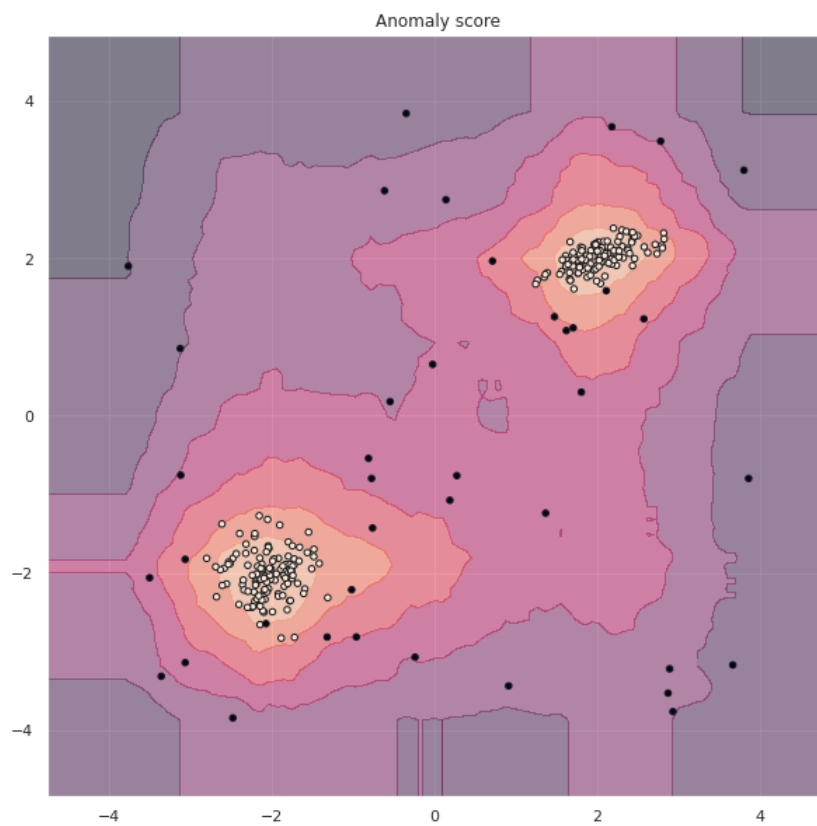





### Недостаток алгоритма

Для того чтобы увидеть недостаток алгоритма, рассмотрим тепловую карту, на которой цветом будут выводиться области с одинаковым anomaly score.

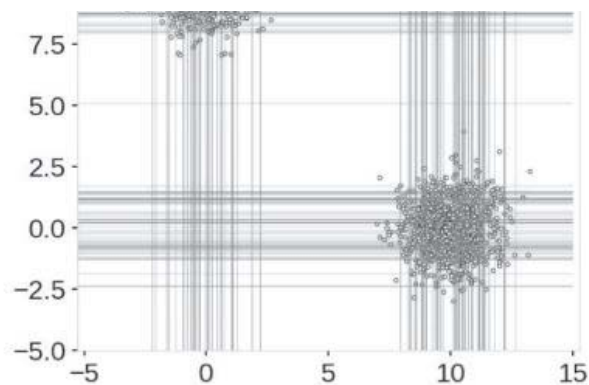
```
1 disp = DecisionBoundaryDisplay.from_estimator(
2     isof,
3     X,
4     response_method = 'decision_function',
5     alpha = 0.5,
6 )
7 disp.ax_.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
8 disp.ax_.set_title('Anomaly score')
9 plt.show()
```



Как вы видите, в верхнем левом и нижнем правом углу также находятся области (их называют *ghost areas*), в которых объекты могли бы быть классифицированы как обычные наблюдения, хотя это неправильно (ложноположительный результат). Это связано с тем, что границы, проводимые алгоритмом параллельны осям координат.

Приведем иллюстрацию из [статьи](#)  исследователей, усовершенствовавших этот алгоритм.





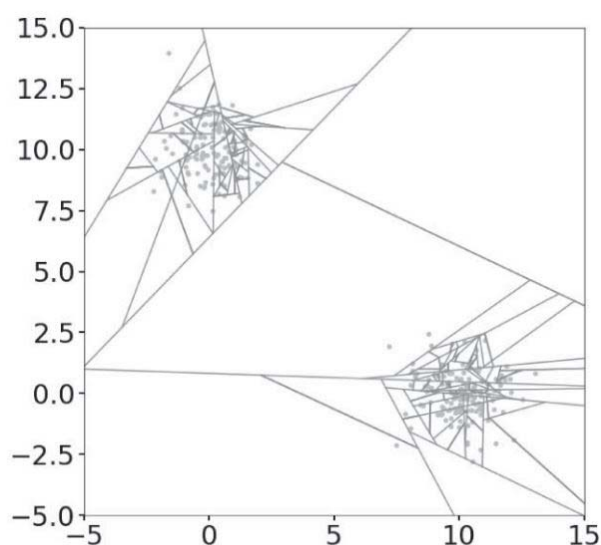
Для решения этой проблемы был предложен расширенный алгоритм изолирующего леса.

## Extended Isolation Forest

**Расширенный алгоритм изолирующего леса** (Extended Isolation Forest) делит пространство с помощью случайных, не обязательно параллельных осей координат, гиперплоскостей.

Авторами этого алгоритма являются [Sahand Hariri, Matias Carrasco Kind и Rober J. Brunner](#).

Для данных на предыдущей иллюстрации разделение с помощью расширенного алгоритма могло бы выглядеть вот так.



Рассмотрим работу этого алгоритма на практике. Приведу две библиотеки, в которых реализован Extended Isolation Forest:

- [библиотека eif](#), созданная авторами алгоритма; и
- Extended Isolation Forest на [платформе h2o](#).

Используем второй вариант.

## Платформа h2o

Установим h2o в Google Colab.

```
1 !pip install h2o
```

Запустим сервер h2o.

```
1 import h2o
2
3 h2o.init()
```

```
1 Checking whether there is an H2O instance running at http://localhost:54321 ..... not
2 Attempting to start a local H2O server...
3   Java Version: openjdk version "11.0.17" 2022-10-18; OpenJDK Runtime Environment (bu
4   Starting server from /usr/local/lib/python3.8/dist-packages/h2o/backend/bin/h2o.jar
5   Ice root: /tmp/tmpztb87_pq
6   JVM stdout: /tmp/tmpztb87_pq/h2o_unknownUser_started_from_python.out
7   JVM stderr: /tmp/tmpztb87_pq/h2o_unknownUser_started_from_python.err
8   Server is running at http://127.0.0.1:54321
9   Connecting to H2O server at http://127.0.0.1:54321 ... successful.
10  H2O_cluster_uptime:    03 secs
11  H2O_cluster_timezone:  Etc/UTC
12  H2O_data_parsing_timezone:  UTC
13  H2O_cluster_version:   3.38.0.3
14  H2O_cluster_version_age:  1 month and 1 day
15  H2O_cluster_name:       H2O_from_python_unknownUser_tcusbr
16  H2O_cluster_total_nodes:  1
17  H2O_cluster_free_memory:  3.172 Gb
18  H2O_cluster_total_cores:  2
19  H2O_cluster_allowed_cores:  2
20  H2O_cluster_status:      locked, healthy
21  H2O_connection_url:      http://127.0.0.1:54321
22  H2O_connection_proxy:    {"http": null, "https": null}
23  H2O_internal_security:    False
24  Python_version:          3.8.16 final
```

## Обучение алгоритмов

Импортируем класс `алгоритма`, создадим объекты этого класса для модели обычного изолирующего леса (без наклона гиперплоскостей) и для модели с максимальным наклоном гиперплоскостей.

Максимальный наклон задается параметром `extension_level` и находится в диапазоне  $[0, P - 1]$ , где  $P$  — это количество признаков. В нашем случае признаков два, поэтому для расширенного алгоритма используем `extension_level = 1`.

```
1 # импортируем класс Extended Isolation Forest
2 from h2o.estimators import H2OExtendedIsolationForestEstimator
3
4 # зададим основные параметры алгоритмов
5 ntrees = 400
6 sample_size = len(X)
7 seed = 42
8
9 # создадим специальный h2o датафрейм
10 training_frame = h2o.H2OFrame(X)
11
12 # создадим класс обычного изолирующего леса
13 IF_h2o = H2OExtendedIsolationForestEstimator(
14     model_id = 'isolation_forest',
15     ntrees = ntrees,
16     sample_size = sample_size,
17     extension_level = 0,
18     seed = seed
19 )
```

```

1 Parse progress: |████████████████████████████████████████| (dc
2 extendedisolationforest Model Build progress: |████████████████████████████████████████| (dc
3 extendedisolationforest Model Build progress: |████████████████████████████████████████| (dc
4 Model Details
5 =====
6 H2OExtendedIsolationForestEstimator : Extended Isolation Forest
7 Model Key: isolation_forest
8
9
10 Model Summary:
11     number_of_trees    size_of_subsample    extension_level    seed    number_of_train
12 -- -----
13     400                280                0              42      400
14
15 ModelMetricsAnomaly: extendedisolationforest
16 ** Reported on train data. **
17
18 Anomaly Score: 13.588221227545635
19 Normalized Anomaly Score: 0.4106598976735543
20 Model Details
21 =====
22 H2OExtendedIsolationForestEstimator : Extended Isolation Forest
23 Model Key: extended_isolation_forest
24
25
26 Model Summary:
27     number_of_trees    size_of_subsample    extension_level    seed    number_of_train
28 -- -----
29     400                280                1              42      400
30
31 ModelMetricsAnomaly: extendedisolationforest
32 ** Reported on train data. **
33
34 Anomaly Score: 14.73433083067248
35 Normalized Anomaly Score: 0.3791101368673747

```

Так как Extended Isolation Forest — это алгоритм без учителя (метод `.predict()` выдает `anomaly_score`), мы не можем напрямую посчитать точность прогноза. С другой стороны, так как в созданных нами данных есть разметка, мы можем провести косвенное сравнение.

Начнем с алгоритма **обычного изолирующего леса**. Сделаем прогноз.

```
1 # посмотрим на результат
2 h2o_anomaly_score_if_df.head()
```

	anomaly_score	mean_length
0	0.414619	13.239968
1	0.503174	10.328823
2	0.405333	13.580600
3	0.381291	14.500156
4	0.376005	14.710097

```
1 data = pd.DataFrame(X, columns = ['x1', 'x2'])
2 data['target'] = y
```

```
1 # выберем количество наблюдений
2 sample = 60
3
4 # для наглядности рассчитаем долю от общего числа наблюдений
5 sample / len(X)
```

1	0.21428571428571427
---	---------------------

```
1 if_df = pd.concat([data, h2o_anomaly_score_if_df], axis = 1)
2 if_df.sort_values(by = 'anomaly_score', ascending = False, inplace = True)
3 np.unique(if_df.iloc[:sample, 2], return_counts = True)
```

```
1 (array([-1,  1]), array([39, 21]))
```

```
1 | h2o anomaly score eif = EIF h2o.predict(training frame)
```

Расширенный алгоритм сделал чуть менее качественный прогноз.

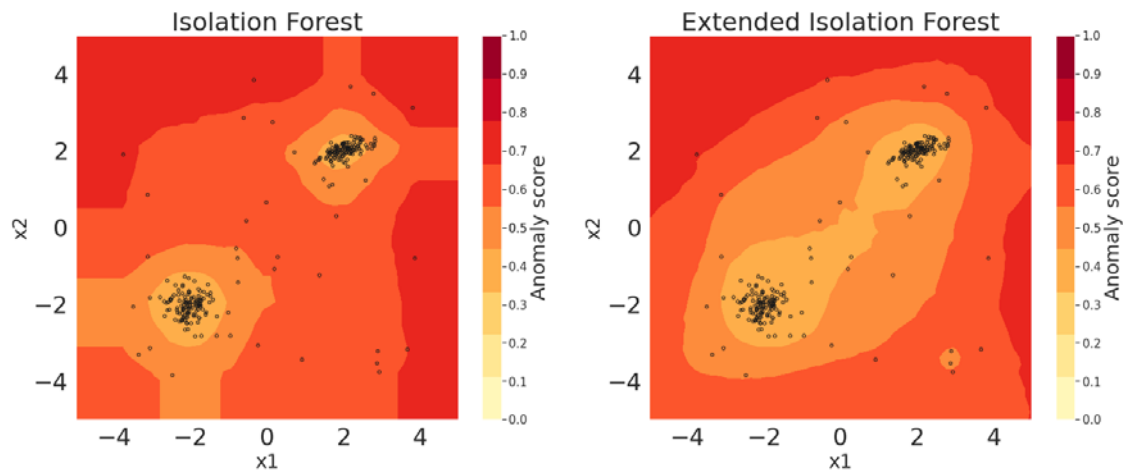
Сравним тепловые карты обоих алгоритмов.

```

1 f = plt.figure(figsize=(24, 9))
2
3 # объявим функцию для вывода подграфиков
4 def plot_heatmap(heatmap_data, subplot, title):
5     ax1 = f.add_subplot(subplot)
6     levels = np.linspace(0,1,10, endpoint = True)
7     v = np.linspace(0, 1, 12, endpoint = True)
8     v = np.around(v, decimals = 1)
9     CS = ax1.contourf(xx, yy, heatmap_data, levels, cmap = plt.cm.YlOrRd)
10    cbar = plt.colorbar(CS, ticks = v)
11    cbar.ax.set_ylabel('Anomaly score', fontsize = 25)
12    cbar.ax.tick_params(labelsize = 15)
13    ax1.set_xlabel('x1', fontsize = 25)
14    ax1.set_ylabel('x2', fontsize = 25)
15    plt.tick_params(labelsize=30)
16    plt.scatter(X[:,0],X[:,1],s=15,c='None',edgecolor='k')
17    plt.axis("equal")
18    plt.title(title, fontsize=32)
19
20 # выведем тепловые карты
21 plot_heatmap(heatmap_h2o_if, 121, 'Isolation Forest')
22 plot_heatmap(heatmap_h2o_eif, 122, 'Extended Isolation Forest')

```

```
23 |  
24 | plt.show()
```



В данном случае, как мы видим, расширенному алгоритму не удалось сформировать два обособленных кластера.

## Подведем итог

На сегодняшнем занятии мы разобрали статистические методы выявления выбросов (boxplot, scatter plot, z-score и 1,5 IQR), а также два метода, основанные на модели (алгоритмы обычного и расширенного изолирующего леса).